multipath

FMS(TM)

FAST MATRIX SOLVER

MANUAL

2025

[A]{X}={B}

Now Including **Dashboard** and **MatrixWarrior**

To navigate this manual, click on the links in the upper-right corner.

For example, to begin, click on the Table of Contents.

Copyright © Multipath Corporation

FMS MANUAL TABLE OF CONTENTS

1. Introduction
   - 1.1. ***FMS***lib
   - 1.2. **Dashboard**
   - 1.3. **MatrixWarrior**
2. ***FMS***lib
3. Installing ***FMS*** In Your Application Program
   - 3.1. Calling ***FMS*** Subroutines
   - 3.2. Using ***FMS*** Parameters
4. ***FMS*** Advanced Features
   - 4.1. Pivoting
   - 4.2. Computing Eigenvalues
   - 4.3. Substructuring (***FMS*** Parameter NEQSUB)
   - 4.4. Nonlinear Reassembly (***FMS*** Parameter LOWASM)
   - 4.5. Force Symmetry (***FMS*** Parameter LUOK)
   - 4.6. Strassen and Winograd Algorithms (***FMS*** Parameter IALGOR)
5. Examples of Using ***FMS***
6. Achieving Maximum Performance With ***FMS***
   - 6.1. Balancing Processing and Disk Speed with Memory Size
   - 6.2. Time and Storage Requirements
   - 6.3. Data Alignment
   - 6.4. Memory Management
   - 6.5. File Striping
   - 6.6. Non-Uniform Memory Access (NUMA) Machines
   - 6.7. Direct I/O
   - 6.8. Large Pages
7. **Dashboard** Performance Monitor
8. **MatrixWarrior** Application

A. ***FMS*** Theory

B. ***FMS*** Files
   - B.1. Vector Files
   - B.2. Submatrix Files
   - B.3. Matrix Files

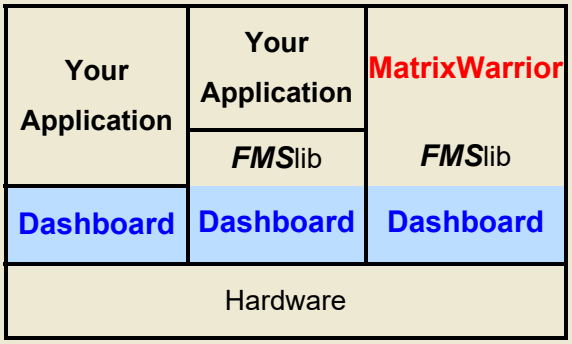
C. ***FMS*** Subroutines

D. ***FMS*** Parameters
E. Error Handling
F. Program DEMO
G. Environment Variables
H. List of Figures

Copyright © Multipath Corporation

1. INTRODUCTION

Welcome to Multipath's manual. Here you will find all the information necessary to incorporate ***FMS***lib and **Dashboard** in your application, or run **MatrixWarrior** to test your hardware.



This software collection consists of the following related products:

1. ***FMS***lib, a matrix algebra library designed for optimum performance on large problems;
2. **Dashboard**, a performance analysis tool which generates a website on your computer showing hardware and performance information; and
3. **MatrixWarrior**, an application which uses ***FMS***lib to test your hardware and **Dashboard** to display the results.

1. *FMS*LIB

FMS (Fast Matrix Solver) is the industry standard for performing matrix algebra operations on large, dense matrices and groups of vectors. Packaged as a FORTRAN or C callable library, ***FMS*** may be incorporated into new or existing scientific and engineering programs to improve performance and provide large problem solving capability.

FMS Features

The primary function of ***FMS*** is to solve the following system of simultaneous equations:

`[A]{X} = {B}`

where:

`[A]` is an N-by-N coefficient matrix,

`{B}` is one or more right-hand side vectors,

`{X}` represents the solution vectors to be determined.

FMS offers the following significant developments that enable computers to sustain their maximum speed throughout the solution process:

- ***FMS*** is based on mathematical formulations specifically designed to exploit parallel and pipeline architectures. All data structures and calculation sequences make maximum use of memory and arithmetic pipelines. This top-down approach to the development of ***FMS*** results in significantly better performance than

can be obtained by performing local optimizations to existing algorithms.

- ***FMS*** automatically takes advantage of parallel processing and GPU units when they are available.
- ***FMS*** is designed from the machine's point of view. Where appropriate, performance-critical subroutines are coded directly in machine language. Maximum reuse is made of registers, cache and memory.
- ***FMS*** performs data transfers to disks in parallel with processing. These asynchronous data transfers allow ***FMS*** to solve extremely large problems at maximum speed without requiring excessive memory.
- ***FMS*** operates multiple disks in parallel to increase disk transfer rates (file striping).

FMS Modules

Storage and processing requirements depend significantly on matrix symmetry and data type, so ***FMS*** subroutines are divided into five modules, as shown in the following table:

FMS Subroutine Modules

MODULE	DATA TYPE	SYMMETRY	TYPICAL APPLICATION
RS	Real	Symmetric	Structural Analysis Heat Transfer
RN	Real	Nonsymmetric	Diffusion Fluid Mechanics
CH	Complex	Hermitian	Chemistry
CS	Complex	Symmetric	Radar Cross-Section Acoustics
CN	Complex	Nonsymmetric	Electromagnetics Circuit Design

Each module is optimized to provide maximum performance and minimum data storage for the type of problem it is designed to solve.

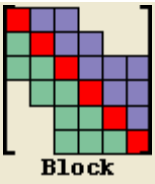
FMS Matrix Formats

FMS is actually 3 out-of-core solvers in 1:

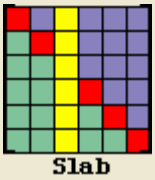


PROFILE SOLVER: Accounts for the sparsity of matrix **[A]** on an equation by equation basis. Typical applications include finite element and finite difference programs.

BLOCK SOLVER: Divides the matrix **[A]** into square blocks, accounting for sparsity on a block



by block basis. This solver uses industry standard BLAS3 kernels and provides excellent I/O performance on extremely large problems. Typical applications include boundary integral and moment method programs.



SLAB SOLVER: Extends the functionality of the Block Solver by providing full column partial pivoting for full nonsymmetric matrices.

Because all 3 ***FMS*** solvers share the same application program interface, you may switch between solvers by simply changing a Parameter value, without recompiling or relinking your program.

Math Function Overview

FMS solves the system of simultaneous equations in two stages. First, the matrix **[A]** is factored into one of the following forms, depending on the symmetry of **[A]**:

Matrix Factoring

OPERATION	MATRIX SYMMETRY	
	SYMMETRIC	NONSYMMETRIC
MATRIX FACTORING	$[L][D][L]^T = [A]$	$[L][U] = [A]$

[L] is a lower triangular, **[D]** a diagonal, and **[U]** an upper triangular matrix. $[L]^T$ is the transpose of matrix **[L]**. For Hermitian matrices, $[L]^T$ will be used for the transposed conjugate of **[L]**. ***FMS*** uses data structures that permit the factors **[L]**, **[D]**, and **[U]** to be overlaid on the original matrix **[A]** to conserve memory and disk space.

Because factoring does not depend on the right-hand side vectors **{B}**, systems having multiple solution vectors are processed very efficiently. Compared to iterative methods, where all calculations must be repeated for each solution, this direct approach provides increased efficiency as the number of solution vectors increases.

If all the solution vectors are not known during the first solution, the factors may be saved and used later.

The second stage is to solve for the solution vectors **{X}**. This calculation is divided into the following steps:

Vector Solution

OPERATION	MATRIX SYMMETRY	
	SYMMETRIC	NONSYMMETRIC

FORWARD REDUCTION	`[L]{Y} = {B}`	`[L]{Y} = {B}`
DIAGONAL SCALING	`[D]{Z} = {Y}`	Not Required
BACK SUBSTITUTION	$[L]^T\{X\} = \{Z\}$	`[U]{X} = {Y}`

The first step is forward reduction, which computes the intermediate vectors **{Y}** from the right-hand side vectors **{B}** and lower triangular matrix factor **[L]**. The same calculation is performed for both symmetric and nonsymmetric matrices.

The second step is diagonal scaling. Because symmetric problems store only half the matrix, this calculation must be performed as a separate step. Nonsymmetric problems perform diagonal scaling during back substitution.

The final step is back substitution, which computes the solution vectors **{X}**. ***FMS*** permits the intermediate and solution vectors to be overlaid on the original right-hand side vectors.

In addition to solving simultaneous equations, ***FMS*** performs other matrix algebra operations. Included are matrix and vector multiply calculations typically used to compute Eigenvalues and Eigenvectors of the matrix **[A]**.

The following matrix-vector multiply calculation multiplies one or more vectors **{X}** by a matrix **[A]** stored in ***FMS*** format:

{Y} = [A] {X}

The following vectors-vectors multiply calculation computes the inner products of groups of vectors and stores the results in the full matrix **[F]**:

[F] = {X}T {Y}

For diagonal matrices **[D]**, ***FMS*** computes the quadratic form directly with the following equation:

[F] = {X}T [D] {Y}

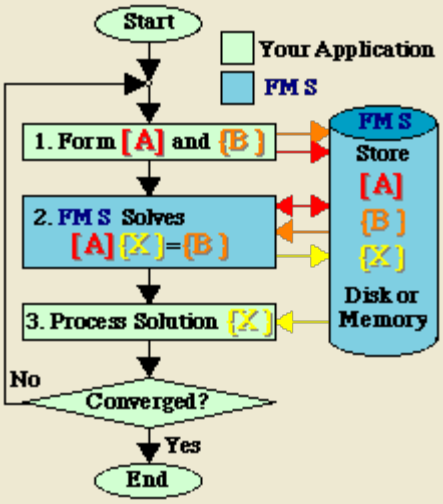
FMS also provides for combining and scaling vectors with the vectors-matrix multiply operation:

{Y} = {X} [F]

Ease of Use

FMS is designed to be easy to incorporate into your application program. The following flowchart shows where ***FMS*** is interfaced to your application.

- **Step 1:**
  Matrix coefficients are transferred to the ***FMS*** Database as they are generated, by rows, columns, blocks or finite elements. ***FMS*** then assembles your data into matrix **[A]**, which is stored in memory or on



disk. Vector **{B}** may be transferred to ***FMS*** for storage, or ***FMS*** may be instructed to operate directly on the vector **{B}** in your program.

- **Step 2:**

  FMS solves the system of equations for the solution vector(s) **{X}**, which may be stored directly in your program or read from ***FMS***.

- **Step 3:**

  Your application processes the solution vector(s) **{X}**.

For nonlinear analysis, where the coefficients in matrix **[A]** depend on the solution values in **{X}**, the process is repeated until convergence is obtained. ***FMS*** contains special features to reduce the number of operations in step 2 when only some of the matrix coefficients are updated.

For most applications, the matrix **[A]** will far exceed the memory in the machine. Therefore the terms in matrix **[A]** must be transferred into the ***FMS*** Database as they are generated, rather than as a full array A(N,N).

Defining Matrix Data

Based on years of practical experience, the following options have been developed in ***FMS*** for defining matrix terms:

- **Rows or Columns**

  For applications which generate the matrix data by rows or columns, ***FMS*** includes the option to transfer each row or column as it is computed.

- **Finite Elements**

  FMS includes the matrix assembly phase of Finite Element programs. The element data is transferred to ***FMS*** as it is computed. ***FMS*** then assembles the global matrix.

- **Blocks**

  You may transfer sub-arrays of the matrix from storage in your application to the ***FMS*** database. Each call defines a window of the matrix. Only nonzero matrix terms need to be defined.

- **Full Matrix in Memory**

  For full matrices residing in memory, ***FMS*** includes subroutines which operate directly on your data.

- **Callback Subroutines**

  You may direct ***FMS*** to call subroutines you provide to define or modify the matrix data. Each call defines a window of the matrix.

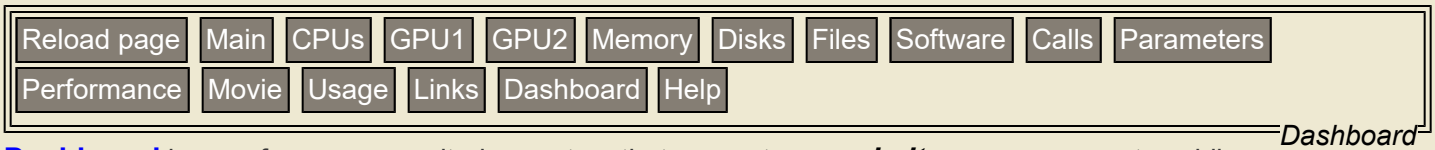
In a Hurry?

If you are anxious to get started inserting ***FMS*** in your application, proceed as follows:

1. Skim over Chapter 3, "Installing ***FMS*** In Your Application Program". You will learn how ***FMS*** subroutines are organized and the role of ***FMS*** Parameters.
2. Skip to Chapter 5, "Examples of Using ***FMS***". There is probably an example that is close to your application. Each ***FMS*** subroutine and Parameter referenced in these examples has been linked back to the detailed description. The source code is also available without the hyperlinks so you can compile and run the examples.

2. DASHBOARD

A Performance tool for your application



Dashboard

Dashboard is a performance-monitoring system that generates a ***website*** on your computer while your application is running. The generated site includes separate pages which provide detailed information on the CPUs, GPUs, Memory, Disks, File system, Software, a Call-tree and ***FMS*** Parameter values. **Dashboard** is easily installed in any application, even those that do not use ***FMS***lib for matrix algebra.

You may use **Dashboard** in one of three ways:

1. Call **Dashboard** directly from your application;
2. Call ***FMS***lib from your application, which is already instrumented with **Dashboard**; or
3. Run **MatrixWarrior**, which includes ***FMS***lib and therefore also includes **Dashboard**.

For applications which use ***FMS***lib, **Dashboard** includes a Performance page specifically designed for matrix algebra. This page shows the current state of the computation and the performance of critical hardware components. These reports may be linked to form a "movie" of the calculation. Multipath's Home Page contains a link to a movie produced by **Dashboard**.

3. MATRIXWARRIOR

As libraries, ***FMS***lib and **Dashboard** receive data from your application and drive the computer hardware. In order to demonstrate the performance of ***FMS***lib and the displays available with **Dashboard**, a test application **MatrixWarrior** is included. This application generates matrix **`[A]`** and vector **`{B}`** according to your specifications. ***FMS***lib then solves the system of equations **`[A]{X} = {B}`** for the solution **`{X}`**. Because **MatrixWarrior** is layered on ***FMS***lib and **Dashboard**, all the features available in these libraries can be demonstrated by **MatrixWarrior**.

MatrixWarrior is an application that requires no programming. It can be used to:

- Measure the actual performance and tune your machine(s);
- Evaluate machines and options prior to purchase;
- Tune and burn-in your machines after purchase;
- Validate new machine designs before production;
- Check possible hardware issues in production using a known application; and
- Learn about the ***FMS*** and **Dashboard** libraries, which can be incorporated into your application to provide performance benefits and useful displays of information.

MatrixWarrior is available as a free download. Versions are available for Linux and Windows. Download **MatrixWarrior** to test your machine's performance and learn more about ***FMS***lib and **Dashboard**.

Copyright © Multipath Corporation

2. FMSLIB

The Fast Matrix Solver (***FMS***) is a software product that enables computers to achieve their maximum performance by continuously utilizing all hardware components at maximum capacity. Designed from a hardware point of view, ***FMS*** takes maximum advantage of high speed registers, cache, memory, disks, accelerators and parallelism. Useful in a wide class of application programs, ***FMS*** significantly extends capability by increasing the maximum problem size and improving the accuracy obtained by computer analysis.

What Type of Problem Does FMS Solve?

Computers are used extensively in science and engineering to predict the behavior of products before they are built. The computer analysis helps in optimizing the design, improving performance and lowering cost.

The computerized analysis is based on the division of the problem into a large number of pieces. Within each piece, a model is built which describes the behavior of that piece. The individual pieces are then assembled together to form an overall model of the product.

As the number of pieces used to model the product is increased, the approximate solution obtained from the computer approaches the exact solution of the original problem. The maximum number of pieces that can be used, and therefore the accuracy of the analysis, is determined by the capacity of the computer hardware and software.

This modeling process is often called discretization, because the original problem is replaced by a finite number of discrete pieces. Analysis programs based on this approach include finite element, finite difference, boundary integral and moment method techniques. These formulations exploit the strength of digital computers, which perform large numbers of similar tasks rapidly.

Mathematically this simulation involves the solution of a large system of simultaneous equations

`[A]{X} = {B}`

where

`[A]` is an N-by-N coefficient matrix that represents the product being analyzed,

`{B}` is an N-by-1 right-hand side vector that represents the environment that the product is operating in, and

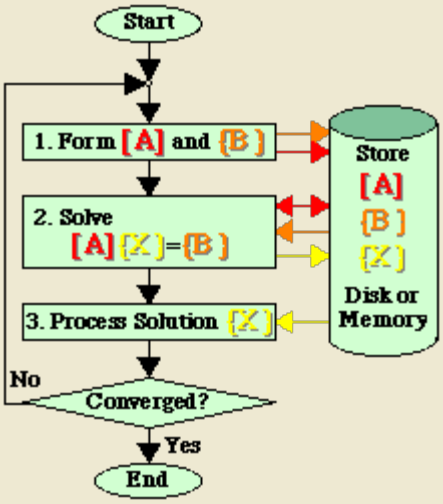
`{X}` is an N-by-1 solution vector being computed, which represents the response of the product to the operating environment.

As the size of the matrix and vectors N is increased, the solution obtained by the computer approaches the exact solution of the problem.

What Limits N?

We want to make N as large as possible, so the computer simulation closely resembles the product. In order to understand what limits N, let us look at this process in more detail.

The following flowchart illustrates the steps involved in performing this type of computer simulation.



1. Form [A] and {B}

The process begins by forming local equations for each of the discrete pieces. These equations are in the form of contributions to the matrix **[A]** and right-hand side vector **{B}**. For large problems, this data is usually written to disk as it is generated.

2. Solve [A]{X} = {B}

The local equations for each piece are then combined to form the global system of equations **[A]{X}={B}**, which is solved for the solution **{X}**.

3. Process solution {X}

After the global solution is complete, **{X}** is used to compute local results of interest.

If the operating environment is severe, then the solution values in **{X}** will change the original model represented by **[A]** (nonlinear analysis). In addition, if the solution **{X}** indicates a bad product design, the process must be repeated (design optimization). In either case, a new matrix **[A]** is formed and the solution is repeated until convergence is obtained.

As we increase N, the time and storage requirements for Steps 1 and 3 increase differently than the requirements for Step 2. To illustrate this point, let us look at a model using a full complex nonsymmetric matrix **[A]** run on a computer operation at 100 Mflops. In this example, we assume that the time and storage requirements for Steps 1 and 3 are the same as Step 2 when N is 1000.

N is limited by processing time

As we increase N, the time required for Steps 1 and 3 increases between linearly and quadratically. If we use twice as many pieces it will take twice as long to compute their contribution. However for fully coupled problems the interaction of the pieces must be computed, introducing a quadratic component.

In Step 2, however, the scaling is drastically different. The time required to solve the system **[A]{X}={B}** increases as the cube of N. As a result, the maximum size of N on any computer is often limited by its floating point performance.

N is limited by storage requirements

Similarly, as we increase N the space required for storage by Steps 1 and 3 increase linearly. However, because **[A]** is an N-by-N array, the storage requirements for Step 2 increase as the square of N.

The address space or storage capacity of a computer, memory or disk, can also limit the maximum value of N.

Modern computers use a hierarchical memory system, storing small amounts of data in high-speed registers close to the processor, working out to multiple levels of cache, memory and disk. Each level of storage offers more space but operates at a slower speed. If the data is not staged properly the performance will significantly degrade to the speed of the slowest storage device.

The Solution

Fortunately there are several things that can be done to solve the above problems and allow large values of N.

Focus on [A]{X}={B}

The obstacles to increasing N listed above all occur in Step 2, the solution of **`[A]{X}={B}`**. Therefore we can focus on a well defined mathematical problem that is generic to all modeling techniques.

Data Reuse

We noted above that the solution of **`[A]{X}={B}`** required N cubed operations on N squared pieces of data. As a result, each piece of data is "reused" N times. We can use this property to design special software that stages data from disk to memory, cache and registers. When accelerators are present, part of the data may also staged from memory to the accelerators. At each level of storage, the data is reused as much as possible. This reduces the total amount of data being transferred.

Parallel Processing

Today's high speed computers contain multiple processors. By designing software to operate these processors in parallel, the time required for solution can be significantly reduced.

Accelerators

Accelerators initially developed for graphic rendering have direct application for speeding up performance of matrix algebra. The large number of operations which can be performed in parallel can fully exploit these architectures.

Disks

Storing data on disks costs about 100 times less than memory. In addition, disks may easily be added while memory is limited by the density of the memory chips and the number of memory sockets. Usually increasing memory requires replacing all the memory chips with those of a newer higher density.
With proper algorithm design that stages data, the high reuse in matrix algebra allows the slower transfer of data from disk to be properly matched with processing at a significantly higher rate.

File Striping

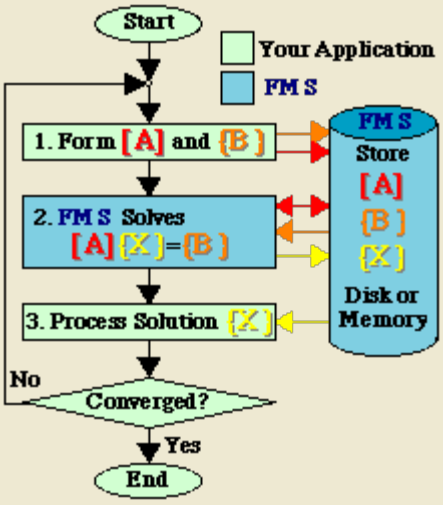
When N is large, multiple disks will be required to store **`[A]`**. These disks can be operated in parallel to increase the rate at which data is transferred to and from memory.

Asynchronous I/O

By default, when an application requires data from disk, processing is suspended until the data has been transferred. If the disks and processors are operating at the same speed, this can double the solution time.
By developing special software to read data ahead of processing, data will be available when required by the processors. As a result, the disks and processors will run concurrently and no time will be lost waiting for data.

How FMS implements this technology

The solutions listed above all depend on special software. That is exactly what the Fast Matrix Solver (***FMS***) is designed to do.
The following figure shows the original analysis program with the equation solution phase replaced by ***FMS***.



1. Form [A] and {B}

FMS captures the matrix `[A]` and right-hand side vector `{B}` as they are generated. Parts of `[A]` may be transferred by rows, columns, blocks or finite elements. This data is stored directly in the ***FMS*** Database. Depending on the problem size, this database may physically reside in memory or on disk.

2. Solve [A]{X} = {B}

All operations involving the matrix `[A]` and vectors `{B}` and `{X}` are performed by ***FMS***, which is highly optimized for each computer system.

3. Process solution {X}

FMS passes the solution vector `{X}` back to your application program for further processing.

Steps 1 and 3 of this analysis depend on the physics of the problem and the modeling technique used. Software and hardware efficiency are not a critical issue. By contrast, Step 2 is independent of the physics and modeling technique, but pushes the hardware to the limit. For this reason, ***FMS*** was designed from a hardware point of view.

There is an optimized version of ***FMS*** for most computer platforms. Each version makes maximum reuse of registers, cache, memory, accelerators (when available) and disks. Processors, accelerators and disks are operated in parallel to reduce runtime. On all systems, disks are operated in parallel and transfer data concurrently with processing.

While the internals of ***FMS*** are machine-specific, the interface to your application program is generic. This allows you to maintain a single version of your application while achieving peak performance on each computer system.

Wherever possible, ***FMS*** detects the available hardware and dynamically loads the supporting libraries as required. This allows you to maintain a single binary version of your application and run on a variety of hardware configurations.

FMS is High Technology Middleware

Today's computer industry consists of multiple vendors, with each vendor specializing in the products they produce. When you purchase a computer system, you are obtaining a product that was built by a team of specialists.

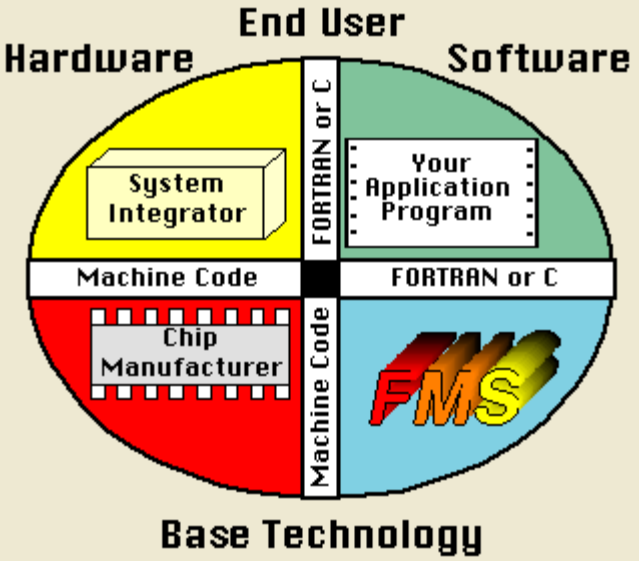
This team may be sorted by type of technology, hardware and software, as shown in the following figure:

System Integrator

They integrate the hardware and software into a system you can use. Also included is the operating system, which may be purchased separately.

Applications

These are programs that you either develop or purchase. They perform a specific function, from word processing to sophisticated scientific analysis.



Chip Manufacturer

Current fabrication technology requires extremely expensive facilities to produce the fast, high density processor and memory components of today's computers. Companies specialize in this technology, producing a variety of components for various system integrators.

Middleware

At the base technology level in software is an industry called middleware. This is software that is closely coupled to the hardware and performs a generic function to a variety of applications. Database software is a typical middleware product.

FMS is middleware that combines database and math library functions. It extends the operating system by providing file striping, asynchronous I/O, memory management and parallel processing. ***FMS*** also interacts with the hardware at a low level with optimized kernels designed for each specific platform. When hardware accelerators are detected, they are automatically used to improve performance and reduce runtime.

Called as a library from programs written in FORTRAN or C, ***FMS*** provides a back door to the chip and operating system technology not normally available in programming languages. ***FMS*** may also be called from c++. In this case it is best viewed as a single managed object.

While the internals of ***FMS*** and its interface to the machine are constantly changing to take advantage of new technology, the application program interface has remained constant. This allows your applications to take advantage of new hardware technology with no development effort.

With today's world economy, maintaining a competitive advantage is critical. Products must continuously be refined to take advantage of new materials, manufacturing techniques and changing customer requirements. Computer simulations are well recognized as a vital ingredient in meeting these demands. By pushing the envelope of computer performance, ***FMS*** provides a technical advantage that helps you meet your goals.

Copyright © Multipath Corporation

3. INSTALLING FMS IN YOUR APPLICATION PROGRAM

This chapter is the guide for installing ***FMS*** in your application. It is an overview, with the details of individual subroutines provided by hyperlinks.
The interface between your application and ***FMS*** consists of the following three items:

1. Subroutine Calls

These are ***FMS*** subroutines that you call from your application. This manual includes a complete list of the ***FMS*** subroutines.

2. FMS Parameters

These are scalar variables that provide "special effects". ***FMS*** initializes them to default values. For most applications you do not need to change these parameters to get ***FMS*** running.
FMS Parameters are used to control:

- print output level and format
- machine configuration
- file management
- error handling
- less frequently used subroutine options.

This manual includes a complete list of the ***FMS*** Parameters.

3. FMS Files

Your matrix and vector data is maintained by ***FMS*** in the ***FMS*** Database. Depending on your problem size and machine configuration, ***FMS*** physically stores this database in memory, disk or both.
You create entries into the ***FMS*** Database by opening ***FMS*** files. Several options are provided to transfer data between your application and the ***FMS*** database, including read and write utilities.

With this background, we are now ready to proceed with installing ***FMS*** in your application.

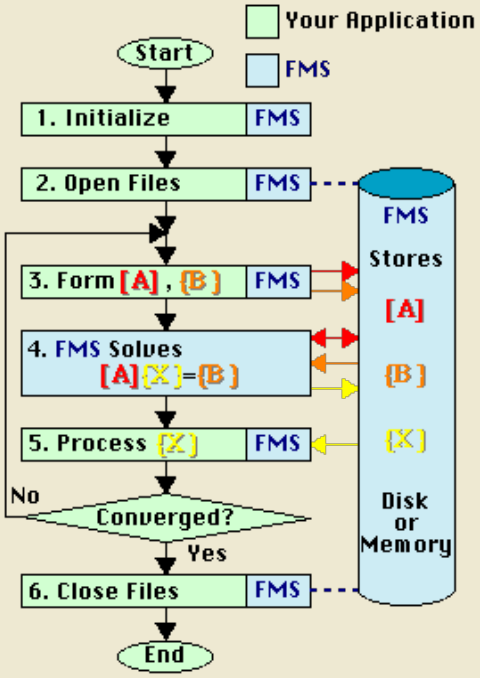
3.1 Calling FMS Subroutines
3.2 Using FMS Parameters

Copyright © Multipath Corporation

3.1. CALLING FMS SUBROUTINES

Overall Program Flow

FMS subroutines are divided into six groups, designed to match the 6 stages of your application program.



1. **Initialize FMS**

   Start ***FMS***.

2. **Open FMS Files**

   Create the ***FMS*** Database for storing `[A]`, `{X}` and `{B}`.

3. **Write data to FMS**

   Generate the data for the matrix `[A]` and right-hand side vector `{B}` and transfer the data to ***FMS***.

4. **Perform Matrix Algebra**

   Assemble and solve the system of equations `[A]{X}={B}`.

5. **Read data from FMS**

   Read the solution `{X}`, and compute local results. If the solution is not converged, repeat steps 3, 4, and 5.

6. **End FMS**

   Close the ***FMS*** Database and end ***FMS***.

All examples are documented with the ***FMS*** Subroutines divided into these six steps.

FMS Subroutine Names

All ***FMS*** subroutines begin with the module name or "FMS". The data type and matrix symmetry determines which subroutine to use as follows:

FMS Subroutine Names

Subroutine Name	Data Type	Symmetry
RSD...	Real	Symmetric
RND...	Real	Nonsymmetric
CHD...	Complex	Hermitian
CSD...	Complex	Symmetric
CND...	Complex	Nonsymmetric
FMS...	Both	All

To prevent any global symbol conflict, you should avoid naming subroutines and common blocks that begin with any of 3 letter

combinations listed above.

Calling Convention

FMS uses a standard FORTRAN calling convention. Actual parameters are passed by reference (address). Arrays are stored by rows first, then columns. For example A(2,2) is stored as {A11, A21, A12, A22}.
You may call all ***FMS*** routines from C by passing addresses and noting that FORTRAN and C store arrays in the opposite order.

Character strings require special mention. The actual mechanics used to pass the address and length of character strings in FORTRAN varies by machine and compiler. A compiler option is usually required to match the FORTRAN interface. The current most popular FORTRAN interface is to pass the address of the character string in the normal position and append the value of the length of the string as an additional argument. When multiple character strings are passed, the length of each is appended in order.

Data Size

FMS stores the data as follows:

- Real data is stored as 64-bit (8-byte) floating point numbers.
- Complex data is stored as a pair of 64-bit floating point numbers: first the real part then the imaginary part.
- On most machines, integers are stored as 64-bit (8-byte) integers. On older machines with 32-bit addressing operating systems integers are stored as 32-bit (4-byte) integers.

With this background, we are ready for a quick tour of the ***FMS*** subroutines, sorted according to the 6 stages of your analysis program.

1. Initialize FMS

1. Initialize FMS

NAME	Function
FMSINI FMSIN2	Initialize ***FMS*** - First Subroutine Called
FMSIST FMSRST FMSCST	Set an Integer Parameter Set a Real Parameter Set a Character Parameter
FMSIGT FMSRGT FMSCGT	Read an Integer Parameter Read a Real Parameter Read a Character Parameter
FMSSET FMSSETC	Set ***FMS*** Parameters interactively from FORTRAN Set ***FMS*** Parameters interactively from C

You initialize ***FMS*** with the following calls:

1. `CALL FMSINI`

REQUIRED:

The first ***FMS*** subroutine you call must be FMSINI. This subroutine initializes the ***FMS*** Parameters to default values and prints a description of the machine. You can only call FMSINI once during the execution of your program.
It is strongly recommended that the call to FMSINI be the first executable statement in your program. On some systems this is required to intercept children processes that begin execution at the beginning of your program. In all cases, the timing information provided by ***FMS*** will be more meaningful if it includes the non- ***FMS*** part of your program.

2. `CALL FMSIST(NAME,VALUE)`
`CALL FMSRST(NAME,VALUE)`
`CALL FMSCST(NAME,VALUE)`
OPTIONAL:
To change the values of integer, real and character ***FMS*** Parameters, call subroutines FMSIST, FMSRST, or FMSCST.

3. `CALL FMSIGT(NAME,VALUE)`
`CALL FMSRGT(NAME,VALUE)`
`CALL FMSCGT(NAME,VALUE)`
OPTIONAL:
You can obtain the value of integer, real and character ***FMS*** Parameters by calling FMSIGT, FMSRGT, or FMSCGT.

4. `CALL FMSSET`
`CALL FMSSETC`
OPTIONAL:
As an aid to software development, ***FMS*** includes subroutine FMSSET, which may be called to interactively view or alter ***FMS*** Parameters. You call FMSSET from FORTRAN or FMSSETC from C.

2. Open FMS Files

2. Open FMS Files

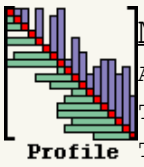
NAME	Function
FMSOV	Open a Vector File
FMSOV2	Open an Incore Vector File
FMSOS	Open a Submatrix File
RSDI RNDI CHDI CSDI CNDI	Open General Matrix
RSDANN RNDANN CHDANN CSDANN CNDANN	Open Full Incore Matrix

FMSOM	Open Second Matrix File
FMSBW	Compute Matrix Bandwidth
FMSPF	Compute Matrix Profile
FMSWF	Compute Matrix Profile from Wavefront Storage

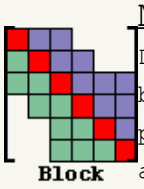
FMS provides the option of storing problem data on disk, which can significantly extend problem size and reduce machine cost. Disks are easily expanded while increasing memory to solve a larger problem is expensive and limited by the number of memory sockets and the density of memory chips. On small machines it may also be limited by the address space.
FMS organizes computation so once data is read from disk it is used many (often thousands) of times. This high "*reuse*" of data, which is possible with matrix algebra computations, allows ***FMS*** to match fast processing speed with slower disk transfer rates.
FMS uses "*asynchronous disk transfers*" to overlap I/O with processing so data is available when required and buffered to disk after computation.

FMS uses three types of files for storing problem data:

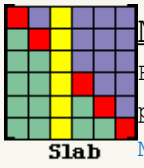
1. Vector Files
Each file record holds one vector, whose length is determined by the problem size, data type and disk properties. Each vector file can hold multiple vectors.
2. Submatrix Files
These files store matrix data as a collection of smaller matrices. Typically they are used by finite element programs. ***FMS*** includes the capability of assembling these submatrices into a global system matrix.
3. Matrix Files
These files store a square array of data, which is partitioned into disk records. Three types of file formats are provided:



MFMAT=1: Profile Solver
Accounts for the sparsity of matrix **[A]** on an equation by equation basis. Typical applications include finite element and finite difference programs. This is the default for sparse matrices.



MFMAT=2: Block Solver
Divides the matrix **[A]** into square blocks, accounting for sparsity on a block by block basis. This solver uses industry standard BLAS3 kernels and provides excellent I/O performance on extremely large problems. Typical applications include boundary integral and moment method programs. This is the default for full matrices.



MFMAT=3: Slab Solver
Extends the functionality of the Block Solver by providing full column partial pivoting for full nonsymmetric matrices. You must explicitly set MFMAT to select this option.

The second step in using ***FMS*** is opening files for storing vector, submatrix, and matrix data. Physically these files may be located on disk or in memory, depending on their size and the ***FMS*** Parameter INCORE. ***FMS*** stores the information it needs to access these files in a 25-word array, called a *"file attribute list"*. The contents of the file attribute lists for the 3 file types are

provided in the following links:

1. LUX(25), Vector File Attribute List
2. LUS(25), Submatrix File Attribute List
3. LUA(25), Matrix File Attribute List

2.1. Open Vector Files

A vector file "LUX(25)" is created in one of two ways:

1. `CALL FMSOV (NUMEQ,IDTYPE,NUMVEC,NAME,LUX)`

   For vectors that will be stored on disk or ***FMS*** managed memory, call FMSOV. You should use this option if there are several vectors or permanent storage is required.

2. `CALL FMSOV2(NUMEQ,IDTYPE,NUMVEC,X,LDX,LUX)`

   If the size or number of vectors is small, and they are already stored in your application, call subroutine FMSOV2. ***FMS*** will create a "vector file" out of your storage so ***FMS*** can operate directly on your data. This eliminates any need to write or read this vector data to/from ***FMS***.

2.2. Open Submatrix Files

A submatrix file "LUS(25)" is created by

`CALL FMSOS(LENR,LENI,LENV,NUMVEC,NUMSUB,NAME,LUS)`

Finite element programs that perform matrix assembly obtain the global matrix data from submatrix files. Subroutine FMSOS opens a submatrix file that includes storage for the submatrix and optional right-hand side vectors.

If you are using wavefront numbering, open a single submatrix file. The order that you write the data to the file determines the wavefront order.

If you generate submatrices by element type, you should try and place each element type on a separate submatrix file. When you open more than one submatrix file, you must store the file attributes in the array `LUS(25, NUMSF)`, where NUMSF is the number of submatrix files. FMSOS is called NUMSF times, with the attribute list `LUS(1,I)` supplied for the I-th submatrix file.

2.3. Open Matrix Files

A matrix file "LUA(25)" is created in one of three ways:

1. `CALL RSDI(LOWEQ,NUMEQ,NAME,LUA)`

   `CALL RNDI(LOWEQ,NUMEQ,NAME,LUA)`

   `CALL CHDI(LOWEQ,NUMEQ,NAME,LUA)`

   `CALL CSDI(LOWEQ,NUMEQ,NAME,LUA)`

   `CALL CNDI(LOWEQ,NUMEQ,NAME,LUA)`

   Most applications should use these subroutines to create matrix files. The profile vector LOWEQ(NUMEQ) describes the sparsity of your matrix. As a special case LOWEQ(1)=-1 describes a full matrix.

2. `CALL RSDANN(A,LDA,N,LUA)`

   `CALL RNDANN(A,LDA,N,LUA)`

   `CALL CHDANN(A,LDA,N,LUA)`

   `CALL CSDANN(A,LDA,N,LUA)`

`CALL CNDANN(A,LDA,N,LUA)`

For full incore matrices A(LDA,N), use these subroutines to perform the combined function of creating the matrix file and writing data to the file. This interface is only recommended when you know the matrix will remain small. For a more general matrix file, use the previous method.

3. `CALL FMSOM(LUA,NAME,LUF)`

Most applications overlay matrix factors on the original matrix to conserve space. However, if the original matrix is to be preserved, subroutine FMSOM may be called to create a second matrix file for storing the matrix factors.

2.4. Utilities for Opening Matrix Files

FMS includes the following utility subroutines which may be used to help create matrix files:

1. `CALL FMSBW(LOWEQ,NUMEQ,BWMAX,BWAVG,BWEFF)`

The FMSBW subroutine uses your profile vector LOWEQ(NUMEQ) to computes the maximum (BWMAX), average (BWAVG) and effective factoring (BWEFF) bandwidths of your matrix. You can use this information to determine matrix storage size and processing time.

2. `CALL FMSPF(LUS,NUMSF,NEWNUM,MAXEQ,LOWEQ,NUMEQ)`

Subroutines FMSPF computes the profile vector LOWEQ(NUMEQ) from the information you wrote on the submatrix files LUS.

3. `CALL FMSWF(LUS,NEWNUM,MAXEQ,LOWEQ,NUMEQ)`

Subroutine FMSWF converts the nicknames you wrote on the submatrix file LUS to equation numbers and computes the profile vector LOWEQ(NUMEQ).

3. Write Data to FMS Files

3. Write Data to FMS Files

NAME	Function
FMSWRT FMSWR8	Direct access write to submatrix and vector files
FMSSWR	Sequential write to submatrix and vector files
FMSEEK FMSEK8	Position file for sequential access
FMSROW FMSCOL	Write rows/columns to matrix file
FMSPUT	Write a block of data to a matrix or vector file
FMSCPY	Make a copy of a ***FMS*** file
RSUBLK RNUBLK CHUBLK CSUBLK CNUBLK	Define a block of matrix data

RNUSLB CNUSLB	Define a slab of matrix data

Data is written to ***FMS*** vector, submatrix and matrix files as follows:

3.1. Write Vector Files

Vector files are written as follows:

1. `CALL FMSWRT(LOGUNT,LOCUNT,ARRAY,NUMWRD)`

   If you used FMSOV to open the vector file, transfer the vectors to ***FMS*** by calling FMSWRT to transfer each vector `ARRAY(NUMWRD)` to file `LOGUNT` starting at `LOCUNT`. For example, the complex vectors `X(NUMEQ, NUMRHS)` are transferred to ***FMS*** file LUX as follows:

```
INTEGER LUX(25)
COMPLEX*16 X(NUMEQ,NUMRHS)
...
LDISK  =  1
DO NRHS  =  1, NUMRHS
   CALL FMSWRT (LUX(1), LDISK, X(1,NRHS), 2*NUMEQ)
   LDISK  =  LDISK  +  LUX(4)
END DO
```

   Note that the disk address begins at 1 and is incremented by the vector file record length, LUX(4). The number of words transferred, 2*NUMEQ, accounts for the real and imaginary parts of complex data.
   If the integers on your machine are only 32-bits, and your file is larger than 2 Gbytes, then you may pass the disk location to ***FMS*** as a 8-byte floating point number using subroutine FMSWR8.

2. If you used FMSOV2 to open the vector file, the vector write operation is not required. ***FMS*** operates directly on the data in your program.

3.2. Write Submatrix Files

1. `CALL FMSWRT (LOGUNT,LOCUNT,ARRAY,NUMWRD)`

   Fixed length record submatrix files are written using FMSWRT. For example, submatrices in standard FORTRAN format are transferred to the fixed length record submatrix file LUS as follows:

```
INTEGER LUS(25)
INTEGER IE(N+2), NELEQ, ITYPE IEQE(N)
EQUIVALENCE (NELEQ, IE(1))
EQUIVALENCE (ITYPE, IE(2))
EQUIVALENCE (IEQE , IE(3))
REAL*8 S(N,N)
...
NELEQ  = N
ITYPE  = 1
LDISKR = 1
LDISKI = 1
```

```
NUMREC = LUS(5)
DO NREC  =  1, NUMREC
   ...
   ... fill arrays IEQE(N) and S(N,N) for record NREC
   ...
   CALL FMSWRT (LUS(1), LDISKR, S, N*N)
   CALL FMSWRT (LUS(2), LDISKI, IE,  N)
   LDISKR = LDISKR + LUS(3)
   LDISKI = LDISKI + LUS(4)
END DO
```

Note that the program increments the file addresses by the submatrix record lengths LUS(3) and LUS(4). These values were computed when you called FMSOS.

If you are doing a nonlinear analysis and the model does not change, the submatrix integer file LUS(2) only needs to be written once.

If possible, you should store the submatrices on each file in the order of increasing lowest equation number. That is, write all submatrices that contribute to equation 1 first, followed by all remaining submatrices that contribute to equation 2, etc. This record order results in a minimum amount of data transfers during the assembly process. During development, the program can direct ***FMS*** to check the record order by including 1024 in the value of the IPRA parameter.

FMS buffers your write operations so the disk transfers can occur in parallel with your processing. The ***FMS*** Parameters LBUFSI, LBUFSR and LBUFSV control the size of the buffers used. If you are writing the records in a random order, you should turn off buffering by calling FMSIST, to set these buffer lengths to zero.

2. `CALL FMSEEK(LOGUNT,LOCUNT)`
`CALL FMSSWR(LOGUNT,ARRAY,NUMWRD)`

Variable length submatrix files are written sequentially using FMSSWR. The example above becomes:

```
INTEGER LUS(25)
INTEGER IE(N+2), NELEQ, ITYPE IEQE(N)
EQUIVALENCE (NELEQ, IE(1))
EQUIVALENCE (ITYPE, IE(2))
EQUIVALENCE (IEQE , IE(3))
REAL*8 S(N,N)
...
NELEQ  = N
ITYPE  = 1
CALL FMSEEK (LUS(1), 0)
CALL FMSEEK (LUS(2), 0)
NUMREC = LUS(5)
DO NREC  =  1, NUMREC
   ...
   ... fill arrays IEQE(N) and S(N,N) for record NREC
   ...
   CALL FMSSWR (LUS(1), S, N*N)
```

```
            CALL FMSSWR (LUS(2), IE,  N)
         END DO
```

3.3. Write Matrix Files

There are 5 ways to write data to ***FMS*** matrix files:

1. **CALL FMSROW(IEQN,DATA,LUA)**

   CALL FMSCOL(IEQN,DATA,LUA)

   These subroutines provide an easy interface when the matrix data is generated by rows or columns. You may also use this interface for Finite Element programs that perform their own assembly and store the data in profile (skyline) format. Each call transfers a row or column of DATA for equation IEQN to the ***FMS*** matrix file LUA. The data may be transferred in any row or column order. Either FMSROW or FMSCOL can be used. You may not mix calls to FMSROW and FMSCOL when filling a single matrix. After completing the calls to FMSROW or FMSCOL, you call RSDAF, RNDAF, CHDAF, CSDAF, CNDAF to factor the matrix.

2. **Use submatrix addition.**

   First you call FMSOS to open submatrix files. Then you use FMSWRT or FMSSWR to write the data to the submatrix real, integer and vector files. The submatrices may then be assembled as a separate step by calling RSDA, RNDA, CHDA, CSDA, CNDA, or during the factoring process with RSDAF, RNDAF, CHDAF, CSDAF, CNDAF. If you created submatrix vector files, they are assembled into global vector(s) using RSDAV and CSDAV

3. **CALL FMSPUT(LUFMS, NROWS, NCOLS, IROW1, JCOL1, A(NROWS,NCOLS), LDA)**

   This subroutine provides an interface when you generate the matrix data by blocks of size NROWS by NCOLS, starting at IROW1 and JCOL1 in the global matrix.

4. **Set MDATAU**

   This option allows you to define or modify a window of the matrix by a subroutine you provide. ***FMS*** calls your subroutine as the matrix is being built.
   For PROFILE and BLOCK matrices, provide one of the following:

```
SUBROUTINE RSUBLK (A,D,LOWEQ,LOCEQ,IROW1,IROW2,JCOL1,JCOL2,IJSTEP)
SUBROUTINE CHUBLK (A,D,LOWEQ,LOCEQ,IROW1,IROW2,JCOL1,JCOL2,IJSTEP)
SUBROUTINE CSUBLK (A,D,LOWEQ,LOCEQ,IROW1,IROW2,JCOL1,JCOL2,IJSTEP)
```

   These subroutines define a window of data in the lower triangle AL(IROW1:IROW2,JCOL1:JCOL2) and diagonal D(IROW1:IROW2).

   For nonsymmetric matrices, also provide one of the following:

```
SUBROUTINE RNUBLK (A,D,LOWEQ,LOCEQ,IROW1,IROW2,JCOL1,JCOL2,IJSTEP)
SUBROUTINE CNUBLK (A,D,LOWEQ,LOCEQ,IROW1,IROW2,JCOL1,JCOL2,IJSTEP)
```

   These subroutines define a window of data in the upper triangle AU(IROW1:IROW2,JCOL1:JCOL2).

   If you are using the SLAB option for full column partial pivoting of full nonsymmetric matrices, provide one of the following:

```
SUBROUTINE RNUSLB (A,LOCI,LOCJ,LUFLAG,JCOL1,JCOL2,NUMEQ)
SUBROUTINE CNUSLB (A,LOCI,LOCJ,LUFLAG,JCOL1,JCOL2,NUMEQ)
```

   to define the matrix data in the window (1:NUMEQ,JCOL1:JCOL2).

5. **CALL RSDANN, RNDANN, CHDANN, CSDANN, CNDANN**

If you used RSDANN, RNDANN, CHDANN, CSDANN or CNDANN to open a full incore matrix file, you do not need to write data to the matrix file. ***FMS*** operates directly on the data in your program.

4. Perform Matrix Algebra

4. Perform Matrix Algebra

NAME	Function
RSDAF RNDAF CHDAF CSDAF CNDAF	Assemble and factor a matrix
RSDA RNDA CHDA CSDA CNDA	Assemble matrix
RSDAV CSDAV	Assemble vectors
RSDF RNDF CHDF CSDF CNDF	Factor a matrix
RSUPIV CSUPIV	Handle zero pivots.
RSDS RNDS CHDS CSDS CNDS	Solve
RSDEX RNDEX CHDEX CSDEX CNDEX	Extract a submatrix
	Out-of-Core Multiply **[C] = [D] + [A][B],**

FMSMM	**[C] = [A][B], or** **[C] = [D] - [A][B].**
RSDMVM RNDMVM CHDMVM CSDMVM CNDMVM	Multiply **{Y} = [A]{X}** (FMSMM preferred)
RSDSVM CHDSVM CSDSVM	Multiply **{Y} = SUM([Si]{X})** (FMSMM preferred)
RNDVMM CNDVMM	Multiply **{Y} = {X}[F]**
RSDVVM CHDVVM CSDVVM	Multiply **[F] = {X}^{T}{Y}**, Symmetric **[F]**
RNDVVM CNDVVM	Multiply **[F] = {X}^{T}{Y}**, General **[F]**
RSDDVM CHDDVM CSDDVM	Multiply **[F] = {X}^{T}[D]{X}**
RNDDVM CNDDVM	Multiply **[F] = {X}^{T}[D]{Y}**
FMSVAN	Vector Norms, Add, Subtract, Move
RSDWTP	Accumulate **[S] = [S] + w[B]^{T}[D][B]**
RNDWTP	Accumulate **[S] = [S] + w[B]^{T}[D][A]**

Once matrix and vector data have been stored in the ***FMS*** Database, you can call the following matrix algebra subroutines to perform operations on the data.

1. Assemble, Fastor, Solve:

```
CALL RSDAF (LUAI,ALPHA,NUMAI,LUS,NUMSF,LUA,LUF,LUB,LUX,NUMRHS)
CALL RNDAF (LUAI,ALPHA,NUMAI,LUS,NUMSF,LUA,LUF,LUB,LUX,NUMRHS)
CALL CHDAF (LUAI,ALPHA,NUMAI,LUS,NUMSF,LUA,LUF,LUB,LUX,NUMRHS)
CALL CSDAF (LUAI,ALPHA,NUMAI,LUS,NUMSF,LUA,LUF,LUB,LUX,NUMRHS)
CALL CNDAF (LUAI,ALPHA,NUMAI,LUS,NUMSF,LUA,LUF,LUB,LUX,NUMRHS)
```

These subroutines are the main workhorse of ***FMS***. They perform the following tasks:

- Initialize matrix **[A]** to **[0]**, or a linear combination of NUMAI input matrices

[A] = SUM(ALPHA[AI])

- Add the submatrices specified on files LUS(25,NUMSF) to **[A]**.
- If MDATAU is set, your subroutines are called to define or modify the matrix data.
- Optionally save the assembled matrix **[A]** on file LUA.
- Factor the matrix and save it on file LUF.

  Subroutines RSDAF, CHDAF and CSDAF factor the symmetric matrix **[A]** into the form

  [L][D][L] T = [A] (symmetric)

  Subroutines RNDAF and CNDAF factor the nonsymmetric matrix **[A]** into the form

  [L][U] = [A] (nonsymmetric)

- Perform forward reduction (and diagonal scaling for symmetric matrices) on NUMRHS vectors located on file LUB. Write the reduced vectors to file LUX.

2. Assemble Matrix

```
CALL RSDA(LUS,NUMSF,LUA)
CALL RNDA(LUS,NUMSF,LUA)
CALL CHDA(LUS,NUMSF,LUA)
CALL CSDA(LUS,NUMSF,LUA)
CALL CNDA(LUS,NUMSF,LUA)
```

If you want to perform matrix assembly as a separate step, use these subroutines. Note that they only assemble matrix data from submatrices LUS(25,NUMSF) and user supplied subroutines, as directed by MDATAU=1. If you wrote the matrix data using FMSROW or FMSCOL, you must call subroutine RSDAF, RNDAF, CHDAF, CSDAF or CNDAF to assemble and factor the matrix.

3. Assemble Vectors

```
CALL RSDAV(LUBI,LUS,NUMSF,LUB)
CALL CSDAV(LUBI,LUS,NUMSF,LUB)
```

If you wrote submatrix vector files, use these subroutines to assemble the submatrix vectors on files LUS(25,NUMSF) and add the results to the vectors on file LUBI. The resulting vectors are written to file LUB.

4. Factor, Solve

```
CALL RSDF(LUA,LUF,LUB,LUX,NUMRHS)
CALL RNDF(LUA,LUF,LUB,LUX,NUMRHS)
CALL CHDF(LUA,LUF,LUB,LUX,NUMRHS)
CALL CSDF(LUA,LUF,LUB,LUX,NUMRHS)
CALL CNDF(LUA,LUF,LUB,LUX,NUMRHS)
```

If you want to perform factoring as a separate step, use these subroutines. The matrix on LUA is factored and written to LUF. Optionally NUMRHS vectors on file LUB are reduced and written to file LUX.

5. Compute Pivots

```
SUBROUTINE RSUPIV(*,IEQN,A,S,D)
SUBROUTINE CSUPIV(*,IEQN,A,S,D)
```

If you want to perform your own zero pivot handling, set the ***FMS*** Parameter MZERO to 2 and provide one of these subroutines. This is almost never required.

6. Solve

```
CALL RSDS(LUF,LUB,LUX,NUMRHS,ISKIP)
CALL RNDS(LUF,LUB,LUX,NUMRHS,ISKIP)
CALL CHDS(LUF,LUB,LUX,NUMRHS,ISKIP)
CALL CSDS(LUF,LUB,LUX,NUMRHS,ISKIP)
CALL CNDS(LUF,LUB,LUX,NUMRHS,ISKIP)
```

The factored matrix on LUF is used to solve the NUMRHS vectors on LUB. The results are written to LUX.
The RSDS, CHDS and CSDS subroutines solve the following symmetric system:

[L][D][L]T{X} = {B} (symmetric)

The RNDS and CNDS subroutines solve the following nonsymmetric system:

[L][U]{X} = {B} (nonsymmetric)

These subroutines contain the formal parameter ISKIP, which can skip forward reduction (ISKIP=1) or back substitution (ISKIP=2).

7. Extract Submatrix

```
CALL RSDEX(LUA,A,LDA,N)
CALL RNDEX(LUA,A,LDA,N)
CALL CHDEX(LUA,A,LDA,N)
CALL CSDEX(LUA,A,LDA,N)
CALL CNDEX(LUA,A,LDA,N)
```

If you are performing substructuring, use these subroutines to extract the substructure matrix A(LDA,N) from the lower-right corner of the ***FMS*** matrix LUA.

8. Matrix Multiply

```
CALL FMSMM(LUC,LUD,IACCUM,TRANSA,LUA,LUB)
```

This subroutine performs one of the following matrix or vector multiply operations:

Compute	LUC and LUD	IACCUM	TRANSA	LUA	LUB
{C} = {D} - [A]{B} {C} = [A]{B} {C} = {D} + [A]{B}	Vector File	-1 0 1	'N'	Matrix File	Vector File
{C} = {D} - [Si]{B} {C} = [Si]{B} {C} = {D} + [Si]{B}	Vector File	-1 0 1	'N'	Submatrix File	Vector File
{C} = {D} - {A}{B} {C} = {A}{B} {C} = {D} + {A}{B}	Vector File	-1 0 1	'N'	Vector File	Vector File
	Vector	-1		Vector	Vector

{C} = {D} - {A}T{B} {C} = {A}T{B} {C} = {D} + {A}T{B}	File	0 1	'T'	File	File
[C] = [D] - {A}T{B} [C] = {A}T{B} [C] = [D] + {A}T{B}	Matrix File	-1 0 1	'T'	Vector File	Vector File

You may use these subroutines to compute the residual vector after solving a system of equations or in evaluating the quadratic form $\{X\}^T[A]\{X\}$ in an Eigenvalue analysis.

9. Matrix-Vectors Multiply

```
CALL RSDMVM(LUA,LUX,LUY,NUMVEC,LUZ)
CALL RNDMVM(LUA,LUX,LUY,NUMVEC,LUZ)
CALL CHDMVM(LUA,LUX,LUY,NUMVEC,LUZ)
CALL CSDMVM(LUA,LUX,LUY,NUMVEC,LUZ)
CALL CNDMVM(LUA,LUX,LUY,NUMVEC,LUZ)
```

These subroutines perform one of the following matrix **[A]** vector **{X}** multiply operations:

Compute	IACCUM
{Y} = {Z} - [A]{X}	-1
{Y} = [A]{X}	0
{Y} = {Z} + [A]{X}	1

The matrix-vectors multiply calculation can be used to check the solution of the system of equations **[A]{X} = {B}**, or for iterative refinement **{B'} = {B} - [A]{X}**.

10. Submatrix-Vectors Multiply

When the matrix **[A]** is generated by submatrices, it may be more efficient to perform the matrix-vector multiplication using the submatrices, rather than the assembled matrix **[A]**.

```
CALL RSDSVM(LUS,NUMSF,LUX,LUY,NUMVEC,LUZ)
CALL CHDSVM(LUS,NUMSF,LUX,LUY,NUMVEC,LUZ)
CALL CSDSVM(LUS,NUMSF,LUX,LUY,NUMVEC,LUZ)
```

11. Vectors-Matrix Multiply

```
CALL RNDVMM(LUX,F,LUY,NUMX,NUMY,LUZ)
CALL CNDVMM(LUX,F,LUY,NUMX,NUMY,LUZ)
```

12. Vectors-Vectors Multiply

```
CALL RSDVVM(LUX,LUY,F,NUMXY)
CALL RNDVVM(LUA,LUY,F,NUMX,NUMY)
CALL CHDVVM(LUA,LUY,F,NUMXY)
CALL CSDVVM(LUA,LUY,F,NUMXY)
CALL CNDVVM(LUA,LUY,F,NUMX,NUMY)
```

13. Diagonal-Vectors-Matrix Multiply

```
CALL RSDDVM(LUX,LUD,F,NUMXY)
CALL RNDDVM(LUX,LUD,LUY,F,NUMX,NUMY)
CALL CHDDVM(LUX,LUD,F,NUMXY)
CALL CSDDVM(LUX,LUD,F,NUMXY)
CALL CNDDVM(LUX,LUD,LUY,F,NUMX,NUMY)
```

The RSDDVM, CHDDVM and CSDVVM subroutines compute the quadratic form

$$[F] = \{X\}^T[D]\{X\}$$

Only the lower triangle terms of the symmetric matrix **[F]** are computed and stored.

The RNDDVM and CNDDVM subroutines compute the full matrix as follows:

$$[F] = \{X\}^T[D]\{Y\}$$

The results are stored in the array F(NUMX,NUMY).

14. Vector Norms, Add, Subtract, Move

```
CALL FMSVAN(LUX,IALPHA,LUY,INORM,RNORM,IOPT)
```

15. Matrix Triple Product

```
CALL RSDWTP(W,B,D,S,N,M)
CALL RNDWTP(W,B,D,A,S,N,M)
```

FMS includes subroutines for performing the following weighted matrix products:

$[S] = [S] + W[B]^T[D][B]$ (rsdwtp)

$[S] = [S] + W[B]^T[D][A]$ (rndwtp).

These calculations are typically used in finite element programs for integrating the element matrices.

5. Read Data from FMS Files

5. Read Data From FMS Files

NAME	Function
FMSRED FMSRD8	Direct access read of vector file
FMSSWR	Sequential read of vector file
FMSGET	Read a block of data from a vector file

To read the data back into your program,

```
CALL FMSRED(LOGUNT,LOCUNT,ARRAY,NUMRED)
```

Most applications require only the solution vector **{X}**. Although submatrix and matrix data can be rewritten as part of a nonlinear analysis, the application program rarely reads it.

If you created the solution vector file by calling the FMSOV2 subroutine, the data is already in your program, and the read

operation is not required. If you created the vector file by calling the FMSOV subroutine, the FMSRED subroutine is called once for each vector.

6. End FMS

6. End FMS

NAME	Function
FMSCV	Close a vector file
FMSCS	Close a submatrix file
FMSCM	Close a matrix file
FMSEND	End FMS - Last Subroutine Called

You complete the use of ***FMS*** with the following calls:

1. `CALL FMSCV(LUX)`

   Whenever a vector file is no longer required, you may call FMSCV to close the file.

2. `CALL FMSCS(LUS)`

   Whenever a submatrix file is no longer required, you may call FMSCS to close the file.

3. `CALL FMSCM(LUA)`

   Whenever a matrix file is no longer required, you may call FMSCM to close the file.

4. `CALL FMSEND`

   REQUIRED:

   The last call to ***FMS*** must be FMSEND. This subroutine shuts down all the ***FMS*** utilities and prints any requested reports. For the timings to have the most meaning, this call should be the last statement of your program.

Print Subroutines

FMS Printing Subroutines

NAME	Function
FMSTIM	Obtain CPU and elapsed time
FMSPRV	Print vectors from vector file
FMSPRF	Print matrix from memory
FMSPRM	Print matrix from matrix file

FMS includes subroutines for obtaining the time used, printing vectors and printing matrices. You can use these subroutines any time after calling FMSINI.

To obtain the CPU, Wall and I/O wait times,

```
CALL FMSTIM(LUX,NUMV1,NUMVEC,NMSG)
```

To print vectors from a ***FMS*** vector file,

```
CALL FMSPRV(LUX,NUMV1,NUMVEC,NMSG)
```

To print an array A(NROWS,NCOLS) in your program,

```
CALL FMSPRF(A,NROWS,NCOLS,ISTYPE,IDTYPE)
```

To print a matrix from a ***FMS*** matrix file,

```
CALL FMSPRM(LUA,NMSG)
```

Memory Management Subroutines

FMS Memory Management Subroutines

NAME	Function
FMSIMG	Get Integer memory
FMSIMR	Return Integer memory
FMSRMG	Get Real*8 memory
FMSRMR	Return Real*8 memory
FMSCMG	Get Complex*16 memory
FMSCMR	Return Complex*16 memory

FMS contains a memory management system that you can use for scratch data storage. When FMSINI is called, a pool of memory is created. To allocate memory from this pool for storing INTEGER, REAL*8 or COMPLEX*16 data in your application, call one of the following subroutines:

```
CALL FMSIMG(IMD,LOC,LEN)
CALL FMSRMG(RMD,LOC,LEN)
CALL FMSCMG(CMD,LOC,LEN)
```

When you no longer require the storage, you can return it to the pool by calling

```
CALL FMSIMR(IMD,LOC,LEN)
CALL FMSRMR(RMD,LOC,LEN)
CALL FMSCMR(CMD,LOC,LEN)
```

If you want to list the contents of the ***FMS*** memory pool, you can include the statement

```
CALL FMSCST('SHOW','MEMORY')
```

All memory in the ***FMS*** memory pool is automatically shared among the processors. If you plan to use the ***FMS*** parallel processing utilities (see below), you should use these utilities to store the data you want shared in the ***FMS*** memory management pool.

Parallel Processing Subroutines

FMS Parallel Processing Subroutines

NAME	Function
FMSPAR	Queue parallel tasks
FMSRUN	Start parallel tasks running
FMSYNC	Wait for parallel tasks to complete
FMSONE	Acquire mutual exclusive lock
FMSALL	Release mutual exclusive lock

FMS includes utilities that let you run the non-***FMS*** part of your application in parallel. You may use these tools as follows:

1. You begin be placing code that can run in parallel in a subroutine, which we will call SNAME(P1, P2, ...). This code is written so that it is called MAXCPU times, with each call performing part of the work. Usually one of the NUMPAR parameters (P1, P2, ...) is different on each call so SNAME will know which part of the work to do. This subroutine may be tested in single processor mode as follows

```
DO ICPU=1,MAXCPU
   CALL SNAME(P1,P2,...)
END DO
```

2. The above code is then modified to use FMSPAR to run SNAME(P1,P2,...) in parallel. FMSPAR is called (MAXCPU-1) times to place the work for each child process in a queue.

```
DO ICPU = 1,(MAXCPU-1)
   CALL FMSPAR(NUMPAR,SNAME,P1,P2,...)
END DO
```

3. Subroutine FMSRUN is then called to start the children running.

```
CALL FMSRUN
```

4. Subroutine SNAME(P1,P2,...) is called by the parent to do its part of the work.

```
CALL SNAME(P1,P2,...)
```

5. After the parent completes its work, it calls subroutine FMSYNC to wait for all the children to complete.

```
CALL FMSYNC
```

For some applications, the children or parent may need to obtain exclusive use of the data for a short period of time. For example, if the calls to SNAME all passed the same parameters (P1, P2, ...) then the individual copies of SNAME will need to determine what work to do. They might do this by incrementing a shared variable NTASKS to determine their task, MYTASK. ***FMS*** includes a mutual exclusive lock that can be used as follows:

```
SUBROUTINE SNAME(P1, P2, ...,NTASKS)
INTEGER MYTASK
...
CALL FMSONE
   MYTASK = NTASKS
   NTASKS = NTASKS + 1
CALL FMSALL
```

The parent initializes NTASKS to 1. Each child then obtains a unique value of MYTASK that can be use to perform its part of the work. The ***FMS*** mutual exclusive lock FMSONE prevents two tasks from reading and incrementing NTASKS at the same time.

This completes the quick tour of ***FMS*** subroutines. You may find more information about any subroutine through Appendix C, FMS Subroutines, available in the Table of Contents.

Copyright © Multipath Corporation

3.2. USING FMS PARAMETERS

FMS Parameters are scalar values used to control the following:

- Hardware configuration including memory, processors, accelerators and disks,
- Use and configuration of accelerators,
- NonUniform Memory Access (NUMA) memory options,
- The ***FMS*** Database file system including file striping,
- Print output and format,
- Matrix file initialization and blocking,
- Matrix assembly,
- Matrix factoring and pivoting,
- Vector solution,
- Vector and matrix multiply,
- Error handling,

When you start ***FMS*** by calling FMSINI, ***FMS*** initializes all Parameters to default values. In most cases, the default values will allow you to get ***FMS*** running in your application. However, the ***FMS*** Parameters do provide some features that are useful during software development. Therefore you should become familiar with the ***FMS*** Parameters to know what options are available.

Setting FMS Parameters

After initialization by FMSINI, the Parameters may be changed in one of the following 3 ways:

1.
```
CALL FMSIST(NAME, IVALUE)
CALL FMSRST(NAME, RVALUE)
CALL FMSCST(NAME, CVALUE)
```
These subroutines provide a way to change INTEGER, REAL*8 and CHARACTER parameters respectively. You supply the NAME of the parameter as a character string and the new value as an INTEGER, 8-byte real or character string. ***FMS*** checks your value, and if appropriate changes the value of the ***FMS*** Parameter.
This method of setting ***FMS*** Parameters is recommended for modifications that become a permanent part of your application. For example, if know your matrix should always be positive definite, you should set the ***FMS*** Parameter MPOSDF as follows:

```
CALL FMSIST ('MPOSDF', 1)
```

2.
```
CALL FMSSET
```

Inserting this call in your application will cause ***FMS*** to solicit input interactively for viewing or changing ***FMS*** Parameters as follows:

```
FMSSET>name
```

where
`FMSSET>` is the prompt displayed by ***FMS*** and *name* is the name of the ***FMS*** Parameter whose value you want to display.

You may change the value of an ***FMS*** Parameter as follows:

```
FMSSET>name=value
```

where
value is the value you want to assign to the parameter.

When you have finished changing all the required parameters, you enter the characters *RETURN* at the FMSSET> prompt,

```
FMSSET>RETURN
```

and ***FMS*** continues on with your application.

This method of changing ***FMS*** Parameters is especially useful during development and tuning. It allows you to print out intermediate results or try out Parameters without them becoming a permanent part of your application.

3. License File
You may also change the initial (default) value of ***FMS*** Parameters in the License File by adding the following lines between the FMSSET and RETURN records:

```
FMSSET
name=value
...
name=value
RETURN
```

When ***FMS*** encounters the FMSSET record in the License File, it switches input to FMSSET for further processing. FMSSET then processes each modification *name=value* as though it was entered directly from a terminal. When FMSSET encounters the RETURN record, it transfers processing back to ***FMS***.

You should use this method to set ***FMS*** Parameters that are not part of the application, especially parameters that describe your hardware configuration. You may also use this method to try different parameters during development in a similar way to calling FMSSET.

Calling FMSSET, either from within your application or through the License File allows you to change ***FMS*** parameters without recompiling and relinking your application.

Reading FMS Parameters

You can obtain the value of ***FMS*** Parameters using one of the following methods:

1. From within your application:

```
CALL FMSIGT(NAME, IVALUE)
CALL FMSRGT(NAME, RVALUE)
CALL FMSCGT(NAME, CVALUE)
```

These subroutines provide a way to obtain the value of INTEGER, REAL*8 and CHARACTER parameters respectively. You supply the NAME of the parameter as a character string and ***FMS*** returns the INTEGER, 8-byte real or character string value.

2. At the FMSSET> prompt enter the name of the Parameter,

`FMSSET>`*name*

where *name* is the name of the ***FMS*** Parameter whose value you want to display. ***FMS*** then displays the Parameter name and its value.

3. Use the *SHOW* value to display a group of Parameters as follows:

- From within your application,

```
CALL FMSCST(SHOW, CVALUE)
```

- At the FMSSET prompt,

`FMSSET>`*SHOW=CVALUE*

where *CVALUE* is a character string describing the information to display. The following table lists the options and their description.

Displaying Multiple Parameters and Tables

CVALUE	**Description**
PARAMETERS	Lists all Parameters and their values
MODIFIED	Lists the Parameters that have been modified since the License File
MEMORY	Displays a table of memory allocation
MALLOC	Displays a table of MALLOCed memory
FILES	Displays a table of open ***FMS*** files
FMSDISK	Displays a table of disk usage
CALLS	Displays a table of subroutine calls including the number of times called and the amount of time (CPU and WALL) spent
PROBLEM	Shows the maximum problem size

SEMAPHORES	Lists the semaphores and their current value
SEMHISTORY	Lists the most recent history of semaphore values
IOQUEUE	Shows the status of the I/O queues
USAGE	Lists the usage statistics from the operating system (not available for all operating systems)
PLACEMENT	Shows memory placement on NUMA machines
PLACEMENTSUMMARY	Shows a summary of memory placement on NUMA machines
LIBRARIES	Lists the dynamically loaded libraries and their entry addresses
GPU	Lists the details of each GPU

You can also list the options available in this table at the FMSSET> prompt with
FMSSET>*SHOW*

With this background, let's take a quick tour of the ***FMS*** Parameters. If you want more information on any Parameter, just click on its name.

Hardware Parameters

NAME	**Function**
MAXCPU	Number of processors to use
MYNODE	Your process number (0 to MAXCPU-1) (read-only)
MDHW	Amount of memory installed (8-byte words)(read-only)
MAXMD	Amount of memory to use (8-byte words)
MAXMB	Amount of memory to use (Megabytes)
MEMPCT	Amount of memory to use (% of MDHW)
MDUSED	Amount of memory already used (8-byte words)(read-only)
MEMPTR	Pointer to ***FMS*** managed memory (read-only)
MDHOW	How memory is allocated
MDINIT	Memory initialization options
MALIGN	Memory alignment (8-byte words)
LPAGE	Page size (bytes)

NOPAGE	Page file location
ISYNC	Type of CPU synchronization
ISCOPE	Scope and priority of threads
MDPADD	Append prefetch area to memory allocations (8-byte words)
LCACHE	Size of cache (bytes)
LSTACK	Size of stack (bytes)
ITPCPU	Processor type (1=scalar, 2=vector)
LENVRG	Vector register length (8-byte words)

These parameters provide a way for ***FMS*** to know your hardware configuration. Some operating systems provide hooks to determine the number of installed processors and amount of memory. If available, ***FMS*** bases its default values on this information. In other cases this information is not available. You should review what ***FMS*** uses for the default values of MAXCPU and MAXMD and modify the value in the License File, if required.

The MYNODE Parameter allows you to obtain the thread number that is currently being used to execute your subroutine. The threads are numbered from 0 (parent) to (MAXCPU-1). Typically you would use this value in subroutines you are running in parallel using the ***FMS*** parallel processing routine FMSPAR.

The MDHW Parameter stores the amount of installed memory in 8-byte words. This is a read-only parameter.

You can specify the amount of memory ***FMS*** uses in one of 3 ways:

1. MAXMD specified the amount of memory in 8-byte words;
2. MAXMB specifies the amount of memory in Megabytes;
3. MEMPCT specifies the amount of memory as a percentage of the amount of installed memory MDHW.

The MDUSED Parameter allows you to obtain the amount of memory that has been used from the ***FMS*** memory pool. You may use this if you are calling FMSROW or FMSCOL to populate multiple matrix files simultaneously, or are using the ***FMS*** Memory Management System.

The MEMPTR returns the starting address of the pool of memory allocated by the ***FMS*** Memory Manager. It is used in FORTRAN programs to set the POINTER value in the arrays used to access ***FMS*** memory. In C applications, it provides a pointer to the start of the ***FMS*** memory.

The MDHOW Parameter determines which system services ***FMS*** uses to allocate memory. The options include mmap, malloc, shm and a fixed array. Not all options are available on all machines. You should use the default value selected by ***FMS***.

The MDINIT Parameter provides options for initializing the memory in the ***FMS*** memory pool. Normally this is not

required. However, if you are performing benchmarking and want to set the system page table to a known state, you may do so with this parameter. When NUMA options are used, the default value of this Parameter is automatically changed to explicitly place the memory.

The MALIGN Parameter determines the alignment of ***FMS*** memory. Typically this is set to the page size LPAGE.

On most systems, ***FMS*** uses the system page file for backing up the memory allocated by ***FMS***. The NOPAGE Parameter provides the option of creating a special page file for ***FMS***. However you should use this option with caution. Some systems will attempt to continuously keep this private page file consistent with the ***FMS*** memory pool, resulting in degraded performance.

The ISYNC Parameter is used to control the type of parallel synchronization. The default value of semaphores should almost always be used instead of the optional spin-lock method.

The ISCOPE Parameter controls the priority and scheduling of the threads. ***FMS*** automatically sets this Parameter for best performance.

The remaining hardware Parameters MDPADD, LCACHE, LSTACK, ITPCPU and LENVRG control hardware architecture features and should be left at the default values.

NUMA Parameters

The following parameters only apply to machines having nonuniform memory access (NUMA). Currently this includes the SGI Origin series. These parameters may only be specified in the license file. These options are most useful when the machine is dedicated to the application using ***FMS***.

NAME	Function
NUMAFL	Master flag. Must be set to 1 for any of the following parameters to take effect.
NUMAPR	Print output level
MYCPU1	Starting processor to use. 1 is first processor.
NPNODE	Number of processors per node
NUMNOD	Number of nodes (read-only)
NUMAFX	Thread binding to processor options.
MDWHEN	When memory is allocated
MAXLMD	Stride between nodes for placing memory (bytes)
NUMATP	Topology (SGI only)
MMGLUE	Combine blocks during matrix multiply

NUMAFL is the main flag that turns all the ***FMS*** NUMA features on or off. By default, NUMA features are turned off, providing an environment best suited for small jobs in a multi-user environment. When NUMAFL is set, threads and memory become bound to processors, providing explicit control of the machine environment. This feature is best used for large jobs on a dedicated machine, or part of a machine.
The NUMAPR Parameter controls the amount of printed output that occurs when the NUMA environment is being established. This includes information about thread and memory placement.

When NUMA effects are on, MYCPU1 specifies the starting CPU number for the job. This Parameter must be set explicitly if multiple jobs are being run concurrently.

Within ***FMS***, a "NODE" is defined as a part of the NUMA machine having uniform memory access. For current NUMA machines, this is between two and four processors. NPNODE specifies the number of processors per node.

The NUMAFX Parameter determines how the threads are bound to the processors.

The MDWHEN Parameter determines when memory for the ***FMS*** memory pool is allocated from the system. For proper NUMA placement, memory is physically allocated from the system and properly placed as required by ***FMS*** or your application.

MAXLMD specifies the stride by which memory is allocated among the nodes in a round-robin fashon. If you are allocating memory from the ***FMS*** memory pool by the parent in your application, the memory will be allocated among the nodes using this stride. A zero value will override this feature and allocate all the memory on the node running the parent thread. When a memory request is made by a child thread, all memory is automatically allocated on the node running the child thread. This feature provides the control required for you to maximize local memory references in the non-***FMS*** part of your application.

The remaining NUMA Parameters, NUMATP and MMGLUE, should be left at the default values selected by ***FMS***.

File Parameters

NAME	Function
INCORE	File location (disk or memory)
IEXIST	Use existing file
IFKEEP	Keep files
IREST	Write restart information
IOKIDS	Open for access by children
NUMIOQ	Number if I/O queues
NUMSTP	Number of stripes per queue

NUMVOL	Number of volumes per stripe
MBYTES	Number of megabytes per volume
IOTYPE	Perform raw or direct I/O
IOSYNC	Perform synchronous I/O
LSECTR	Disk sector length (8-byte words)
NOOPEN	Do not physically open files
LBUFSI	Buffer size for submatrix integer records
LBUFSR	Buffer size for submatrix real records
LBUFSV	Buffer size for submatrix vector records

There are several parameters which provide options for configuring the ***FMS*** Database.

You may direct ***FMS*** to place files in memory with the INCORE Parameter. This can provide improved performance for small problems.

By default, ***FMS*** treats files in the Database as scratch files that are deleted at the end of the job. You may set the IFKEEP Parameter before the file is closed to retain a file. Later, when you want to reopen the file, you can set the IEXIST Parameter to use an existing file, rather than to create a new one.

You may also direct ***FMS*** to write a restart file during matrix factoring, or to restart a job in the middle of matrix factoring using a previously written restart file. The ***FMS*** Parameter IREST controls this feature.

Normally ***FMS*** opens a file for access by the parent. If you plan to read or write an ***FMS*** file from threads you created by calling FMSPAR, you should set the IOKIDS Parameter.

The ***FMS*** File Striping System is controlled by the NUMIOQ, NUMSTP, NUMVOL and MBYTES Parameters. These Parameters specify the number of queues, stripes per queue, volumes per stripe and capacity of each volume. The ***FMS*** File Striping System can significantly improve I/O performance. You should review this option before completing your installation. Because the disk configuration on most systems may change, you should set these parameters in the License File.

On some Unix systems, ***FMS*** can bypass the buffer cache and perform I/O directly to the disks using raw or direct I/O. The ***FMS*** Parameter IOTYPE discusses these options.

By default, ***FMS*** performs transfers to the Database asynchronously. You may turn this feature off and perform transfers synchronously with the IOSYNC Parameter.

The LSECTR specifies the minimum number of 8-byte words which may be transfered to a physical disk dirve. Normally the default value selected by ***FMS*** will be correct. However if your file system contains multiple drives

striped into a single mount point, the default value may be too low. ***FMS*** automatically detects this condition and provides instructions on how to change this Parameter to the correct value.

The NOOPEN Parameter skips the physical opening of files. This feature is used to estimate the disk space that will be required when the files are actually opened.

If you are using submatrices to define the matrix data, ***FMS*** will buffer the data as it is written by your application. The LBUFSI, LBUFSR and LBUFSV parameters control the size of the buffers used for the integer, real and vector files.

Data Formatting Input/Output Parameters

NAME	Function
	Print Code Summary
PRINT	Set all print codes
IPRI	Initialization print code
IPRA	Assembly print code
IPRF	Factoring print code
IPRS	Solution print code
IPRMV	Matrix-vectors multiply print code
IPRVV	Vectors-vectors multiply print code
IPRDV	Vectors-diagonal-vectors multiply print code
LOGTIM	Accumulate statistics on ***FMS*** subroutine usage
NEQPR1	Lower limit on equation to print
NEQPR2	Upper limit on equation to print
NROWPG	Number of rows per page
NCOLPG	Number of columns per page
IFMAT	Format for printing integers
RFMAT	Format for printing real numbers
CFMAT	Format for printing complex numbers
SHOW	Display reports

RESET	Reset ***FMS*** parameters to default values
LUINPT	Input unit number
LUPR	Output unit number

FMS provides complete control over printed output with the print Parameters. You may select the level of printing at any stage of your application with the IPRI, IPRA, IPRF, IPRS, IPRMV, IPRVV and IPRDV Parameters. If you want to set all these parameters to the same value, you may use PRINT. The Print Code Summary is a useful list of the options that are available with each parameter.
The default print level includes status and timings for each subroutine, which is appropriate for most production applications. However, during installation you may wish to modify these parameters to check intermediate results. The FMSSET utility is especially useful for setting these parameters during installation.

Normally ***FMS*** accumulates timing information for subroutines you call. The LOGTIM Parameter provides additional options for timing internal routines. This information may be useful for final tuning. Because the additional low level timings will increase run time, this Parameter should be left at the default value for production.

The amount of data in a matrix or vector may be very large. ***FMS*** provides the NEQPR1 and NEQPR2 parameters to control the window of data that is printed.

The NROWPG and NCOLPG control the size of your display. The default values of 24 by 80 is appropriate for most terminals.

The IFMAT, RFMAT and CFMAT control the format for INTEGER, REAL*8 and COMPLEX*16 data. You may adjust these Parameters to control the amount of information printed by each matrix and vector term.

The SHOW Parameter is useful for printing reports on ***FMS*** Parameters, subroutine calls, memory usage and the status of the ***FMS*** Database.

You may use the RESET Parameter to reset all parameter values to the default values, including any changes you made in the License File.

FMS prints output to file LUPR. FMSSET reads input from file LUINPT.

Matrix File Initialization Parameters

NAME	Function
MFMAT	Matrix format (PROFILE, BLOCK, SLAB)
LENSDF	I/O record size (PROFILE format)
NEQBIO	I/O block size (BLOCK & SLAB format)
NEQBLK	Compute block size

NEQBLO	Minimum compute block size
NEQBHI	Maximum compute block size
NEQBV	Vector block size
IJSTEP	Stride toward diagonal
INSIDE	Blocking pattern (PROFILE format)

When you open a matrix file, ***FMS*** stores the data in one of three formats: PROFILE, BLOCK or SLAB. You may use the MFMAT Parameter to override the default selection. SLAB format, which provides for full column partial pivoting, must always be selected by setting MFMAT. Because the best format depends only on your matrix, you should call FMSIST in your application to change this value.
The LENSDF and NEQBIO Parameters control the record size that is used for the matrix file. The default values of these parameters should provide the best performance.

The storage within each record is controlled by the NEQBLK, IJSTEP and INSIDE Parameters. The default values of these parameters are carefully selected to match optimized kernels within ***FMS***. ***FMS*** selects the value of NEQBLK to be between NEQBLO and NEQBHI. You may experiment with these during final tuning, but the default values should provide the best performance.

Matrix Assembly Parameters

NAME	Function
LOWASM	First changed equation for nonlinear reassembly
MDATAU	Call your subroutine to define or modify matrix data
LUOK	Make a nonsymmetric matrix symmetric

The ***FMS*** Parameters also provide options for building the matrix. LOWASM may be used during nonlinear analysis to reuse the part of the matrix factors which have not changed. LUOK will convert a nonsymmetric matrix to symmetric. The MDATAU Parameter directs ***FMS*** to call subroutines you provide to define or modify the matrix data.

Matrix Factoring Parameters

NAME	Function
MPOSDF	Positive definite flag
MZERO	Zero pivot flag

NBITSZ	Number of lost bits on diagonal to call zero
NBITSP	Number of lost bits on diagonal to print a warning
RPIVOT	Value to set zero pivot reciprocal
PIVMIN	Absolute value of minimum pivot
NPIVOT	Number of equations pivoted (SLAB format) Read-only
NUMSCG	Number of diagonal sign changes during factoring
NEQSUB	First equation of substructure
IALGOR	Optional factoring algorithms
MINDIM	Minimum dimension to terminate Strassen's algorithm
SAVOPS	Number of floating point operations saved (read-only)
CONINV	Inverse condition number (read-only)
ATERMS	Number of terms in matrix (read-only)
AZEROS	Number of zero's in matrix (read-only)
FZEROS	Number of zero's in matrix factor (read-only)
IPOLLD	CPU synchronization on diagonal elements

There are several options provided for handling pivots during the factoring process. MPOSDF instructs ***FMS*** to halt if a negative pivot is found. NBITSP controls warning messages for pivots "close" to zero. NBITSZ sets the tolerance for zero. PIVMIN also sets a limit on the minimum pivot value.

When ***FMS*** detects a bad pivot, it halts by default. You may use the MZERO Parameter to override this action. The options include setting the pivot reciprocal to RPIVOT and continuing or calling a subroutine you provide.

The NEQSUB Parameter offers a useful modification to the factoring and solving process for many applications.

For some applications, the algorithm used for matrix multiply and/or complex arithmetic may be altered to improve performance without loss of accuracy. The IALGOR, MINDIM and SAVOPS Parameters provide these options.

After the factoring process, you may call FMSIGT to obtain information about the matrix factors. NUMSCG provides the number of sign changes on the diagonal and NPIVOT provides the number of equations pivoted for SLAB matrices.

You may also call FMSRGT to find out information about how the matrix filled in during the factoring process.

ATERMS lists the total number of terms in the matrix, AZEROS lists the number zero terms in the unfactored matrix and FZEROS lists the number of zero terms in the factored matrix.

During factoring the parallel processes must synchronize as they compute terms around the diagonal. The IPOLLD parameter selects between event flags and spin-locks for this synchronization. It is strongly recommended that you leave this parameter at the default value.

Solving Parameters

NAME	Function
IPBACK	Use dot products during back substitution (PROFILE format)
LUTRAN	Solve `[A]T{X}={B}` instead of `[A]{X}={B}`

As an option, you may use the IPBACK Parameter to direct ***FMS*** to perform back substitution with inner products. For most cases, the default value is preferred.

The LUTRAN Parameter may be used to solve using the transpose of the matrix `[A]` for nonsymmetric problems. This feature has applications in evaluating derivatives of potential functions.

Multiply Parameters

NAME	Function
MMROW	Shifts the product of `[A][B]` by MMROW rows in `[C]`.
MMCOL	Shifts the product of `[A][B]` by MMCOL columns in `[C]`.
MMKA	Adds MMKA columns of 0's to the beginning of matrix `[A]`.

When using FMSMM to compute matrix or vector multiply operations, the resulting product is normally stored in the upper-left corner of the output matrix. The ***FMS*** Parameters MMROW, MMCOL and MMKA provide a way of shifting where the product is stored.

Error Handling Parameters

NAME	Function
NERROR	***FMS*** error number Read-only
IOSTAT	I/O status returned from last system call

	Read-only
LUERR	Unit number for writing error messages
IPRERR	Information to print if an error occurs
IERROR	Action to take if a fatal error is detected

FMS contains an error monitoring system that provides useful information whenever an error occurs. This information is printed on unit LUERR. The amount of information is controlled by IPRERR. The ***FMS*** error number is provided by NERROR. If the error was due to a system service, the system error is available in IOSTAT.

The ***FMS*** error reporting system is designed to provide the information you need without modification. If, however, you want to process your own error conditions, you may set the IERROR Parameter and check the status in NERROR after each ***FMS*** call.

This completes the quick tour of ***FMS*** Parameters. A detail description of each Parameter is listed in Appendix D.

Copyright © Multipath Corporation

4. FMS ADVANCED FEATURES

4.1 Pivoting

FMS monitors the diagonal pivots during the factoring process. You may set tolerances based on machine architecture for printing warning messages when a pivot becomes small, or when a pivot is considered zero. You may also provide a subroutine you write to handle exceptions when a zero pivot is encountered.

4.2 Computing Eigenvalues

FMS includes utilities for computing Eigenvalues using subspace iteration.

4.3 Substructuring

This option allows you to interrupt the factoring and solving process at the bottom of the matrix and extract a reduced system of equations. You may use this for design optimization, nonlinear analyais, superelements and solving nonsymmetric systems when only a few of the matrix terms are nonsymmetric. You select this option with the NEQSUB Parameter.

4.4 Nonlinear Reassembly

This option allows you to reuse part of the matrix factors when only some of the matrix terms have changed. You select this option with the LOWASM Parameter.

4.5 Force Symmetry

This option transforms a matrix stored as nonsymmetric into a symmetric matrix by averaging the off-diagonal terms. You select this option with the LUOK Parameter.

4.6 Strassen and Winograd Algorithms

These algorithms reduce the number of floating point operations for BLOCK and SLAB matrices. They are selected by the IALGOR Parameter

Copyright © Multipath Corporation

4.1. FMS PIVOTING

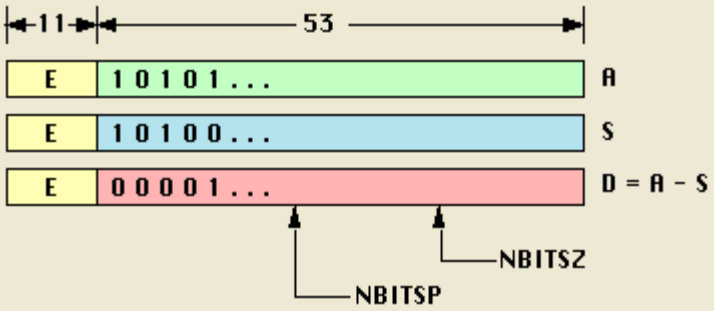
During the factoring process, the diagonal terms, which are called pivots, are monitored to detect possible error conditions. Positive definite matrices should not generate any negative pivots. If the ***FMS*** Parameter MPOSDF is set to 1 and a negative pivot is found, a fatal error condition is generated.

Not all matrices **[A]** can be factored into the form **[L][D][L]T** (symmetric) or **[L][U]** (nonsymmetric). If two rows are identical except for a scale factor, the equations are linearly dependent and a zero pivot results. In actual practice, equations that are close to being linearly dependent generate pivots that are close to zero. The result is errors in the solution vector when the matrix factors are used.

To detect these ill-conditioned matrices, ***FMS*** monitors the accuracy of the diagonal pivots. Each term in the diagonal factor is computed from an equation of the following form:

```
D = A - S
```

where D is the matrix factor diagonal being computed, A is the diagonal of the original matrix, and S is the sum of an inner product of the row to the left of D and the column above D.
The following figure shows how precision is lost during subtraction from a hardware point of view.



In this example, a 64-bit floating point representation is used with 11 exponent bits and 53 mantissa bits (IEEE format).

The first step in the subtraction process is to right shift the mantissa of the smaller term so that both the exponents of A and S are identical. Next, an integer subtraction is performed to compute the result D = A - S.

If A and S are close, their exponents and leading mantissa bits will be the same, as shown in the figure above. The result D will then contain leading 0's, representing lost bits of accuracy.

FMS contains two precision flags which are based on the number of lost mantissa bits. If the calculation of D results in NBITSP lost mantissa bits, ***FMS*** prints a warning message and continues with the factoring process. The default value of 26 corresponds to a ratio of

```
(A-S)/A < 1.5E-8.
```

If the calculation of D results in NBITSZ lost mantissa bits, ***FMS*** handles the pivot as though it were zero. The default value of 39 corresponds to a ratio of

```
(A-S)/A < 1.8E12.
```

FMS will also generate a zero pivot condition if the absolute value of D is less than or equal to the ***FMS*** Parameter PIVMIN (for complex data, D squared is compared to PIVMIN). The default value of 0 for PIVMIN prevents this test from being used unless you call FMSRST to reset PIVMIN.

The action taken by ***FMS*** when a zero (or close to zero) pivot is found depends on the parameter MZERO. If MZERO is 0 (default), a fatal error is generated. If MZERO is 1, ***FMS*** sets the pivot reciprocal to RPIVOT, prints a warning message and continues. If MZERO is set to 2, ***FMS*** calls your subroutine RSUPIV or CSUPIV according to the appropriate data type.

When a zero pivot is encountered, it does not mean the matrix cannot be factored. It just means that the equation could not be eliminated at that particular point. The objective of pivoting is to relocate zero pivot equations to a lower point in the matrix where they may be processed.

For nonsymmetric full matrices, ***FMS*** includes an option for full column partial pivoting. This option selects the largest absolute value pivot as each diagonal is processed by examining all the remaining equations in the column containing the diagonal. If a term larger than the diagonal is found, the rows are swapped to move the larger term to the diagonal position. This option may be selected by setting the matrix format type MFMAT to a SLAB matrix.

You can always process any matrix as full and nonsymmetric. However if the matrix is symmetric or contains a lot of zeros, this would be inefficient.

It is possible to accomplish pivoting for sparse or symmetric matrices by making two passes on the matrix. On the first pass, the ***FMS*** Parameter MZERO is set to 1. If a zero pivot is found, ***FMS*** sets its reciprocal to RPIVOT (default 0) and continues. This has the effect of eliminating the row and column corresponding to that equation from the matrix. At the end of factoring, ***FMS*** has printed a list of all zero pivot equations, if any.

For the second step, you renumber the equations placing the zero pivot equations at the bottom of the matrix. This has the effect of full row and column pivoting, which leaves diagonal matrix terms on the diagonal.

Once the matrix has been rebuilt in the new order, it may be refactored as in step 1. In the unlikely event that more zero pivots were found, these steps may be repeated.

You may automate the above procedure to provide zero pivot handling by setting

MZERO = 2.

This will direct ***FMS*** to call a subroutine you provide, either RSUPIV or CSUPIV, according to the appropriate data type. You may flag the zero pivot equations in an array or provide other special effects particular to your own program.

Copyright © Multipath Corporation

4.2. FMS COMPUTING EIGENVALUES

FMS contains subroutines which are useful for determining some of the lowest eigenvalues and eigenvectors of the following systems:

`[A]{X} = [M]{X}[E]`

where:

`[A]` = a known N-by-N matrix stored by ***FMS***,

`{X}` = a N-by-M group of eigenvectors to be determined,

`[M]` = a known N-by-N matrix with the same sparsity as **`[A]`**, or is a diagonal matrix,

`[E]` = a diagonal M-by-M matrix of eigenvalues to be determined.

Numerous techniques are currently available for computing **`{X}`** and **`[E]`**. No single technique is best for all possible matrices **`[A]`** and **`[M]`**. For this reason, ***FMS*** offers a building block approach for computing eigenvalues.

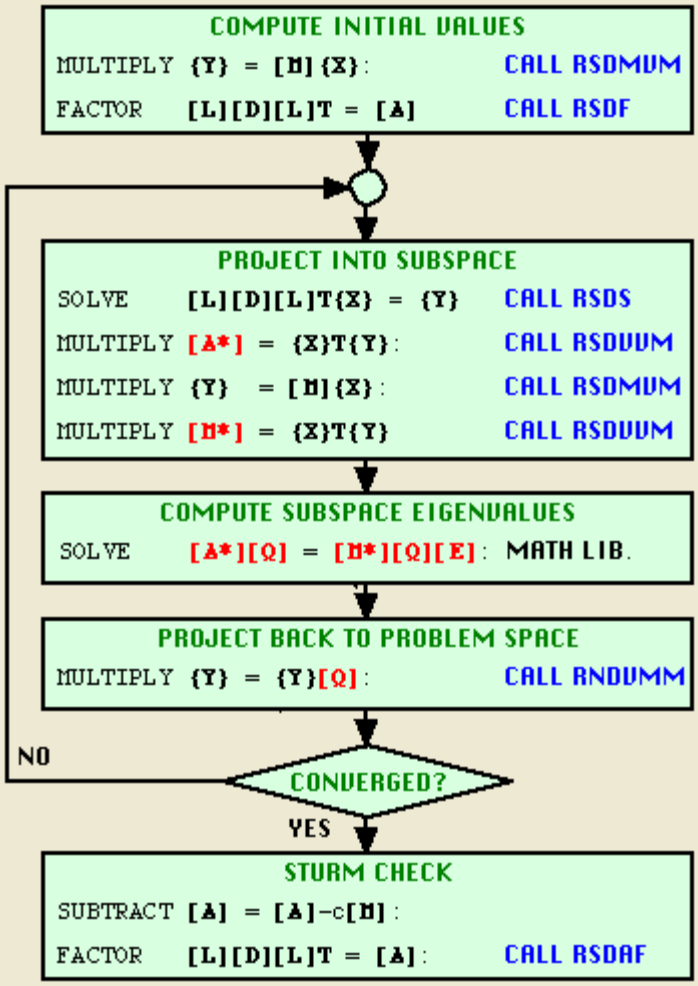
Two of the most important tools used for computing eigenvalues have already been presented: matrix factoring and vector solution. You can factor the matrices **`[A]`** and **`[M]`** by themselves or compute the factors for the shifted matrix **`[A]`** - c**`[M]`**. The processing speed of ***FMS*** permits you to perform matrix factoring on a more frequent basis than can currently be implemented in most programs. You can use the ***FMS*** parameter NUMSCG to count the number of diagonal sign changes during the factoring process to determine the number of eigenvalues below the current shift point.

One technique that shows how to use ***FMS*** for computing eigenvalues of large matrices is subspace iteration. In this technique, ***FMS*** projects the original problem into a subspace having considerably smaller dimensions.

You can easily compute the eigenvectors in the subspace and project them back to the problem space. You repeat this process until the eigenvalues in the problem space converge. This technique is especially useful for large systems that are resident on disk.

The following figure shows a flowchart of the subspace iteration algorithm and the ***FMS*** subroutines that you can use at each stage of the analysis.

You begin the process by selecting initial values for the **`{X}`** vectors, which define the subspace. The first set of **`{Y}`** vectors are computed from **`{Y}=[M]{X}`**, which the ***FMS*** matrix-vectors multiply subroutine performs. The matrix **`[A]`**, which can optionally be shifted, is then factored using the appropriate ***FMS*** factoring subroutine.



The first step in the interactive loop is to project the problem into the subspace by forming matrices **[A*]** and **[M*]**. This process involves solving for new **{X}** vectors using the **{Y}** vectors as right-hand sides. ***FMS*** vectors-vectors multiply subroutine computes **[A*]={X}T{Y}** to obtain the matrix **[A*]**. The ***FMS*** matrix- vectors multiply subroutine then computes **{Y}=[M]{X}** to obtain a new set of **{Y}** values. The ***FMS*** vectors-vectors multiply subroutine computes **[M*]={X}T{Y}** to obtain the subspace matrix **[M*]**.

An important feature of the above process is that it preserves the sparsity and symmetry of the large matrices **[A]** and **[M]**. For ***FMS***, another important feature is that it can process a large number of vectors in parallel.

The next step is to solve the following problem for the eigenvectors **[Q]** in the subspace:

[A][Q] = [M*][Q][E]

You can perform this calculation using standard math library subroutines designed to solve eigenvalue problems for full memory resident matrices.

After you compute the eigenvectors in the subspace, project them back to the problem space by computing **{Y}={Y}[Q]**. You can use the ***FMS*** vectors-matrix multiply subroutine to perform this task.

Repeat the above process until you obtain an acceptable set of eigenvectors in the problem space. As a final check, shift and factor the matrix to determine if all required eigenvalues have been computed.

FMS intentionally leaves the control of this calculation to your application program. You must specify the dimension of the subspace, the initial starting values, and the convergence criteria. Making these decisions properly usually involves insight into the particular problem you are solving.

Copyright © Multipath Corporation

4.3. FMS SUBSTRUCTURING

FMS includes the capability of partially factoring the lower corner of the matrix **[A]** and vectors **{B}**, which may then be used to solve a reduced system of equations. The ***FMS*** parameter NEQSUB is used to specify the first partially factored diagonal in matrix **[A]** and the first reduced term in vectors **{X}**.

The matrix and vectors are partitioned into the form

```
+-          -+ +-  -+   +-  -+
| A11 | A12 | | X1 |   | B1 |
+-----+-----+ +----+ = +----+<--NEQSUB
| A21 | A22 | | X2 |   | B2 |
+--   |     -+ +-  -+   +-  -+
```

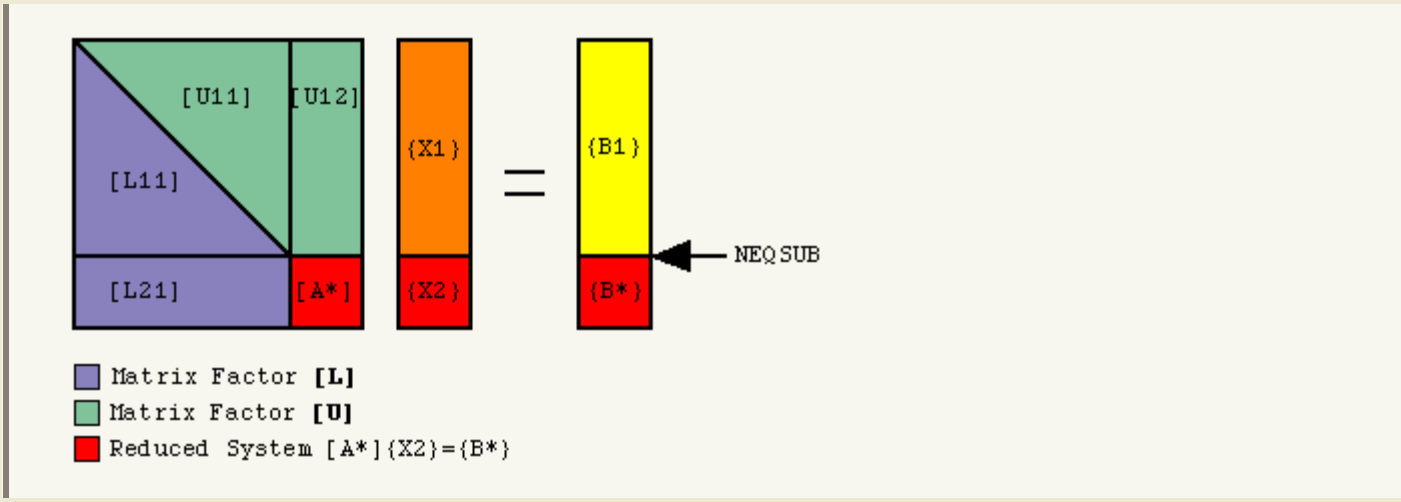
For nonsymmetric problems, the matrix terms **[A11]**, **[A21]** and **[A12]** are factored normally into **[L11]**, **[U11]**, **[L21]** and **[U12]**. However the matrix **[A22]** is only partially factored to produce the modified matrix **[A*]**,

[A*] = [A22] - [L21][U12] .

Similarly during forward reduction the vectors **{B1}** are reduced normally but the vector **{B2}** is only partially reduced by

{B*} = {B2} - [L21]{B1} .

The result is a reduced system of equations



which may be solved for the solution vector **{X2}**. Once the solution vector **{X2}** is obtained, the values may be placed in the original vector **{X}** and ***FMS*** will complete the solution process for **{X1}**.

One important property of the reduced matrix **[A*]** and vector **{B*}** is worth noting. They are computed from the

original matrix **[A22]** and vector **{B2}** by subtracting terms only. No multiplications on **[A*]** and **{B*}** have been performed. Therefore if you want to produce changes to the original terms in **[A22]** or **{B2}**, and those changes are in the form of additions or subtractions, they may be applied directly to the reduced system **[A*]** and **{B*}**.

The above strategy has some important applications which can save a considerable amount of computer time.

Design Optimization and Nonlinear Analysis

One application is in design optimization or nonlinear analysis where the system must be solved several times. Between each solution, some of the matrix coefficients in the lower right corner of the matrix are changed. Instead of completely resolving the entire system at each step, NEQSUB is first used to produce a reduced system containing the part of the matrix that changes. This smaller system is then used to find the optimal design or perform the nonlinear analysis. When the final solution is obtained, the values of **{X2}** are substituted back into the original system to compute **{X1}**. Note that the benefits of doing this depend on locating the changing matrix values in the bottom of the matrix.

Superelements

A second application is in building models of objects that have repeated sections. For example, suppose you want to analyze a ship that is constructed by repeating the same design down its length. You could first build a model of the generic section, numbering the equations so that the unknowns on the boundary were at the bottom of the matrix. The parameter NEQSUB could then be used to compute a reduced matrix **[A*]** which exactly represents the generic section in terms of the boundary unknowns only. The element assembly feature of ***FMS*** could then be used to assemble these generic sections into an overall model.

FMS contains some tools that can simplify this process. Subroutine RSDEX, RNDEX, CHDEX, CSDEX, CNDEX will extract the matrix **[A*]** from the original matrix file after factoring. Subroutine RSDANN, RNDANN, CHDANN, CSDANN, CNDANN will create a ***FMS*** matrix file from the extracted matrix **[A*]**. The regular ***FMS*** factoring and solving routines can be used to solve the reduced system.

Example

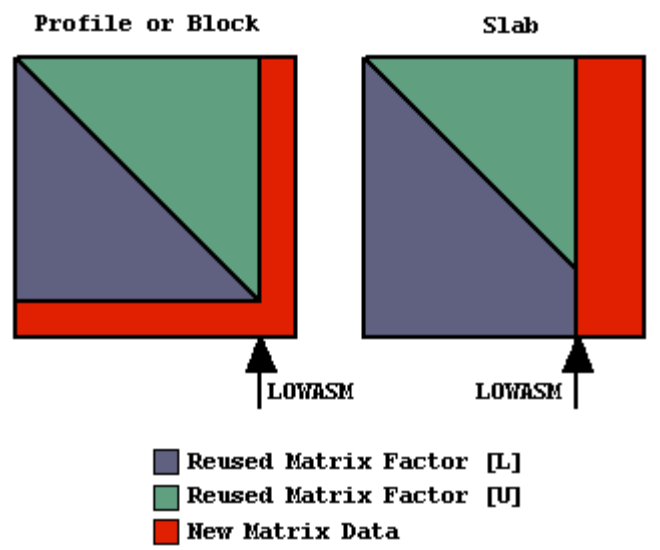
For further illustration on using NEQSUB to perform substructuring, refer to Example 7.

Copyright © Multipath Corporation

4.4. FMS NONLINEAR REASSEMBLY

For some applications the system of equations must be solved several times. This frequently occurs in nonlinear analysis where the matrix coefficients Aij depend on the value of the solution vectors **{X}**. When this occurs significant computation can be saved by starting the second and subsequent solutions at the first changed equation. It is not necessary to know which equations will be changed during the current solution. This information is not available until the solution is completed and the values of **{X}** are known.

After the first solution is completed, the ***FMS*** Parameter LOWASM is used to specify the first equation in the matrix which has a term which changed. The next solution will skip all computation below this value of LOWASM. By properly locating the equations containing nonlinearities at the bottom of the matrix, you may save a significant amount of computation. The value of LOWASM can change with each solution.



The original matrix and factored matrix data must be kept on separate files. The new matrix data, starting at LOWASM, is used to update the original matrix file. During factoring ***FMS*** will round the value of LOWASM down to start on a segment boundary. This prevents records from containing a mix of factored and unfactored data. After the factoring process the original matrix file and the factored matrix file will be based on the current values.

One benefit of this feature is that you do not need to know the restart point in advance when the matrix **[A]** is factored the first time. You simply determine LOWASM as you are processing the solution **{X}** and ***FMS*** automatically skips the appropriate calculations on the next solution.

For example, in a structural finite element analysis, this feature would be used as follows:

1. Initialize your value of LOWASM to 1 greater than the number of equations.

2. Loop over the elements. Use the displacements **{X}** from the previous solution to compute new values of strains and stresses.

3. If nonlinear effects are observed due to large deformations or material nonlinearities, set LOWASM=MIN0(LOWASM,ILOW), where ILOW is the lowest equation in the element.

4. `CALL FMSIST('LOWASM',LOWASM)`

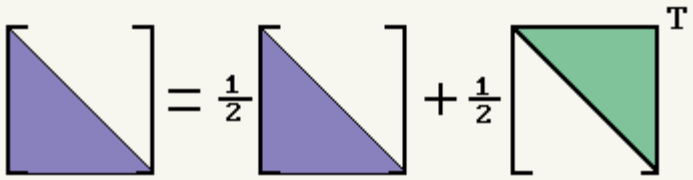
to pass your value of LOWASM to ***FMS***.

This parameter may also be used to recover a factoring process that terminates prematurely. During the initial factoring, set the ***FMS*** Parameter IREST to write the restart data. On the second (recovery) run, set this ***FMS*** parameter LOWASM to the highest factored segment (or slab) printed by ***FMS*** during the first run.

Copyright © Multipath Corporation

4.5. FMS FORCE SYMMETRY

For some applications it is known from the properties of the system that the matrix is symmetric. However in the generation of the matrix coefficients some differences may occur between the lower and upper triangular terms. Since factoring a symmetric matrix requires half the computation of a nonsymmetric matrix, ***FMS*** includes a way to average the lower and upper terms to allow the faster symmetric factoring. The ***FMS*** Parameter LUOK is used to control this feature.



The matrix is first opened and built as a nonsymmetric matrix. The ***FMS*** Parameter LUOK is then set to process this matrix as symmetric, and the symmetric factoring routine is called.

A second application for this feature occurs when matrix data is generated. In some cases generating half of the matrix coefficients as required for a symmetric matrix will naturally populate some terms in the upper triangle and some terms in the lower triangle. Transposing the upper triangle terms to force all data into the lower triangle may be awkward. This feature allows you to provide both lower and upper triangles during matrix assembly and then compact the data into symmetric form after the matrix coefficients have been generated. Note that ***FMS*** automatically initializes all matrix coefficients to zero so only the nonzero values need to be specified.

Copyright © Multipath Corporation

4.6. FMS STRASSEN AND WINOGRAD ALGORITHMS

The ***FMS*** parameter IALGOR directs ***FMS*** to use alternative algorithms when performing matrix multiplies during factoring, solving and matrix-vector multiply. These algorithms are designed to reduce the number of floating point operations performed, thereby providing a faster solution. The following two algorithms are available:

1. Strassen's Algorithm

This algorithm modifies the traditional matrix multiply

`[C] = [A][B]`

by first bisecting each of the matrices **`[A]`** and **`[B]`** into 4 equal parts. By appropriately adding, subtracting and multiplying the resulting quadrants of **`[A]`** and **`[B]`**, it is possible to compute the matrix **`[C]`** with 7/8 of the multiplies required by the traditional technique. Because this algorithm is recursive, the matrices may be bisected several times, with each bisection reducing the remaining work by another factor of 7/8. Ultimately, however, the matrices become too small and the overhead of adding and subtracting the quadrants exceeds the 7/8 benefit. The ***FMS*** parameter MINDIM is used to terminate this process by specifying the minimum dimension of a matrix quadrant at or below which traditional techniques are used.

2. Winograd's Algorithm (Complex data only)

(also called 3M Complex multiply)

This algorithm performs complex matrix multiply using 3 instead of the traditional 4 multiply and add operations. In the traditional case, when two complex numbers A and B are multiplied and added to C, 4 multiplies and 4 adds are required as follows:

```
(Re_C + Im_C i) = (Re_A + Im_A i) * (Re_B + Im_B i)


P1 = Re_A*Re_B
P2 = Re_A*Im_B
P3 = Im_A*Re_B
P4 = Im_A*Im_B


Re_C = Re_C + (P1 - P4)
Im_C = Im_C + (P2 + P3)
```

In this reduced operation case, the same calculation is performed with 3 multiplies as follows:

```
P1 = Re_A*(Re_B - Im_B)
P2 = (Re_A + Im_A)*Re_B
```

```
P3 = (Re_A - Im_A)*Im_B


Re_C = Re_C + (P1 + P3)
Im_C = Im_C + (P2 - P1)
```

At first it would appear that we have not gained anything because, while we have reduced the number of multiplies from 4 to 3, we have increased the number of adds from 4 to 7. Since most computers perform multiplies and adds at the same speed (and many concurrently), we have actually increased the number of cycles to perform the operation.

However when A, B and C become matrices **[A], [B]** and **[C]** the balance shifts. For matrices of size N by N, N square operations are required for addition while 2(N cube) operations are required for multiplication. For sufficiently large N, the extra additions become insignificant.

For complex data both these algorithms may be used. Winograd's algorithm is applied first, producing the three real matrices. Strassen's algorithm may then be applied to perform the real matrix multiplies.

There are, however, some downsides to these algorithms. First, because additions and subtractions are performed ahead of multiplications, some precision will be lost. The amount is problem dependent. You should always check your answer with the traditional technique, IALGOR=0.

Second, the adding and subtracting of the quadrants results in a more random memory access pattern than the traditional technique. For this reason, some of the benefit may be lost in cache thrashing.

Third, current floating point processors fuse the multiplier and adder so the output of the multiplier flows directly into the adder. This development has occurred after these algorithms were developed. When additions are performed, multiply cycles go unused.

Copyright © Multipath Corporation

5. EXAMPLES OF USING FMS

The following examples are FORTRAN programs which illustrate how to use ***FMS***.

There are two copies of each example. The "linked" copy has hyperlinks to all ***FMS*** subroutines and Parameters. This is useful for learning ***FMS***.

The "code" copy has the minimum number of html tags at the beginning and end of the file. This copy is useful for extracting the code and running it on your machine.

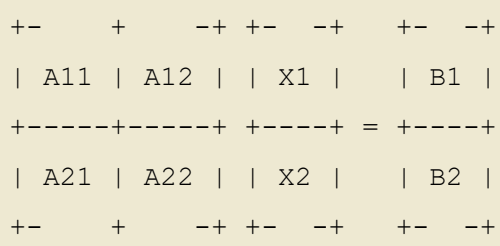
Example 11 is designed for benchmarking. A pre-linked copy is available on this server for downloading. You may configure a problem to meet your needs and then try it on several machines.

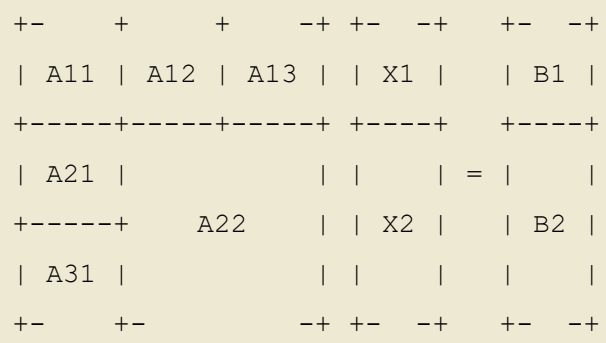
Linked	Description	Code
Example_1	**DESCRIPTION:** Full Incore Matrix opened by CNDANN **INPUT VALUES:** None.	CODE
Example_2:	**DESCRIPTION:** Same as Example 1 but the matrix and vector storage is dynamically allocated using the ***FMS*** memory management system. **INPUT VALUES:** • Number of equations.	CODE
Example_3:	**DESCRIPTION:** General Out-of-core Matrix populated by user-supplied subroutines. **INPUT VALUES:** • Number of equations. • Number of solution vectors.	CODE
Example_4:	**DESCRIPTION:** Finite Element example.	CODE

	INPUT VALUES: None.	
Example_5:	**DESCRIPTION:** Finite Element example using subroutine FMSPF to eliminate equations constrained to zero. **INPUT VALUES:** None.	CODE
Example_6:	**DESCRIPTION:** Finite Element example using wavefront numbering. **INPUT VALUES:** None.	CODE
Example_7:	**DESCRIPTION:** Substructuring **INPUT VALUES:** • Number of equations. • First equation of substructure (1 to N). • Number of solution vectors.	CODE
Example_8:	**DESCRIPTION:** Full complex nonsymmetric matrix populated with FMSROW **INPUT VALUES:** None.	CODE
Example_9:	**DESCRIPTION:** Sample RCS program including using ***FMS*** utilities to fill matrix in parallel. **INPUT VALUES:** • Number of equations. • Number of solution vectors.	CODE
	DESCRIPTION:	

Example_10:	Full Complex Symmetric Matrix populated with user-supplied subroutine CSUBLK. **INPUT VALUES:** • Number of equations. • Number of solution vectors.	CODE
Example_11:	**DESCRIPTION:** Benchmark problem. This example is designed for benchmarking machines The sequence of calculations is as follows: • Populate and factor the matrix **[A]** • Solve **[A]{X} = {B}** • Compare **{X}** to the known solution. **INPUT VALUES:** • Data type and matrix symmetry according to the following: ◦ 1=Real Symmetric ◦ 2=Real Nonsymmetric ◦ 3=Complex Hermitian ◦ 4=Complex Symmetric ◦ 5=Complex Nonsymmetric • Number of equations • Number of solution vectors • Any ***FMS*** Parameter. For benchmarking you may want to review and possibly modify the following: ◦ MFMAT=(1=Profile,2=Block,3=Slab) ◦ MAXCPU=Number of processors to use, ◦ MAXMD=Number of Real*8 words to use, ◦ NOPAGE=Location of page file, ◦ MDINIT=Memory initialization, ◦ NEQBIO=I/O Block size ◦ IALGOR=Reduced operations, ◦ INCORE=(0=disk,1=memory) ◦ NUMIOQ=Number of I/O queues, ◦ NUMSTP=Number of stripes per queue.	CODE

	• RETURN - Ends input of ***FMS*** Parameters and continues processing.	
Example_12:	**DESCRIPTION:** Benchmark problem. This example is similar to Example 11, except the test data forces pivoting of almost every equation. The sequence of calculations is as follows: • Populate **`[A]`** • Factor **`[A]`** • Solve **`[A]{X} = {B}`** • Compute **`[F] = {X}T{Y}`** • Repopulate **`[A]`** • Compute **`{B'} = [A]{X}`** • Compute the error **`{B} - {B'}`** **INPUT VALUES:** Same as Example 11.	CODE
Example_13:	**DESCRIPTION:** This example demonstrates using the FMSPUT subroutine to write matrix and vectors terms in parallel. The sequence of calculations is as follows: • Populate the vector and matrix by calling FMSPUT from subroutines which run in parallel. • Factor the matrix **`[A]`** • Solve **`[A]{X} = {B}`** • Compare **`{X}`** to the known solution. **INPUT VALUES:** • Data type and matrix symmetry according to the following: ◦ 1=Real Symmetric ◦ 2=Real Nonsymmetric ◦ 3=Complex Hermitian ◦ 4=Complex Symmetric ◦ 5=Complex Nonsymmetric	CODE

	Number of equations • Number of solution vectors • Number of rows in the block used to fill the matrix If you enter 0, the I/O block size NEQBIO is used. • Number of columns in the block used to fill the matrix If you enter 0, the I/O block size NEQBIO is used. • Any ***FMS*** Parameter • RETURN - Ends input of ***FMS*** Parameters and continues processing.	
Example_14:	**DESCRIPTION:** This example shows how the ***FMS*** subroutines may be used to perform a sequence of solutions when only some of the matrix terms are changed. The original system is partitioned into the form:  You may change any of the submatrices and only the terms needing re-evaluation are computed. The matrix **[A]** is full, complex nonsymmetric and is factored with full column partial pivoting. This Example consists of a test driver and subroutine RESOLV, which performs the work. RESOLV may be inserted into your application to perform similar analysis. **INPUT VALUES:** • Number of equations in **[A11]** • Number of equations in **[A22]** • Number of solution vectors • Any ***FMS*** Parameter • RETURN - Ends input of ***FMS*** Parameters and continues processing. The system is then solved the first time. When the computation is completed, you are asked the following questions: • Do you want another solution?	CODE

	• Do you want a new A11? • Do you want a new A12? • Do you want a new A21? • Do you want a new A22? • Do you want a new B1? • Do you want a new B2? If you answer yes to any of these questions, the solution is repeated.	
Example_15:	**DESCRIPTION:** This example shows how the ***FMS*** subroutine FMSMM and the ***FMS*** Parameters MMROW, MMCOL and MMKA may be used to solve a system partitioned as follows:  The matrix **[A]** is full. The symmetry may be symmetric, nonsymmetric or Hermitian. The data may be real or complex. Block format (MFMAT=2) is used. **INPUT VALUES:** • The ***FMS*** module number (1 to 5) The ***FMS*** modules are numbered as follows: ◦ 1=Real Symmetric, ◦ 2=Real Nonsymmetric, ◦ 3=Complex Hermitian, ◦ 4=Complex Symmetric, ◦ 5=Complex Nonsymmetric. • Number of equations in **[A11]**, N1 • Number of columns in **[A12]**, N2 • Number of columns in **[A13]**, N3 • Number of solution vectors • Type of test data (1=fixed,2=random)	CODE

	Any ***FMS*** Parameter • RETURN - Ends input of ***FMS*** Parameters and continues processing. Any of the matrix size dimensions N1, N2 or N3 may be set to zero. The system is then solved and the error computed. You are then asked, • Do you want another solution? If you answer yes, the solution is repeated.	
Example_16:	**DESCRIPTION:** This example shows how the ***FMS*** Parameters IREST and LOWASM may be used to restart ***FMS*** during factoring. The sequence of calculations is as follows: • Populate the matrix **[A]** • Factor the matrix **[A]**. This process may be interrupted. • Solve **[A]{X} = {B}** • Compare **{X}** to the known solution. This example is then run a second time to perform the following: • Restart factoring at the specified equation, • Solve **[A]{X} = {B}** • Compare **{X}** to the known solution. **INPUT VALUES:** Is this job being restarted? • For jobs NOT being restarted, the following information is supplied: ◦ The FMS module number, according to the following: ▪ 1=Real Symmetric ▪ 2=Real Nonsymmetric ▪ 3=Complex Hermitian ▪ 4=Complex Symmetric ▪ 5=Complex Nonsymmetric ◦ Number of equations ◦ Number of solution vectors ◦ Any ***FMS*** Parameter. • For jobs being restarted, the input information is as follows: ◦ Enter the highest factored equation	CODE

- Should all files be deleted after this run?

This example uses separate files for the input and factored matrix. If you use the same file, you may need to repopulate part of the unfactored matrix data beyond the restart point. This data may have been partially overwritten by the previous factoring process. The window of potentially modified data is limited to the first segment after the restart point.

DESCRIPTION:

This example shows how the ***FMS*** subroutines may be used to perform a sequence of iterative solutions.

```
 +-    +    -+ +-  -+   +-  -+
 | A11 | A12 | | X1 |   | B1 |
 +-----+-----+ +----+ = +----+
 | A21 | A22 | | X2 |   | B2 |
 +-    +    -+ +-  -+   +-  -+
```

The system of equations is partitioned as shown above. The diagonal blocks [A11] and [A22] are factored but the off-diagonal blocks [A12] and [A21] are not. A sequence of iterative solutions is performed using Jacobi or Gauss-Seidel iteration to obtain approximate solutions {X1} and {X2}.

This approach can save computer time over the exact solution under the following conditions:

- The matrix values in the diagonal blocks [A11] and [A22] are large compared to the values in the off-diagonal blocks [A12] and [A21].
- An approximate solution to {X1} and {X2} is acceptable.
- The number of vectors, times the number of iterations, is less than the dimension of [A22].

In the Jacobi solution, the "old" values of {X1} and {X2} are used to compute the new values of {X1} and {X2}. In the Gauss-Seidel solution, the old value of {X2} is first used to compute a new value of {X1}. This updated value of {X1} is then used to compute the new value of {X2}. Generally the Gauss-Seidel method will converge faster than the Jacobi method.
A variation of the Jacobi method is also presented where the solution values are the change in {DX1} and {DX2}. This approach is numerically equivalent to the standard Jacobi method that solves directly for the solution {X1} and {X2}.

After a solution is obtained, you may solve a similar problem where [A11] and [A22] may change or remain the same. It is assumed that [A12], [A21], {B1} and

Example_17:	{B2} change. The matrix **[A]** is full, complex nonsymmetric and is factored with full column partial pivoting. This Example consists of a test driver and subroutines J_GS and JACOBI2, which performs the work. These subroutines may be inserted into your application to perform similar analysis. **INPUT VALUES:** • Number of equations in **[A11]** • Number of equations in **[A22]** • Number of solution vectors • Maximum number of iterations to perform • Maximum solution error to terminate iterations • Solution method: ◦ 1=Jacobi ◦ 2=Gauss-Seidel ◦ 3=Incremental Jacobi • Vector norm used to test for convergence: ◦ 0=Infinity norm (maximum absolute value) ◦ 1=Sum of the absolute values ◦ 2=Square root of the sum of the values squared • If you want ***FMS*** output during the iteration • Any ***FMS*** Parameter • RETURN - Ends input of ***FMS*** Parameters and continues processing. The system is then solved the first time. When the computation is completed, you are asked the following questions: • Do you want another solution? • Do you want a new A11? • Do you want a new A22? If you answer yes to any of these questions, the solution is repeated. When all solutions of the given problem are complete, you are asked if you want to solve another problem, in which any of the input may change.	CODE
	DESCRIPTION:	

Example_18:	This example illustrates how to solve with **[A]T** using the ***FMS*** Parameter LUTRAN. The expression **V = {C}T [Ainv] {B}** is evaluated by two methods: 1. Solve **[A]{X} = {B}** for **{X}**, then multiply **Vcx = {C}T{X}**, 2. Solve **[A]T {Y} = {C}** for **{Y}**, then multiply **Vyb = {Y}T{B}**. The maximum error in computing **[A]{X}={B}** and the difference between **Vcx** and **Vyb** is computed and printed. The calculation proceeds as follows: • Populate **[A]** with random numbers, • Factor **[A]**, • Solve **[A]{X} = {B}**, • Solve **[A]T{Y} = {C}**, • Repopulate **[A]**, • Compute **{E} = {B} - [A]{X}**, • Compute **Vcx = {C}T{X}**, • Compute **Vyb = {Y}T{B}**, • Compute **Ev = |Vcx - Vyb|**, • Print the maximum value of Ei and Ev **INPUT VALUES:** Same as Example 11, except only the nonsymmetric modules RN and CN can be specified.	CODE
	DESCRIPTION: This example shows how to use the user-supplied subroutines and subroutine xxDAF to read elements of a matrix file. The calculation proceeds as follows: • Populate matrix **[A]** with test data using FMSROW, • Populate vector **{X}** with test data using FMSWRT, • Factor **[A]** into **[F]**, • Solve **[A]{X} = {B}**, • Read **[A]** and compute average values in coarse windows, • Read **[F]** and compute average values in coarse windows,	

	• Check the answer	
Example_19:	**INPUT VALUES:** • Data type and matrix symmetry according to the following: ◦ 1=Real Symmetric ◦ 2=Real Nonsymmetric ◦ 3=Complex Hermitian ◦ 4=Complex Symmetric ◦ 5=Complex Nonsymmetric • Number of equations • Number of solution vectors • Cell size for computing averages. For example, if the matrix has rank 1000 and a cell size of 100 is specified, then an array of 10x10 cells will be computed and printed. Each cell will contain the average value of the 100x100 matrix elements it represents. • Any ***FMS*** Parameter.	CODE
Example_20:	**DESCRIPTION:** This example illustrates the use of subroutine FMSINI2 and ***FMS*** Parameter NOOPEN for estimating disk space requirements for matrix and vector files. The calculation proceeds as follows: • **FMS** is initialized according to the input options selected, • Subroutine FMSIST(' NOOPEN', 1) is called to prevent files from physically being opened, • Subroutine RSDI, RNDI, CHDI, CSDI or CNDI is called to compute the matrix storage requirements but not physically open the file, • The matrix file storage requirements are computed from the matrix file attribute list LUA, • Subroutine FMSOV is called to compute the vector storage requirements but not physically open the file, • The vector file storage requirements are computed from the vector file attribute list LUX, • Subroutines FMSCM and FMSCV are called to close the matrix and vector files, • Subroutine FMSEND is called to end ***FMS***.	CODE

	INPUT VALUES: • Data type and matrix symmetry according to the following: ◦ 1=Real Symmetric ◦ 2=Real Nonsymmetric ◦ 3=Complex Hermitian ◦ 4=Complex Symmetric ◦ 5=Complex Nonsymmetric • Number of equations • Number of solution vectors • Type of initialization ◦ 0=Call FMSINI for a normal initialization, ◦ 1=Call FMSINI2(1) for a reduced initialization, ◦ 2=Call FMSINI2(2) for a reduced initialization with no output. • Any ***FMS*** Parameter.	
Example_21:	**DESCRIPTION:** This example illustrates the use of subroutine FMSMM. The calculation proceeds as follows: • Matrix or vector files [A], {A}, {A}T and {B} are filled with random numbers. • FMSMM is called to compute one of the following: ◦ {C} = [A]{B}, Case 1 ◦ {C} = {A}{B}, Case 3 ◦ {C} = {A}T{B}, Case 4 ◦ [C] = {A}T{B}, Case 5 • FMSMM is called again to compute one of the following: ◦ {C} = {C} - [A]{B}, Case 1 ◦ {C} = {C} - {A}{B}, Case 3 ◦ {C} = {C} - {A}T{B}, Case 4 ◦ [C] = [C] - {A}T{B}, Case 5 • [C] or {C}, which should be [0] or {0}, is checked for errors. **INPUT VALUES:** • Data type and matrix symmetry, according to the following: ◦ 1=Real Symmetric	CODE

	◦ 2=Real Nonsymmetric ◦ 3=Complex Hermitian ◦ 4=Complex Symmetric ◦ 5=Complex Nonsymmetric • The case number, according to the following: ◦ 1, {C} = {D} + s*[A]{B}, ◦ 3, {C} = {D} + s*{A}{B}, ◦ 4, {C} = {D} + s*{A}T{B}, ◦ 5, [C] = [D] + s*{A}T{B} • M, the number of rows in {C} or [C] • N, the number of columns in {C} (except for case 5, where N=M) • K, the number of rows in {B} (except for case 1, where K=M) • Do you want to fill (always answer Y) • Any ***FMS*** Parameter.	
Example_22:	**DESCRIPTION:** I/O Performance Test. This example is designed to test the performance of the ***FMS*** file system. **INPUT VALUES:** • Size of transfers (Mbytes) • Number of transfers • Any ***FMS*** Parameter. For testing you may want to review and possibly modify the following: ◦ IOTYPE=(0=Buffered,1=Raw,2=Direct,3=Flushed) Set this Parameter in the license file. ◦ NUMIOQ=Number of I/O queues, ◦ NUMSTP=Number of stripes per queue. ◦ NUMVOL=Number of volumes per stripe. • RETURN - Ends input of ***FMS*** Parameters and continues processing.	CODE
	DESCRIPTION: Mixing OpenMP and ***FMS*** parallel. Building 2 matrices simultaneously in parallel.	

Example_23:	**INPUT VALUES:** • Number of equations in matrix 1. • Number of equations in matrix 2. • Number of solution vectors. • Do you want to use OpenMP to generate the data (y,n) A y answer uses OpenMP directives; A n answer uses FMS Parallel Processing Routines • Do you want to use OpenMP MUTEX locks (y,n)> A y answer uses !$OMP CRITICAL directives A n answer uses FMSONE, FMSALL routines.	CODE
Example_24:	**DESCRIPTION:** Finite element example using 8-node bricks. A mesh is generated NE_THICK x NE_WIDE x NE_LONG. The nodes are numbered first in the THICK, then in the WIDE and then in the LONG direction. For best performance use THICK <= WIDE <= LONG. **INPUT VALUES:** • Number of elements thick. • Number of elements wide. • Number of elements long. • Number of degrees of freedom per node.	CODE

Copyright © Multipath Corporation

FMS EXAMPLE 1. FULL INCORE MATRIX

This example illustrates how ***FMS*** may be used to solve a full real nonsymmetric matrix **`A(N,N)`** which is dimensioned in your program. In this example step 3 (write data to ***FMS*** files) and step 5 (read data from ***FMS*** files) are not required because ***FMS*** operates directly on data dimensioned in your program. Note that the array **`A(N,N)`** and vector **`X(N)`** are placed in blank common for proper data alignment. Subroutine MATGEN is used to generate test data.

```
C       E X A M P L E   1
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_1')
C
C       Problem size:
        INTEGER     N
        PARAMETER  (N = 100)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no)
        PARAMETER (ISKIP  = 0)
C
C       Matrix and vector:
        COMMON     A(N,N), X(N)
        REAL*8     A, X
C
C       FMS matrix and vector file attributes:

        INTEGER    LUA(25)
```

```
      INTEGER    LUX(25)
C
C     Local variables:
      REAL*8     EI, ERROR
C
C (1) Initialize FMS:
      CALL FMSINI
      CALL FMSPSH(MYNAME)
      CALL FMSIST ('IPRF',1026)
C
C     Populate [A] and {X} with test data:
      CALL MATGEN (A, N, N, X)
C
C (2) Open FMS files:
C     NOTE: The arrays A(N,N) and X(N) are automatically shared on
C     machines using threads.  On machines using separate processes
C     for parallel, they are not automatically shared.  Therefore we
C     will limit the number of processes accessing these arrays to 1
C     for this example.  Example_2 illustrates how to place these
C     arrays in FMS shared memory so they can be operated on by
C     parallel processes on all machines.
      CALL FMSIST ('MAXCPU',1)

      CALL RNDANN (A, N, N, LUA)
      CALL FMSOV2 (N, IDTYPE, NUMRHS, X, N, LUX)
C
C (3) Write data to FMS files: (Not required).
C
C (4) Perform matrix algebra:
      CALL RNDF (LUA, LUA, LUX, LUX,    NUMRED)
      CALL RNDS (LUA, LUX, LUX, NUMRHS, ISKIP)
C
C (5) Read data from FMS files: (Not required).
C
C     Check the answer:
      ERROR = 0.0D0
      DO 50 I = 1,N
         EI = ABS(X(I) - 1.0D0)
```

```
            IF(EI .GT. ERROR) ERROR = EI
   50    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE MATGEN (A, LDA, N, B)
C=======================================================================
         INTEGER    LDA, N
         REAL*8     A(LDA,N), B(N), ZERO, ONE
         CHARACTER*6 MYNAME
         PARAMETER  (ZERO=0.D0)
         PARAMETER  (ONE =1.D0)
         PARAMETER  (MYNAME='MATGEN')
C
C        Populate [A] and {B} with test data:
C                [A]         {X}  =  {B}
C        +-             -+ +   +   +   +
C        | N -1 -1 -1 -1| | 1 |   | 1 |
C        |-1  1  0  0  0| | 1 |   | 0 |
C        |-1  0  1  0  0| | 1 | = | 0 |
C        |-1  0  0  1  0| | 1 |   | 0 |
C        |-1  0  0  0  1| | 1 |   | 0 |
C        +-             -+ +   +   +   +
C
         CALL FMSPSH(MYNAME)
         A(1,1) = DCMPLX(N,N)
         B(1)   = ONE
         DO 20 I = 2,N
            DO 10 J = 2,N
               A(I,J) = ZERO
   10       CONTINUE
            A(1,I) =-ONE
            A(I,1) =-ONE
```

```
            A(I,I) = ONE
            B(I)   = ZERO
   20    CONTINUE
         CALL FMSPOP(MYNAME)
         RETURN
         END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   1
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_1')
C
C       Problem size:
        INTEGER     N
        PARAMETER  (N = 100)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no)
        PARAMETER (ISKIP  = 0)
C
C       Matrix and vector:
        COMMON     A(N,N), X(N)
        REAL*8     A, X
C
C       FMS matrix and vector file attributes:

        INTEGER    LUA(25)
        INTEGER    LUX(25)
C
C       Local variables:
        REAL*8     EI, ERROR
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('IPRF',1026)
C
C       Populate [A] and {X} with test data:
        CALL MATGEN (A, N, N, X)
C
C (2)   Open FMS files:
C       NOTE: The arrays A(N,N) and X(N) are automatically shared on
C       machines using threads.  On machines using separate processes
C       for parallel, they are not automatically shared.  Therefore we
C       will limit the number of processes accessing these arrays to 1
C       for this example.  Example_2 illustrates how to place these
C       arrays in FMS shared memory so they can be operated on by
C       parallel processes on all machines.
        CALL FMSIST ('MAXCPU',1)

        CALL RNDANN (A, N, N, LUA)
        CALL FMSOV2 (N, IDTYPE, NUMRHS, X, N, LUX)
C
C (3)   Write data to FMS files: (Not required).
C
C (4)   Perform matrix algebra:
        CALL RNDF (LUA, LUA, LUX, LUX,    NUMRED)
        CALL RNDS (LUA, LUX, LUX, NUMRHS, ISKIP)
C
C (5)   Read data from FMS files: (Not required).
C
```

```
C       Check the answer:
        ERROR = 0.0D0
        DO 50 I = 1,N
           EI = ABS(X(I) - 1.0D0)
           IF(EI .GT. ERROR) ERROR = EI
   50   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUX)
        CALL FMSPOP(MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE MATGEN (A, LDA, N, B)
C=======================================================================
        INTEGER    LDA, N
        REAL*8     A(LDA,N), B(N), ZERO, ONE
        CHARACTER*6 MYNAME
        PARAMETER  (ZERO=0.D0)
        PARAMETER  (ONE =1.D0)
        PARAMETER  (MYNAME='MATGEN')
C
C       Populate [A] and {B} with test data:
C              [A]         {X}  =  {B}
C       +-            -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-            -+ +   +   +   +
C
        CALL FMSPSH (MYNAME)
        A(1,1) = DCMPLX(N,N)
        B(1)   = ONE
        DO 20 I = 2,N
           DO 10 J = 2,N
              A(I,J) = ZERO
   10      CONTINUE
           A(1,I) =-ONE
           A(I,1) =-ONE
           A(I,I) = ONE
           B(I)   = ZERO
   20   CONTINUE
        CALL FMSPOP(MYNAME)
        RETURN
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 2. FMS MEMORY MANAGEMENT

This example extends example 1 to include the ***FMS*** memory manager. The storage for A(N,N) and X(N) are dynamically allocated at run time. This example is also limited to full matrices which may be stored in memory. If you specify a matrix size N that is too large, fatal error FMS$ERR_MEMORY will result. Subroutine MATGEN from the previous example is used to generate test data.

```
C       E X A M P L E   2
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_2')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER    LUA(25)
        INTEGER    LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
```

```
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C
C     Local variables:
      INTEGER    N, LA, LX
      REAL*8     EI, ERROR
C
C (1) Initialize FMS:
      CALL FMSINI
      CALL FMSPSH(MYNAME)
      CALL FMSIST ('IPRF',1026)
      CALL FMSIGT ('MEMPTR',IMD_PTR)
      CALL FMSIGT ('MEMPTR',RMD_PTR)
      CALL FMSIGT ('MEMPTR',CMD_PTR)
      N = ASK_I ('Enter the number of equations')
C
C     Populate [A] and {X} with test data:
      CALL FMSCMG (CMD, LA, N*N)
      CALL FMSCMG (CMD, LX, N)
      CALL MATGEN (CMD(LA), N, N, CMD(LX))
C
C (2) Open FMS files:
      CALL CNDANN (CMD(LA), N, N, LUA)
      CALL FMSOV2 (N, IDTYPE, NUMRHS, CMD(LX), N, LUX)
C
C (3) Write data to FMS files:(Not required)
C
C (4) Perform matrix algebra:
      CALL CNDF (LUA, LUA, LUX, LUX,    NUMRED)
      CALL CNDS (LUA, LUX, LUX, NUMRHS, ISKIP)
C (5) Read data from FMS files:(Not required)
C
C     Check the answer:
      ERROR  = 0.0D0
      DO 50 I = 1,N
         EI = ABS(CMD(LX-1+I) - 1.0D0)
         IF(EI .GT. ERROR) ERROR = EI
   50 CONTINUE
      WRITE(6,*) 'MAXIMUM ERROR =', ERROR
```

```
C
C (6)   Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUX)
        CALL FMSCMR (CMD, LA, N*N)
        CALL FMSCMR (CMD, LX, N)
        CALL FMSPOP(MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE MATGEN (A, LDA, N, B)
C=======================================================================
        CHARACTER*6 MYNAME
        COMPLEX*16 A(LDA,N), B(N), ZERO, ONE
        PARAMETER  (MYNAME='MATGEN')
        PARAMETER  (ZERO=(0.D0,0.D0), ONE=(1.D0,1.D0))
C
C       Populate [A] and {B} with test data:
C              [A]        {X}  =  {B}
C       +-            -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-            -+ +   +   +   +
C
        CALL FMSPSH(MYNAME)
        A(1,1) = DCMPLX(N,N)
        B(1)   = ONE
        DO 20 I = 2,N
           DO 10 J = 2,N
              A(I,J) = ZERO
   10      CONTINUE
           A(1,I) =-ONE
           A(I,1) =-ONE
           A(I,I) = ONE
           B(I)   = ZERO
   20   CONTINUE
```

```
        CALL FMSPOP(MYNAME)
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   2
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_2')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER    LUA(25)
        INTEGER    LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Local variables:
        INTEGER    N, LA, LX
        REAL*8     EI, ERROR
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('IPRF',1026)
        CALL FMSIGT ('MEMPTR',IMD_PTR)
        CALL FMSIGT ('MEMPTR',RMD_PTR)
        CALL FMSIGT ('MEMPTR',CMD_PTR)
        N = ASK_I ('Enter the number of equations')
C
C       Populate [A] and {X} with test data:
        CALL FMSCMG (CMD, LA, N*N)
        CALL FMSCMG (CMD, LX, N)
        CALL MATGEN (CMD(LA), N, N, CMD(LX))
C
C (2)   Open FMS files:
        CALL CNDANN (CMD(LA), N, N, LUA)
        CALL FMSOV2 (N, IDTYPE, NUMRHS, CMD(LX), N, LUX)
C
C (3)   Write data to FMS files:(Not required)
C
C (4)   Perform matrix algebra:
        CALL CNDF (LUA, LUA, LUX, LUX,    NUMRED)
        CALL CNDS (LUA, LUX, LUX, NUMRHS, ISKIP)
C (5)   Read data from FMS files:(Not required)
C
C       Check the answer:
```

```
         ERROR  = 0.0D0
         DO 50 I = 1,N
            EI = ABS(CMD(LX-1+I) - 1.0D0)
            IF(EI .GT. ERROR) ERROR = EI
   50    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         CALL FMSCMR (CMD, LA, N*N)
         CALL FMSCMR (CMD, LX, N)
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE MATGEN (A, LDA, N, B)
C=======================================================================
         CHARACTER*6 MYNAME
         COMPLEX*16 A(LDA,N), B(N), ZERO, ONE
         PARAMETER  (MYNAME='MATGEN')
         PARAMETER  (ZERO=(0.D0,0.D0), ONE=(1.D0,1.D0))
C
C        Populate [A] and {B} with test data:
C               [A]         {X}  =  {B}
C        +-             -+ +   +   +   +
C        | N -1 -1 -1 -1| | 1 |   | 1 |
C        |-1  1  0  0  0| | 1 |   | 0 |
C        |-1  0  1  0  0| | 1 | = | 0 |
C        |-1  0  0  1  0| | 1 |   | 0 |
C        |-1  0  0  0  1| | 1 |   | 0 |
C        +-             -+ +   +   +   +
C
         CALL FMSPSH(MYNAME)
         A(1,1) = DCMPLX(N,N)
         B(1)   = ONE
         DO 20 I = 2,N
            DO 10 J = 2,N
               A(I,J) = ZERO
   10       CONTINUE
            A(1,I) =-ONE
            A(I,1) =-ONE
            A(I,I) = ONE
            B(I)   = ZERO
   20    CONTINUE
         CALL FMSPOP(MYNAME)
         RETURN
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
         CHARACTER* (*) STRING
         WRITE(6,2000) STRING
         READ (5,*) ASK_I
         RETURN
 2000    FORMAT (1X,A,'>')
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 3. GENERAL OUT-OF-CORE MATRIX

This example extends the application of ***FMS*** to problems which exceed physical or virtual memory. Subroutines CSUBLK and CNUBLK are used to define matrix coefficients in the lower triangle, diagonal and upper triangle. The matrix size is only limited by the available disk space.

```
C       E X A M P L E   3
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_3')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
```

```
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       Local variables:
        INTEGER     I, LX, LENX, LDISK, NV
        REAL*8      EI, ERROR
C
C       Common block to communicate with CSUBLK:
        COMMON /MYDATA/N, NRHS
        DATA LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
        CALL FMSIST ('IPRF', 1026)
        CALL FMSIST ('MDATAU', 2)
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        N    = ASK_I ('Enter the number of equations')
        NRHS = ASK_I ('Enter the number of solution vectors')
C
C (2)   Open FMS files:
        CALL CNDI (LOWEQ, N, 'LUF', LUF)
        CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C       Populate test vector:
        LENX = LUX(4)/2
        CALL FMSCMG (CMD, LX, LENX)
        DO 10 I = 1,N
           CMD(LX-1+I) = (0.0D0,0.0D0)
   10   CONTINUE
        CMD(LX) = (1.0D0,1.0D0)
C
C (3)   Write data to FMS files:
        LDISK  = 1
        DO 30 NV = 1,NRHS
           CALL FMSWRT (LUX(1), LDISK, CMD(LX), LUX(4))
```

```
            LDISK = LDISK + LUX(4)
   30    CONTINUE
C
C (4)    Perform matrix algebra:
         CALL CNDF (LUF, LUF, LUX, LUX,  NUMRED)
         CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
         LDISK  = 1
         DO 60 NV = 1,NRHS
            CALL FMSRED (LUX(1), LDISK, CMD(LX), LUX(4))
            LDISK  = LDISK  + LUX(4)
            DO 50 I = 1,N
               EI = ABS(CMD(LX-1+I) - 1.0D0)
               IF(EI .GT. ERROR) ERROR = EI
   50       CONTINUE
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUF)
         CALL FMSCV (LUX)
         CALL FMSCMR (CMD, LX, LENX)
         CALL FMSPop(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         COMPLEX*16 A(0:*), D(*), ONE
         PARAMETER  (ONE=(1.0D0,1.0D0))
         COMMON /MYDATA/N, NRS
C
```

```
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = ONE
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = -ONE
   20   CONTINUE
        RETURN
        END
C=======================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), ONE
        PARAMETER  (ONE=(1.0D0,1.0D0))
C
C       Populate profile of [AU] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 10 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = -ONE
   10   CONTINUE
        RETURN
        END
C=======================================================================
```

```
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   3
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_3')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       Local variables:
        INTEGER     I, LX, LENX, LDISK, NV
        REAL*8      EI, ERROR
C
C       Common block to communicate with CSUBLK:
        COMMON /MYDATA/N, NRHS
        DATA LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('IPRF', 1026)
        CALL FMSIST ('MDATAU', 2)
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        N    = ASK_I ('Enter the number of equations')
        NRHS = ASK_I ('Enter the number of solution vectors')
C
C (2)   Open FMS files:
        CALL CNDI (LOWEQ, N, 'LUF', LUF)
        CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C       Populate test vector:
        LENX = LUX(4)/2
        CALL FMSCMG (CMD, LX, LENX)
        DO 10 I = 1,N
           CMD(LX-1+I) = (0.0D0,0.0D0)
   10   CONTINUE
        CMD(LX) = (1.0D0,1.0D0)
```

```
C
C (3)   Write data to FMS files:
        LDISK  = 1
        DO 30 NV = 1,NRHS
           CALL FMSWRT (LUX(1), LDISK, CMD(LX), LUX(4))
           LDISK = LDISK + LUX(4)
   30   CONTINUE
C
C (4)   Perform matrix algebra:
        CALL CNDF (LUF, LUF, LUX, LUX,  NUMRED)
        CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Check the answer:
        ERROR = 0.0D0
        LDISK  = 1
        DO 60 NV = 1,NRHS
           CALL FMSRED (LUX(1), LDISK, CMD(LX), LUX(4))
           LDISK  = LDISK  + LUX(4)
           DO 50 I = 1,N
              EI = ABS(CMD(LX-1+I) - 1.0D0)
              IF(EI .GT. ERROR) ERROR = EI
   50      CONTINUE
   60   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUF)
        CALL FMSCV (LUX)
        CALL FMSCMR (CMD, LX, LENX)
        CALL FMSPOP (MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), ONE
        PARAMETER  (ONE=(1.0D0,1.0D0))
        COMMON /MYDATA/N, NRS
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = ONE
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = -ONE
   20   CONTINUE
        RETURN
        END
C=======================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
```

```
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), ONE
        PARAMETER  (ONE=(1.0D0,1.0D0))
C
C       Populate profile of [AU] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 10 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = -ONE
   10   CONTINUE
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 4. FINITE ELEMENT INTERFACE

This example illustrates how ***FMS*** may be incorporated into a finite element program. Subroutine ELGEN is used to generate a test finite element mesh which is NEWIDE elements wide and NELONG elements long.

```
C       E X A M P L E   4
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_4')
C
C       Problem size parameters:
C
C       Number of elements wide and long:
        PARAMETER (NEWIDE = 20, NELONG = 100)
C
C       Number of nodes wide and long:
        PARAMETER (NPWIDE = NEWIDE+1, NPLONG = NELONG+1)
C
C       Number of elements and equations:
        PARAMETER (NUMEL  = NEWIDE*NELONG, NUMEQ = NPWIDE*NPLONG)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
C       Length of element real record:
        PARAMETER (LENR = 4*4)
C
C       Length of element integer record:
        PARAMETER (LENI = 2 + 4)
C
```

```
C       Length of element vector record:
        PARAMETER (LENV = 2)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
C       (Performed during call to RSDF)
        PARAMETER (ISKIP  = 1)
C
C       FMS matrix file attributes:
        INTEGER   LUA(25)
        INTEGER   LUX(10), LUB(10), LUAX(10)
        INTEGER   LUS(25)
C
C       Profile vector and equation numbers:
        INTEGER   LOWEQ(NUMEQ), NEWNUM(NUMEQ)
        INTEGER   IDUM(1)
C
C       Solution vector {X}
C       RHS vector      {B}
C       Product [A]{X}, {AX}
C       Diagonals       {D}
        REAL*8    X(NUMEQ), B(NUMEQ), AX(NUMEQ), D(NUMEQ)
        COMMON    X, B, AX, D
C
C       Big diagonal value to impose constraints:
        REAL*8     BIGNUM
        PARAMETER (BIGNUM = 1.0D30)
C
C       Local variables:
        REAL*8    EI, ERROR
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
        CALL FMSIST ('MPOSDF',1)
        CALL FMSIST ('INCORE',2)
C
```

```
C (2)   Open FMS files:
        CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C       Generate test data:
        CALL ELGEN (NEWIDE, NELONG, NUMEL, NEWNUM, LUS, B, NUMEQ)
        WRITE(6,*) 'NUMBER OF ELEMENTS  = ', NUMEL
        WRITE(6,*) 'NUMBER OF EQUATIONS = ', NUMEQ
C       Compute profile vector:
        CALL FMSPF  (LUS, NUMSF, IDUM, 0, LOWEQ, NUMEQ)
        CALL RSDI   (LOWEQ, NUMEQ, 'LUA', LUA)
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  X, NUMEQ, LUX)
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  B, NUMEQ, LUB)
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, AX, NUMEQ, LUAX)
C
C (3)   Write data to FMS files:
        CALL RSDA (LUS, NUMSF, LUA)
C       Impose boundary conditions on diagonal and RHS vector:
        CALL FMSRED (LUA(2), 1, D, NUMEQ)
        DO 31 I = 1,NUMEQ
           INEW   = NEWNUM(I)
           IF(INEW .LT. 0) THEN
              B(I) = BIGNUM*B(I)
              D(I) = BIGNUM
           END IF
   31   CONTINUE
        CALL FMSWRT (LUA(2), 1, D, NUMEQ)
C
C (4)   Perform matrix algebra:
        CALL RSDF (LUA, LUA, LUB, LUX,    NUMRHS)
        CALL RSDS (LUA, LUB, LUX, NUMRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Not required.
C
C       Compute [A]{X} from SUM([Si]{X}) to check error:
        CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
        ERROR = 0.0D0
        DO 50 I = 1,NUMEQ
           INEW   = NEWNUM(I)
           IF(INEW .GT. 0) THEN
```

```
C             Value which was determined:
              EI = ABS(AX(I)-B(I))
              IF(EI .GT. ERROR) ERROR = EI
           END IF
   50   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =',ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUAX)
        CALL FMSCV (LUX)
        CALL FMSCV (LUB)
        CALL FMSCS (LUS)
        CALL FMSPOP(MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE ELGEN (NEWIDE,NELONG,NUMEL,NEWNUM,LUS,B,NUMEQ)
C=======================================================================
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='ELGEN')
        INTEGER NEWIDE, NELONG, NUMEL, NUMEQ
        INTEGER LUS(25), NEWNUM(NUMEQ)
        INTEGER IREC(6), NSUB, ITYPE, ID(4)
        REAL*8  S(4,4), B(NUMEQ)
        EQUIVALENCE (IREC(1),NSUB), (IREC(2),ITYPE), (IREC(3),ID)
C
        CALL FMSPSH(MYNAME)
C
C       Generate a 4-node quadrilateral finite element mesh that
C       is NEWIDE elements wide by NELONG elements long.  Use the
C       same matrix S(4,4) for all elements. Impose the following
C       solution values: X=0 along the top, X=1 along the bottom.
C
C       Generate element connectivity array ID(4):
C       Write element integer records:
        CALL FMSEEK (LUS(2), 0)
        NSUB   = 4
        ITYPE  = 1
```

```
         NPWIDE = NEWIDE + 1
         DO 20 I = 1,NELONG
            DO 10 J = 1,NEWIDE
               ID(1) = I*NPWIDE + J
               ID(2) = ID(1) + 1
               ID(3) = ID(2) - NPWIDE
               ID(4) = ID(1) - NPWIDE
               CALL FMSSWR (LUS(2), IREC, 6)
   10       CONTINUE
   20    CONTINUE
C
C        Generate element data array S(4,4):
         DO 40 I = 1,4
            DO 30 J = 1,4
               S(I,J) = -1.0D0
   30       CONTINUE
            S(I,I) = 3.0D0
   40    CONTINUE
C
C        Write element real records:
         CALL FMSEEK (LUS(1), 0)
         DO 50 NE = 1,NUMEL
            CALL FMSSWR (LUS(1), S, 16)
   50    CONTINUE
C
C        Generate right-hand side vector B(NUMEQ)
C        Generate boundary condition code vector NEWNUM(NUMEQ)
         DO 60 I = 1,NUMEQ
            B(I) = 0.0D0
            NEWNUM(I) = 1
   60    CONTINUE
C        Specify X(1:NPWIDE) = 0, X(NUMEQ-NPWIDE+1:NUMEQ) = 1
         I2 = NUMEQ - NPWIDE
         DO 70 I1 = 1,NPWIDE
            NEWNUM(I1) = 0
            B(I1)   = 0.0D0
            I2      = I2 + 1
            NEWNUM(I2) = -1
```

```
         B(I2)   = 1.0D0
70    CONTINUE
      CALL FMSPOP (MYNAME)
      RETURN
      END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   4
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_4')
C
C       Problem size parameters:
C
C       Number of elements wide and long:
        PARAMETER (NEWIDE = 20, NELONG = 100)
C
C       Number of nodes wide and long:
        PARAMETER (NPWIDE = NEWIDE+1, NPLONG = NELONG+1)
C
C       Number of elements and equations:
        PARAMETER (NUMEL  = NEWIDE*NELONG, NUMEQ = NPWIDE*NPLONG)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
C       Length of element real record:
        PARAMETER (LENR = 4*4)
C
C       Length of element integer record:
        PARAMETER (LENI = 2 + 4)
C
C       Length of element vector record:
        PARAMETER (LENV = 2)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
C       (Performed during call to RSDF)
        PARAMETER (ISKIP  = 1)
C
C       FMS matrix file attributes:
        INTEGER   LUA(25)
        INTEGER   LUX(25), LUB(25), LUAX(25)
        INTEGER   LUS(25)
C
C       Profile vector and equation numbers:
        INTEGER   LOWEQ(NUMEQ), NEWNUM(NUMEQ)
        INTEGER   IDUM(1)
C
C       Solution vector {X}
C       RHS vector      {B}
C       Product [A]{X}, {AX}
C       Diagonals       {D}
        REAL*8    X(NUMEQ), B(NUMEQ), AX(NUMEQ), D(NUMEQ)
        COMMON    X, B, AX, D
C
C       Big diagonal value to impose constraints:
        REAL*8     BIGNUM
        PARAMETER (BIGNUM = 1.0D30)
C
C       Local variables:
```

```
      REAL*8    EI, ERROR
C
C (1)   Initialize FMS:
      CALL FMSINI
      CALL FMSPSH (MYNAME)
      CALL FMSIST ('MPOSDF',1)
      CALL FMSIST ('INCORE',2)
C
C (2)   Open FMS files:
      CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C       Generate test data:
      CALL ELGEN (NEWIDE, NELONG, NUMEL, NEWNUM, LUS, B, NUMEQ)
      WRITE(6,*) 'NUMBER OF ELEMENTS  = ', NUMEL
      WRITE(6,*) 'NUMBER OF EQUATIONS = ', NUMEQ
C       Compute profile vector:
      CALL FMSPF  (LUS, NUMSF, IDUM, 0, LOWEQ, NUMEQ)
      CALL RSDI   (LOWEQ, NUMEQ, 'LUA', LUA)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  X, NUMEQ, LUX)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  B, NUMEQ, LUB)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, AX, NUMEQ, LUAX)
C
C (3)   Write data to FMS files:
      CALL RSDA (LUS, NUMSF, LUA)
C       Impose boundary conditions on diagonal and RHS vector:
      CALL FMSRED (LUA(2), 1, D, NUMEQ)
      DO 31 I = 1,NUMEQ
         INEW   = NEWNUM(I)
         IF(INEW .LT. 0) THEN
            B(I) = BIGNUM*B(I)
            D(I) = BIGNUM
         END IF
   31   CONTINUE
      CALL FMSWRT (LUA(2), 1, D, NUMEQ)
C
C (4)   Perform matrix algebra:
      CALL RSDF (LUA, LUA, LUB, LUX,    NUMRHS)
      CALL RSDS (LUA, LUB, LUX, NUMRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Not required.
C
C       Compute [A]{X} from SUM([Si]{X}) to check error:
      CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
      ERROR = 0.0D0
      DO 50 I = 1,NUMEQ
         INEW   = NEWNUM(I)
         IF(INEW .GT. 0) THEN
C           Value which was determined:
            EI = ABS(AX(I)-B(I))
            IF(EI .GT. ERROR) ERROR = EI
         END IF
   50   CONTINUE
      WRITE(6,*) 'MAXIMUM ERROR =',ERROR
C
C (6)   Close FMS files:
      CALL FMSCM (LUA)
      CALL FMSCV (LUAX)
      CALL FMSCV (LUX)
      CALL FMSCV (LUB)
      CALL FMSCS (LUS)
      CALL FMSPOP (MYNAME)
      CALL FMSEND
      END
C=======================================================================
      SUBROUTINE ELGEN (NEWIDE,NELONG,NUMEL,NEWNUM,LUS,B,NUMEQ)
```

```
C=======================================================================
      CHARACTER*5 MYNAME
      PARAMETER (MYNAME='ELGEN')
      INTEGER NEWIDE, NELONG, NUMEL, NUMEQ
      INTEGER LUS(25), NEWNUM(NUMEQ)
      INTEGER IREC(6), NSUB, ITYPE, ID(4)
      REAL*8  S(4,4), B(NUMEQ)
      EQUIVALENCE (IREC(1),NSUB), (IREC(2),ITYPE), (IREC(3),ID)
C
      CALL FMSPSH (MYNAME)
C
C     Generate a 4-node quadrilateral finite element mesh that
C     is NEWIDE elements wide by NELONG elements long.  Use the
C     same matrix S(4,4) for all elements. Impose the following
C     solution values: X=0 along the top, X=1 along the bottom.
C
C     Generate element connectivity array ID(4):
C     Write element integer records:
      CALL FMSEEK (LUS(2), 0)
      NSUB   = 4
      ITYPE  = 1
      NPWIDE = NEWIDE + 1
      DO 20 I = 1,NELONG
         DO 10 J = 1,NEWIDE
            ID(1) = I*NPWIDE + J
            ID(2) = ID(1) + 1
            ID(3) = ID(2) - NPWIDE
            ID(4) = ID(1) - NPWIDE
            CALL FMSSWR (LUS(2), IREC, 6)
   10    CONTINUE
   20 CONTINUE
C
C     Generate element data array S(4,4):
      DO 40 I = 1,4
         DO 30 J = 1,4
            S(I,J) = -1.0D0
   30    CONTINUE
         S(I,I) = 3.0D0
   40 CONTINUE
C
C     Write element real records:
      CALL FMSEEK (LUS(1), 0)
      DO 50 NE = 1,NUMEL
         CALL FMSSWR (LUS(1), S, 16)
   50 CONTINUE
C
C     Generate right-hand side vector B(NUMEQ)
C     Generate boundary condition code vector NEWNUM(NUMEQ)
      DO 60 I = 1,NUMEQ
         B(I) = 0.0D0
         NEWNUM(I) = 1
   60 CONTINUE
C     Specify X(1:NPWIDE) = 0, X(NUMEQ-NPWIDE+1:NUMEQ) = 1
      I2 = NUMEQ - NPWIDE
      DO 70 I1 = 1,NPWIDE
         NEWNUM(I1) = 0
         B(I1)   = 0.0D0
         I2      = I2 + 1
         NEWNUM(I2) = -1
         B(I2)   = 1.0D0
   70 CONTINUE
      CALL FMSPOP (MYNAME)
      RETURN
      END
```

Copyright © Multipath Corporation

FMS EXAMPLE 5. ELIMINATING CONSTRAINED EQUATIONS

This example illustrates how subroutine FMSPF may be used to eliminate equations whose solution value is constrained to zero and to modify the profile of nonzero constrained equations to a diagonal only term. The vector NEWNUM is used to relate new equation numbers in the condensed system to old equation numbers of the original system.

```
C       E X A M P L E   5
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_5')
C
C       Problem size parameters:
C
C       Number of elements wide and long:
        PARAMETER (NEWIDE = 20, NELONG = 100)
C
C       Number of nodes wide and long:
        PARAMETER (NPWIDE = NEWIDE+1, NPLONG = NELONG+1)
C
C       Number of elements and equations:
        PARAMETER (NUMEL  = NEWIDE*NELONG, MAXEQ = NPWIDE*NPLONG)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
C       Length of element real record:
        PARAMETER (LENR = 4*4)
```

```
C
C       Length of element integer record:
        PARAMETER (LENI = 2 + 4)
C
C       Length of element vector record:
        PARAMETER (LENV = 2)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
C       (Performed during call to RSDF)
        PARAMETER (ISKIP  = 1)
C
C       FMS matrix and vector file attributes:
        INTEGER   LUA(25)
        INTEGER   LUX(25), LUB(25), LUAX(25)
        INTEGER   LUS(25)
C
C       Profile vector and equation numbers:
        INTEGER   LOWEQ(MAXEQ), NEWNUM(MAXEQ)
C
C       Solution vector {X}
C       RHS vector      {B}
C       Product [A]{X}, {AX}
C       Diagonals       {D}
        REAL*8    X(MAXEQ), B(MAXEQ), AX(MAXEQ), D(MAXEQ)
        COMMON    X, B, AX, D
C
C       Big diagonal value to impose constraints:
        REAL*8     BIGNUM
        PARAMETER (BIGNUM = 1.0D30)
C
C       Local variables:
        REAL*8    EI, ERROR
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
```

```
      CALL FMSIST ('MPOSDF',1)
      CALL FMSIST ('INCORE',2)
C
C (2) Open FMS files:
      CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C     Generate test data:
      CALL ELGEN (NEWIDE, NELONG, NUMEL, NEWNUM, LUS, B, MAXEQ)
      WRITE(6,*) 'NUMBER OF ELEMENTS          = ', NUMEL
      WRITE(6,*) 'INITIAL NUMBER OF EQUATIONS = ', MAXEQ
C     Compute profile vector; eliminate zero constraints:
      CALL FMSPF (LUS, NUMSF, NEWNUM, MAXEQ, LOWEQ, NUMEQ)
      WRITE(6,*) 'ACTUAL  NUMBER OF EQUATIONS = ', NUMEQ
      CALL RSDI   (LOWEQ, NUMEQ, 'LUA', LUA)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  X, NUMEQ, LUX)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  B, NUMEQ, LUB)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, AX, NUMEQ, LUAX)
C
C (3) Write data to FMS files:
      CALL RSDA (LUS, NUMSF, LUA)
C     Impose boundary conditions on diagonal and RHS vector:
C     Reorder RHS terms for new numbering system:
      CALL FMSRED (LUA(2), 1, D, NUMEQ)
      DO 31 I = 1,MAXEQ
         INEW   = NEWNUM(I)
         IF(INEW .EQ. 0) GO TO 31
         IF(INEW .GT. 0) THEN
            B(INEW) = B(I)
         ELSE
            INEW   = -INEW
            B(INEW) = BIGNUM*B(I)
            D(INEW) = BIGNUM
         END IF
   31 CONTINUE
      CALL FMSWRT (LUA(2), 1, D, NUMEQ)
C
C (4) Perform matrix algebra:
      CALL RSDF (LUA, LUA, LUB, LUX,    NUMRHS)
      CALL RSDS (LUA, LUB, LUX, NUMRHS, ISKIP)
C
```

```
C (5)   Read data from FMS files:
C       Not required.
C
C       Compute [A]{X} from SUM([Si]{X}) to check error:
        CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
        ERROR = 0.0D0
C       Expand solution vector to include zero constrained values:
        DO 50 I = MAXEQ,1,-1
           INEW   = NEWNUM(I)
           IF(INEW .EQ. 0) THEN
C             Constrained zero value:
              X(I) = 0.0D0
           ELSE IF(INEW .LT. 0) THEN
C             Constrained nonzero value:
              INEW = -INEW
              X(I) = X(INEW)
           ELSE
C             Value which was determined:
              X(I) = X(INEW)
              EI = ABS(AX(INEW)-B(INEW))
              IF(EI .GT. ERROR) ERROR = EI
           END IF
   50   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =',ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUAX)
        CALL FMSCV (LUX)
        CALL FMSCV (LUB)
        CALL FMSCS (LUS)
        CALL FMSPOP(MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE ELGEN (NEWIDE,NELONG,NUMEL,NEWNUM,LUS,B,NUMEQ)
C=======================================================================
        CHARACTER*5 MYNAME
```

```
      PARAMETER (MYNAME='ELGEN')
      INTEGER NEWIDE, NELONG, NUMEL, NUMEQ
      INTEGER LUS(25), NEWNUM(NUMEQ)
      INTEGER IREC(6), NSUB, ITYPE, ID(4)
      REAL*8  S(4,4), B(NUMEQ)
      EQUIVALENCE (IREC(1),NSUB), (IREC(2),ITYPE), (IREC(3),ID)
C
      CALL FMSPSH(MYNAME)
C
C     Generate a 4-node quadrilateral finite element mesh that
C     is NEWIDE elements wide by NELONG elements long.  Use the
C     same matrix S(4,4) for all elements. Impose the following
C     solution values: X=0 along the top, X=1 along the bottom.
C
C     Generate element connectivity array ID(4):
C     Write element integer records:
      CALL FMSEEK (LUS(2), 0)
      NSUB   = 4
      ITYPE  = 1
      NPWIDE = NEWIDE + 1
      DO 20 I = 1,NELONG
         DO 10 J = 1,NEWIDE
            ID(1) = I*NPWIDE + J
            ID(2) = ID(1) + 1
            ID(3) = ID(2) - NPWIDE
            ID(4) = ID(1) - NPWIDE
            CALL FMSSWR (LUS(2), IREC, 6)
   10    CONTINUE
   20 CONTINUE
C
C     Generate element data array S(4,4):
      DO 40 I = 1,4
         DO 30 J = 1,4
            S(I,J) = -1.0D0
   30    CONTINUE
         S(I,I) = 3.0D0
   40 CONTINUE
C
C     Write element real records:
```

```
          CALL FMSEEK (LUS(1), 0)
          DO 50 NE = 1,NUMEL
             CALL FMSSWR (LUS(1), S, 16)
    50    CONTINUE
C
C         Generate right-hand side vector B(NUMEQ)
C         Generate boundary condition code vector NEWNUM(NUMEQ)
          DO 60 I = 1,NUMEQ
             B(I) = 0.0D0
             NEWNUM(I) = 1
    60    CONTINUE
C         Specify X(1:NPWIDE) = 0, X(NUMEQ-NPWIDE+1:NUMEQ) = 1
          I2 = NUMEQ - NPWIDE
          DO 70 I1 = 1,NPWIDE
             NEWNUM(I1) = 0
             B(I1)   = 0.0D0
             I2      = I2 + 1
             NEWNUM(I2) = -1
             B(I2)   = 1.0D0
    70    CONTINUE
          CALL FMSPOP(MYNAME)
          RETURN
          END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   5
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_5')
C
C       Problem size parameters:
C
C       Number of elements wide and long:
        PARAMETER (NEWIDE = 20, NELONG = 100)
C
C       Number of nodes wide and long:
        PARAMETER (NPWIDE = NEWIDE+1, NPLONG = NELONG+1)
C
C       Number of elements and equations:
        PARAMETER (NUMEL  = NEWIDE*NELONG, MAXEQ = NPWIDE*NPLONG)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
C       Length of element real record:
        PARAMETER (LENR = 4*4)
C
C       Length of element integer record:
        PARAMETER (LENI = 2 + 4)
C
C       Length of element vector record:
        PARAMETER (LENV = 2)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
C       (Performed during call to RSDF)
        PARAMETER (ISKIP  = 1)
C
C       FMS matrix and vector file attributes:
        INTEGER   LUA(25)
        INTEGER   LUX(25), LUB(25), LUAX(25)
        INTEGER   LUS(25)
C
C       Profile vector and equation numbers:
        INTEGER   LOWEQ(MAXEQ), NEWNUM(MAXEQ)
C
C       Solution vector {X}
C       RHS vector      {B}
C       Product [A]{X}, {AX}
C       Diagonals       {D}
        REAL*8    X(MAXEQ), B(MAXEQ), AX(MAXEQ), D(MAXEQ)
        COMMON    X, B, AX, D
C
C       Big diagonal value to impose constraints:
        REAL*8     BIGNUM
        PARAMETER (BIGNUM = 1.0D30)
C
C       Local variables:
        REAL*8    EI, ERROR
```

```
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('MPOSDF',1)
        CALL FMSIST ('INCORE',2)
C
C (2)   Open FMS files:
        CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C       Generate test data:
        CALL ELGEN (NEWIDE, NELONG, NUMEL, NEWNUM, LUS, B, MAXEQ)
        WRITE(6,*) 'NUMBER OF ELEMENTS          = ', NUMEL
        WRITE(6,*) 'INITIAL NUMBER OF EQUATIONS = ', MAXEQ
C       Compute profile vector; eliminate zero constraints:
        CALL FMSPF (LUS, NUMSF, NEWNUM, MAXEQ, LOWEQ, NUMEQ)
        WRITE(6,*) 'ACTUAL  NUMBER OF EQUATIONS = ', NUMEQ
        CALL RSDI   (LOWEQ, NUMEQ, 'LUA', LUA)
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  X, NUMEQ, LUX)
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  B, NUMEQ, LUB)
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, AX, NUMEQ, LUAX)
C
C (3)   Write data to FMS files:
        CALL RSDA (LUS, NUMSF, LUA)
C       Impose boundary conditions on diagonal and RHS vector:
C       Reorder RHS terms for new numbering system:
        CALL FMSRED (LUA(2), 1, D, NUMEQ)
        DO 31 I = 1,MAXEQ
           INEW   = NEWNUM(I)
           IF(INEW .EQ. 0) GO TO 31
           IF(INEW .GT. 0) THEN
              B(INEW) = B(I)
           ELSE
              INEW   = -INEW
              B(INEW) = BIGNUM*B(I)
              D(INEW) = BIGNUM
           END IF
   31   CONTINUE
        CALL FMSWRT (LUA(2), 1, D, NUMEQ)
C
C (4)   Perform matrix algebra:
        CALL RSDF (LUA, LUA, LUB, LUX,    NUMRHS)
        CALL RSDS (LUA, LUB, LUX, NUMRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Not required.
C
C       Compute [A]{X} from SUM([Si]{X}) to check error:
        CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
        ERROR = 0.0D0
C       Expand solution vector to include zero constrained values:
        DO 50 I = MAXEQ,1,-1
           INEW   = NEWNUM(I)
           IF(INEW .EQ. 0) THEN
C             Constrained zero value:
              X(I) = 0.0D0
           ELSE IF(INEW .LT. 0) THEN
C             Constrained nonzero value:
              INEW = -INEW
              X(I) = X(INEW)
           ELSE
C             Value which was determined:
              X(I) = X(INEW)
              EI = ABS(AX(INEW)-B(INEW))
              IF(EI .GT. ERROR) ERROR = EI
           END IF
```

```
   50   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =',ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUAX)
        CALL FMSCV (LUX)
        CALL FMSCV (LUB)
        CALL FMSCS (LUS)
        CALL FMSPOP (MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE ELGEN (NEWIDE,NELONG,NUMEL,NEWNUM,LUS,B,NUMEQ)
C=======================================================================
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='ELGEN')
        INTEGER NEWIDE, NELONG, NUMEL, NUMEQ
        INTEGER LUS(25), NEWNUM(NUMEQ)
        INTEGER IREC(6), NSUB, ITYPE, ID(4)
        REAL*8  S(4,4), B(NUMEQ)
        EQUIVALENCE (IREC(1),NSUB), (IREC(2),ITYPE), (IREC(3),ID)
C
        CALL FMSPSH (MYNAME)
C
C       Generate a 4-node quadrilateral finite element mesh that
C       is NEWIDE elements wide by NELONG elements long.  Use the
C       same matrix S(4,4) for all elements. Impose the following
C       solution values: X=0 along the top, X=1 along the bottom.
C
C       Generate element connectivity array ID(4):
C       Write element integer records:
        CALL FMSEEK (LUS(2), 0)
        NSUB   = 4
        ITYPE  = 1
        NPWIDE = NEWIDE + 1
        DO 20 I = 1,NELONG
           DO 10 J = 1,NEWIDE
              ID(1) = I*NPWIDE + J
              ID(2) = ID(1) + 1
              ID(3) = ID(2) - NPWIDE
              ID(4) = ID(1) - NPWIDE
              CALL FMSSWR (LUS(2), IREC, 6)
   10      CONTINUE
   20   CONTINUE
C
C       Generate element data array S(4,4):
        DO 40 I = 1,4
           DO 30 J = 1,4
              S(I,J) = -1.0D0
   30      CONTINUE
           S(I,I) = 3.0D0
   40   CONTINUE
C
C       Write element real records:
        CALL FMSEEK (LUS(1), 0)
        DO 50 NE = 1,NUMEL
           CALL FMSSWR (LUS(1), S, 16)
   50   CONTINUE
C
C       Generate right-hand side vector B(NUMEQ)
C       Generate boundary condition code vector NEWNUM(NUMEQ)
        DO 60 I = 1,NUMEQ
           B(I) = 0.0D0
           NEWNUM(I) = 1
```

```
   60   CONTINUE
C       Specify X(1:NPWIDE) = 0, X(NUMEQ-NPWIDE+1:NUMEQ) = 1
        I2 = NUMEQ - NPWIDE
        DO 70 I1 = 1,NPWIDE
           NEWNUM(I1) = 0
           B(I1)   = 0.0D0
           I2      = I2 + 1
           NEWNUM(I2) = -1
           B(I2)   = 1.0D0
   70   CONTINUE
        CALL FMSPOP (MYNAME)
        RETURN
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 6. WAVEFRONT INTERFACE

This example illustrates how ***FMS*** is used in finite element programs which use a wavefront solution algorithm. First you call FMSOS once to create a single submatrix file on which you store your element data in the wavefront order. The order that you write your records to this file will determine the wavefront numbering scheme. Then you call subroutine FMSWF to generate global equation numbers and update the submatrix integer file. Subroutine FMSWF also computes the profile vector LOWEQ and the NEWNUM vector which relates the equation labels you wrote to the submatrix integer file to the new global equation numbers. This example also illustrates how you can write a user supplied subroutine to modify the equations after assembly before the factoring process begins. In this example, RSUBLK is used to modify diagonal terms to impose specified solution values.

```
C       E X A M P L E   6
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_6')
C
C       Problem size parameters:
C
C       Number of elements wide and long:
        PARAMETER (NEWIDE = 20, NELONG = 100)
C
C       Number of nodes wide and long:
        PARAMETER (NPWIDE = NEWIDE+1, NPLONG = NELONG+1)
C
C       Number of elements and equations:
        PARAMETER (NUMEL  = NEWIDE*NELONG, MAXEQ = NPWIDE*NPLONG)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
```

```
C       Length of element real record:
        PARAMETER (LENR = 4*4)
C
C       Length of element integer record:
        PARAMETER (LENI = 2 + 4)
C
C       Length of element vector record:
        PARAMETER (LENV = 2)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
C       (Performed during call to RSDF)
        PARAMETER (ISKIP  = 1)
C
C       FMS matrix and vector file attributes:
        INTEGER   LUA(25)
        INTEGER   LUX(25), LUAX(25)
        INTEGER   LUS(25)
C
C       Solution vector {X}
C       RHS vector      {B}
C       Product [A]{X}, {AX}
        REAL*8    RDUM(1), X(MAXEQ), B(MAXEQ), AX(MAXEQ)
        COMMON    /DATA/ RDUM, X, B, AX
C
C       Big diagonal value to impose constraints:
        REAL*8     BIGNUM
C
C       Local variables:
        REAL*8    EI, ERROR
C
C       Common block to communicate with RSUBLK:
        COMMON /EXAMP6/ BIGNUM, IBC
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
```

```
      POINTER (IMD_PTR, IMD)
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
      DATA LUA0(1)/0/
C
C (1)   Initialize FMS:
      CALL FMSINI
      CALL FMSPSH(MYNAME)
      CALL FMSIST ('MPOSDF',1)
      CALL FMSIST ('INCORE',2)
      CALL FMSIST ('MDATAU',2)
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
      WRITE(6,*) 'NUMBER OF ELEMENTS          = ', NUMEL
      WRITE(6,*) 'INITIAL NUMBER OF EQUATIONS = ', MAXEQ
      BIGNUM = 1.0D30
C
C (2)   Open FMS files:
      CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C       Generate element data:
      CALL FMSIMG (IMD, NEWNUM, MAXEQ)
      CALL ELGEN (NEWIDE,NELONG,NUMEL,IMD(NEWNUM),LUS,B,MAXEQ)
C       Compute profile vector; eliminate zero constraints:
      CALL FMSIMG (IMD, LOWEQ, MAXEQ)
      CALL FMSWF (LUS, IMD(NEWNUM), MAXEQ, IMD(LOWEQ), NUMEQ)
      WRITE(6,*) 'ACTUAL  NUMBER OF EQUATIONS = ', NUMEQ
      CALL RSDI   (IMD(LOWEQ), NUMEQ, 'LUA', LUA)
      CALL FMSIMR (IMD, LOWEQ, MAXEQ)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, X, NUMEQ, LUX)
C
C (3)   Write data to FMS files:
C       Elements written during ELGEN.
C       Assembly during factoring.
C       Impose boundary conditions on RHS vector:
C       Flag IBC array for constrained diagonals:
C       Reorder RHS terms for new numbering system:
      CALL FMSIMG (IMD, IBC, NUMEQ)
```

```
         DO 31 I = 1,MAXEQ
            INEW   = IMD(NEWNUM-1+I)
            IF(INEW .EQ. 0) GO TO 31
            IF(INEW .GT. 0) THEN
               IMD(IBC-1+INEW) = 0
               X(INEW) = B(I)
            ELSE
               INEW   = -INEW
               IMD(IBC-1+INEW) = 1
               X(INEW) = BIGNUM*B(I)
            END IF
   31    CONTINUE
C
C (4)    Perform matrix algebra:
         CALL RSDAF (LUA0, RDUM, 0, LUS, NUMSF, LUA0, LUA,
     1              LUX, LUX, NUMRHS)
         CALL FMSIMR (IMD, IBC, NUMEQ)
         CALL RSDS (LUA, LUX, LUX, NUMRHS, ISKIP)
C
C (5)    Read data from FMS files:
C        Not required.
C
C        Compute [A]{X} from SUM([Si]{X}) to check error:
         CALL FMSRMG (RMD, LAX, NUMEQ)
         CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, RMD(LAX), NUMEQ, LUAX)
         CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
         ERROR = 0.0D0
C        Expand solution vector to include zero constrained values:
         DO 50 I = 1,MAXEQ
            INEW   = IMD(NEWNUM-1+I)
            IF(INEW .EQ. 0) THEN
C              Constrained zero value:
               B(I) = 0.0D0
            ELSE IF(INEW .LT. 0) THEN
C              Constrained nonzero value:
               INEW = -INEW
               B(I) = X(INEW)
            ELSE
C              Value which was determined:
```

```
              EI = ABS(RMD(LAX-1+INEW) - B(I))
              IF(EI .GT. ERROR) ERROR = EI
              B(I) = X(INEW)
           END IF
   50   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =',ERROR
        CALL FMSIMR (IMD, NEWNUM, MAXEQ)
C
C (6)   Close FMS files:
        CALL FMSCV (LUAX)
        CALL FMSRMR (RMD, LAX, NUMEQ)
        CALL FMSCM (LUA)
        CALL FMSCV (LUX)
        CALL FMSCS (LUS)
        CALL FMSPOP(MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER LOWEQ(*), LOCEQ(*)
        REAL*8  A(0:*), D(*), BIGNUM
        COMMON /EXAMP6/ BIGNUM, IBC
C
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
```

```
            DO 10 IROW = IROW1,IROW2
               IF(IMD(IBC-1+IROW) .NE. 0) D(IROW) = BIGNUM
   10       CONTINUE
         END IF
         RETURN
         END
C=======================================================================
         SUBROUTINE ELGEN (NEWIDE,NELONG,NUMEL,NEWNUM,LUS,B,NUMEQ)
C=======================================================================
         INTEGER NEWIDE, NELONG, NUMEL, NUMEQ
         INTEGER LUS(25), NEWNUM(NUMEQ)
         INTEGER IREC(6), NSUB, ITYPE, ID(4)
         CHARACTER*5 MYNAME
         PARAMETER (MYNAME='ELGEN')
         REAL*8  S(4,4), B(NUMEQ)
         EQUIVALENCE (IREC(1),NSUB), (IREC(2),ITYPE), (IREC(3),ID)
C
         CALL FMSPSH(MYNAME)
C
C        Generate a 4-node quadrilateral finite element mesh that
C        is NEWIDE elements wide by NELONG elements long.  Use the
C        same matrix S(4,4) for all elements. Impose the following
C        solution values: X=0 along the top, X=1 along the bottom.
C
C        Generate element connectivity array ID(4):
C        Write element integer records:
         CALL FMSEEK (LUS(2), 0)
         NSUB   = 4
         ITYPE  = 1
         NPWIDE = NEWIDE + 1
         DO 20 I = 1,NELONG
            DO 10 J = 1,NEWIDE
               ID(1) = I*NPWIDE + J
               ID(2) = ID(1) + 1
               ID(3) = ID(2) - NPWIDE
               ID(4) = ID(1) - NPWIDE
               CALL FMSSWR (LUS(2), IREC, 6)
   10       CONTINUE
   20    CONTINUE
```

```
C
C       Generate element data array S(4,4):
        DO 40 I = 1,4
           DO 30 J = 1,4
              S(I,J) = -1.0D0
   30      CONTINUE
           S(I,I) = 3.0D0
   40   CONTINUE
C
C       Write element real records:
        CALL FMSEEK (LUS(1), 0)
        DO 50 NE = 1,NUMEL
           CALL FMSSWR (LUS(1), S, 16)
   50   CONTINUE
C
C       Generate right-hand side vector B(NUMEQ)
C       Generate boundary condition code vector NEWNUM(NUMEQ)
        DO 60 I = 1,NUMEQ
           B(I) = 0.0D0
           NEWNUM(I) = 1
   60   CONTINUE
C       Specify X(1:NPWIDE) = 0, X(NUMEQ-NPWIDE+1:NUMEQ) = 1
        I2 = NUMEQ - NPWIDE
        DO 70 I1 = 1,NPWIDE
           NEWNUM(I1) = 0
           B(I1)   = 0.0D0
           I2      = I2 + 1
           NEWNUM(I2) = -1
           B(I2)   = 1.0D0
   70   CONTINUE
        CALL FMSPOP(MYNAME)
        RETURN
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   6
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_6')
C
C       Problem size parameters:
C
C       Number of elements wide and long:
        PARAMETER (NEWIDE = 20, NELONG = 100)
C
C       Number of nodes wide and long:
        PARAMETER (NPWIDE = NEWIDE+1, NPLONG = NELONG+1)
C
C       Number of elements and equations:
        PARAMETER (NUMEL  = NEWIDE*NELONG, MAXEQ = NPWIDE*NPLONG)
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
C       Length of element real record:
        PARAMETER (LENR = 4*4)
C
C       Length of element integer record:
        PARAMETER (LENI = 2 + 4)
C
C       Length of element vector record:
        PARAMETER (LENV = 2)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
C       (Performed during call to RSDF)
        PARAMETER (ISKIP  = 1)
C
C       FMS matrix and vector file attributes:
        INTEGER   LUA(25), LUA0(25)
        INTEGER   LUX(25), LUAX(25)
        INTEGER   LUS(25)
C
C       Solution vector {X}
C       RHS vector      {B}
C       Product [A]{X}, {AX}
        REAL*8    RDUM(1), X(MAXEQ), B(MAXEQ), AX(MAXEQ)
        COMMON    /DATA/ RDUM, X, B, AX
C
C       Big diagonal value to impose constraints:
        REAL*8     BIGNUM
C
C       Local variables:
        REAL*8    EI, ERROR
C
C       Common block to communicate with RSUBLK:
        COMMON /EXAMP6/ BIGNUM, IBC
C
C       FMS memory management requires the following arrays:
```

```
      POINTER (CMD_PTR, CMD)
      POINTER (RMD_PTR, RMD)
      POINTER (IMD_PTR, IMD)
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
      DATA LUA0(1)/0/
C
C (1)   Initialize FMS:
      CALL FMSINI
      CALL FMSPSH (MYNAME)
      CALL FMSIST ('MPOSDF',1)
      CALL FMSIST ('INCORE',2)
      CALL FMSIST ('MDATAU',2)
      CALL FMSIGT ('MEMPTR',IMD_PTR)
      CALL FMSIGT ('MEMPTR',RMD_PTR)
      CALL FMSIGT ('MEMPTR',CMD_PTR)
      WRITE(6,*) 'NUMBER OF ELEMENTS          = ', NUMEL
      WRITE(6,*) 'INITIAL NUMBER OF EQUATIONS = ', MAXEQ
      BIGNUM = 1.0D30
C
C (2)   Open FMS files:
      CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C       Generate element data:
      CALL FMSIMG (IMD, NEWNUM, MAXEQ)
      CALL ELGEN (NEWIDE,NELONG,NUMEL,IMD(NEWNUM),LUS,B,MAXEQ)
C       Compute profile vector; eliminate zero constraints:
      CALL FMSIMG (IMD, LOWEQ, MAXEQ)
      CALL FMSWF (LUS, IMD(NEWNUM), MAXEQ, IMD(LOWEQ), NUMEQ)
      WRITE(6,*) 'ACTUAL  NUMBER OF EQUATIONS = ', NUMEQ
      CALL RSDI   (IMD(LOWEQ), NUMEQ, 'LUA', LUA)
      CALL FMSIMR (IMD, LOWEQ, MAXEQ)
      CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, X, NUMEQ, LUX)
C
C (3)   Write data to FMS files:
C       Elements written during ELGEN.
C       Assembly during factoring.
C       Impose boundary conditions on RHS vector:
C       Flag IBC array for constrained diagonals:
C       Reorder RHS terms for new numbering system:
      CALL FMSIMG (IMD, IBC, NUMEQ)
      DO 31 I = 1,MAXEQ
         INEW   = IMD(NEWNUM-1+I)
         IF(INEW .EQ. 0) GO TO 31
         IF(INEW .GT. 0) THEN
            IMD(IBC-1+INEW) = 0
            X(INEW) = B(I)
         ELSE
            INEW   = -INEW
            IMD(IBC-1+INEW) = 1
            X(INEW) = BIGNUM*B(I)
         END IF
   31   CONTINUE
C
C (4)   Perform matrix algebra:
      CALL RSDAF (LUA0, RDUM, 0, LUS, NUMSF, LUA0, LUA,
     1              LUX, LUX, NUMRHS)
      CALL FMSIMR (IMD, IBC, NUMEQ)
      CALL RSDS (LUA, LUX, LUX, NUMRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Not required.
C
C       Compute [A]{X} from SUM([Si]{X}) to check error:
      CALL FMSRMG (RMD, LAX, NUMEQ)
```

```
         CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, RMD(LAX), NUMEQ, LUAX)
         CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
         ERROR = 0.0D0
C        Expand solution vector to include zero constrained values:
         DO 50 I = 1,MAXEQ
            INEW   = IMD(NEWNUM-1+I)
            IF(INEW .EQ. 0) THEN
C              Constrained zero value:
               B(I) = 0.0D0
            ELSE IF(INEW .LT. 0) THEN
C              Constrained nonzero value:
               INEW = -INEW
               B(I) = X(INEW)
            ELSE
C              Value which was determined:
               EI = ABS(RMD(LAX-1+INEW) - B(I))
               IF(EI .GT. ERROR) ERROR = EI
               B(I) = X(INEW)
            END IF
   50    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =',ERROR
         CALL FMSIMR (IMD, NEWNUM, MAXEQ)
C
C (6)    Close FMS files:
         CALL FMSCV (LUAX)
         CALL FMSRMR (RMD, LAX, NUMEQ)
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         CALL FMSCS (LUS)
         CALL FMSPOP (MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER LOWEQ(*), LOCEQ(*)
         REAL*8  A(0:*), D(*), BIGNUM
         COMMON /EXAMP6/ BIGNUM, IBC
C
         POINTER (CMD_PTR, CMD)
         POINTER (RMD_PTR, RMD)
         POINTER (IMD_PTR, IMD)
         COMPLEX*16 CMD(0:1)
         REAL*8     RMD(0:1)
         INTEGER    IMD(0:1)
C
         CALL FMSIGT ('MEMPTR',IMD_PTR)
         CALL FMSIGT ('MEMPTR',RMD_PTR)
         CALL FMSIGT ('MEMPTR',CMD_PTR)
         IF(IROW2 .EQ. JCOL2) THEN
C           This is a diagonal block:
            DO 10 IROW = IROW1,IROW2
               IF(IMD(IBC-1+IROW) .NE. 0) D(IROW) = BIGNUM
   10       CONTINUE
         END IF
         RETURN
         END
C=======================================================================
         SUBROUTINE ELGEN (NEWIDE,NELONG,NUMEL,NEWNUM,LUS,B,NUMEQ)
C=======================================================================
         INTEGER NEWIDE, NELONG, NUMEL, NUMEQ
         INTEGER LUS(25), NEWNUM(NUMEQ)
         INTEGER IREC(6), NSUB, ITYPE, ID(4)
```

```
      CHARACTER*5 MYNAME
      PARAMETER (MYNAME='ELGEN')
      REAL*8  S(4,4), B(NUMEQ)
      EQUIVALENCE (IREC(1),NSUB), (IREC(2),ITYPE), (IREC(3),ID)
C
      CALL FMSPSH (MYNAME)
C
C     Generate a 4-node quadrilateral finite element mesh that
C     is NEWIDE elements wide by NELONG elements long.  Use the
C     same matrix S(4,4) for all elements. Impose the following
C     solution values: X=0 along the top, X=1 along the bottom.
C
C     Generate element connectivity array ID(4):
C     Write element integer records:
      CALL FMSEEK (LUS(2), 0)
      NSUB   = 4
      ITYPE  = 1
      NPWIDE = NEWIDE + 1
      DO 20 I = 1,NELONG
         DO 10 J = 1,NEWIDE
            ID(1) = I*NPWIDE + J
            ID(2) = ID(1) + 1
            ID(3) = ID(2) - NPWIDE
            ID(4) = ID(1) - NPWIDE
            CALL FMSSWR (LUS(2), IREC, 6)
   10    CONTINUE
   20 CONTINUE
C
C     Generate element data array S(4,4):
      DO 40 I = 1,4
         DO 30 J = 1,4
            S(I,J) = -1.0D0
   30    CONTINUE
         S(I,I) = 3.0D0
   40 CONTINUE
C
C     Write element real records:
      CALL FMSEEK (LUS(1), 0)
      DO 50 NE = 1,NUMEL
         CALL FMSSWR (LUS(1), S, 16)
   50 CONTINUE
C
C     Generate right-hand side vector B(NUMEQ)
C     Generate boundary condition code vector NEWNUM(NUMEQ)
      DO 60 I = 1,NUMEQ
         B(I) = 0.0D0
         NEWNUM(I) = 1
   60 CONTINUE
C     Specify X(1:NPWIDE) = 0, X(NUMEQ-NPWIDE+1:NUMEQ) = 1
      I2 = NUMEQ - NPWIDE
      DO 70 I1 = 1,NPWIDE
         NEWNUM(I1) = 0
         B(I1)   = 0.0D0
         I2      = I2 + 1
         NEWNUM(I2) = -1
         B(I2)   = 1.0D0
   70 CONTINUE
      CALL FMSPOP (MYNAME)
      RETURN
      END
```

Copyright © Multipath Corporation

FMS EXAMPLE 7. SUBSTRUCTURING

This example extends Example 3 to illustrate the substructuring capability within ***FMS***. First a complex nonsymmetric matrix is formed and factored down to the substructure equation NEQSUB. The substructure matrix **[ASUB]** and substructure vector ***{XSUB}*** are then extracted to form a new reduced system **[ASUB]{X2} = {BSUB}** .

This system is then solved for **{X2}**.

The values of **{X2}** are then inserted into **{X}** and the back substitution on **{X}** is completed.

```
C       E X A M P L E   7
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_7')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of vectors to reduce during factoring:
        INTEGER NUMRED
C
C       Skip operations during solving (no):
        INTEGER ISKIP
C
C       FMS matrix and vector file attributes:
        INTEGER     LUA(25), LUASUB(25)
        INTEGER     LUX(25), LUXSUB(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
```

```
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C
C     Profile vector for a full matrix;
      INTEGER    LOWEQ(1)
C
C     Local variables:
      INTEGER     N, NRHS, I, LDISK, NV
      INTEGER     L_X, L_XSUB, L_ASUB, LENX
      REAL*8      EI, ERROR
C
C     Common block to communicate with CSUBLK:
      COMMON /MYDATA/N, NEQSUB, NRHS
      DATA LOWEQ/-1/
C
C (1) Initialize FMS:
      CALL FMSINI
      CALL FMSPSH(MYNAME)
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
      N      = ASK_I ('Enter the number of equations')
      NEQSUB = ASK_I ('Enter the substructure equation NEQSUB')
      NRHS   = ASK_I ('Enter the number of solution vectors')
C
C (2) Open FMS files:
      CALL FMSIST ('NEQSUB', NEQSUB)
      CALL CNDI (LOWEQ, N, 'LUA', LUA)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C     Populate test vector:
      LENX = LUX(4)/2
      CALL FMSCMG (CMD, L_X, LENX)
      DO I = 1,LENX
         CMD(L_X-1+I) = (0.0D0,0.0D0)
      END DO
      CMD(L_X) = (1.0D0,1.0D0)
C
```

```
C (3)   Write data to FMS files:
        LDISK  = 1
        DO NV = 1,NRHS
           CALL FMSWRT (LUX(1), LDISK, CMD(L_X), LUX(4))
           LDISK = LDISK + LUX(4)
        END DO
C
C (4)   Perform matrix algebra:
C
C       Factor the matrix down to the substructure:
        CALL FMSIST ('MDATAU',  2)
        NUMRED = 0
        CALL CNDF (LUA, LUA, LUX, LUX,  NUMRED)
C
C       Reduce the RHS vectors down to the substructure.
        ISKIP = 2
        CALL CNDS (LUA, LUX, LUX, NRHS, ISKIP)
C
C       Complete substructure solution:
        NSUB   = N - NEQSUB + 1
        IF(NSUB .GT. 0) THEN
C          Substructuring selected:
C          Turn off further substructuring:
           CALL FMSIST ('NEQSUB', N+1)
C          Turn off calling user subroutine:
           CALL FMSIST ('MDATAU',  0)
C
C          Allocate memory to store the submatrix:
           CALL FMSCMG (CMD, L_ASUB, NSUB*NSUB)
C
C          Extract the substructure matrix:
           CALL CNDEX (LUA, CMD(L_ASUB), NSUB, NSUB)
C
C          Form an incore FMS matrix for [ASUB]:
           CALL CNDANN (CMD(L_ASUB), NSUB, NSUB, LUASUB)
C
C          Allocate memory to store the submatrix vector:
           CALL FMSCMG (CMD, L_XSUB, NSUB*NRHS)
C
```

```
C          Read the reduced vectors {BSUB}:
           LBUF  = L_X    - 1 + (NEQSUB-1)
           LX    = L_XSUB - 1
           LDISK = 1
           DO IRHS = 1,NRHS
              CALL FMSRED (LUX(1), LDISK, CMD(L_X), LUX(4))
              DO I=1,NSUB
                 CMD(LX+I) = CMD(LBUF+I)
              END DO
              LDISK = LDISK + LUX(4)
              LX    = LX    + NSUB
           END DO
C
C          Form an incore FMS vector for {XSUB}:
           CALL FMSOV2 (NSUB, IDTYPE, NRHS, CMD(L_XSUB), NSUB, LUXSUB)
C
C          Factor the submatrix:
           NUMRED = 0
           CALL CNDF (LUASUB, LUASUB, LUXSUB, LUXSUB, NUMRED)
C
C          Perform forward reduction and back substitution on {XSUB}:
           ISKIP = 0
           CALL CNDS (LUASUB, LUXSUB, LUXSUB, NRHS, ISKIP)
C
C          Close substructure matrix and return storage:
           CALL FMSCM  (LUASUB)
           CALL FMSCMR (CMD, L_ASUB, NSUB*NSUB)
C
C          Place the solution {XSUB} back in {X}:
C
C          Read the reduced vectors {BSUB}:
           LBUF  = L_X    - 1 + (NEQSUB-1)
           LX    = L_XSUB - 1
           LDISK = 1
           DO IRHS = 1,NRHS
              CALL FMSRED (LUX(1), LDISK, CMD(L_X), LUX(4))
              DO I=1,NSUB
                 CMD(LBUF+I) = CMD(LX+I)
              END DO
```

```
               CALL FMSWRT (LUX(1), LDISK, CMD(L_X), LUX(4))
               LDISK = LDISK + LUX(4)
               LX    = LX    + NSUB
            END DO
C
C           Close substructure vector file and return storage:
            CALL FMSCV  (LUXSUB)
            CALL FMSCMR (CMD, L_XSUB, NSUB*NRHS)
C
C           Restore the value of NEQSUB:
            CALL FMSIST ('NEQSUB', NEQSUB)
         END IF
C
C        Complete the back-substitution for {X1}:
         ISKIP = 1
         CALL CNDS (LUA, LUX, LUX, NRHS, ISKIP)
C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
         LDINC  = LUX(4)
         LDISK  = 1 - LDINC
         DO 60 NV = 1,NRHS
            LDISK  = LDISK  + LDINC
            CALL FMSRED (LUX(1), LDISK, CMD(L_X), LUX(4))
            DO 50 I = 1,N
               EI = ABS(CMD(L_X-1+I) - 1.0D0)
               IF(EI .GT. ERROR) ERROR = EI
   50       CONTINUE
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         CALL FMSCMR (CMD, L_X, LENX)
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
```

```
C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), ONE
        PARAMETER  (ONE=(1.0D0,1.0D0))
        COMMON /MYDATA/N, NEQSUB, NRHS
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = ONE
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = -ONE
   20   CONTINUE
        RETURN
        END
C=======================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), ONE
        PARAMETER  (ONE=(1.0D0,1.0D0))
C
C       Populate profile of [AU] with test data:
```

```
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 10 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = -ONE
   10   CONTINUE
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   7
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_7')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of vectors to reduce during factoring:
        INTEGER NUMRED
C
C       Skip operations during solving (no):
        INTEGER ISKIP
C
C       FMS matrix and vector file attributes:
        INTEGER     LUA(25), LUASUB(25)
        INTEGER     LUX(25), LUXSUB(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Profile vector for a full matrix;
        INTEGER    LOWEQ(1)
C
C       Local variables:
        INTEGER     N, NRHS, I, LDISK, NV
        INTEGER     L_X, L_XSUB, L_ASUB, LENX
        REAL*8      EI, ERROR
C
C       Common block to communicate with CSUBLK:
        COMMON /MYDATA/N, NEQSUB, NRHS
        DATA LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        N      = ASK_I ('Enter the number of equations')
        NEQSUB = ASK_I ('Enter the substructure equation NEQSUB')
        NRHS   = ASK_I ('Enter the number of solution vectors')
C
C (2)   Open FMS files:
        CALL FMSIST ('NEQSUB', NEQSUB)
        CALL CNDI (LOWEQ, N, 'LUA', LUA)
        CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C       Populate test vector:
        LENX = LUX(4)/2
        CALL FMSCMG (CMD, L_X, LENX)
        DO I = 1,LENX
           CMD(L_X-1+I) = (0.0D0,0.0D0)
        END DO
```

```
      CMD(L_X) = (1.0D0,1.0D0)
C
C (3)   Write data to FMS files:
      LDISK  = 1
      DO NV = 1,NRHS
         CALL FMSWRT (LUX(1), LDISK, CMD(L_X), LUX(4))
         LDISK = LDISK + LUX(4)
      END DO
C
C (4)   Perform matrix algebra:
C
C       Factor the matrix down to the substructure:
      CALL FMSIST ('MDATAU',  2)
      NUMRED = 0
      CALL CNDF (LUA, LUA, LUX, LUX,  NUMRED)
C
C       Reduce the RHS vectors down to the substructure.
      ISKIP = 2
      CALL CNDS (LUA, LUX, LUX, NRHS, ISKIP)
C
C       Complete substructure solution:
      NSUB   = N - NEQSUB + 1
      IF(NSUB .GT. 0) THEN
C          Substructuring selected:
C          Turn off further substructuring:
           CALL FMSIST ('NEQSUB', N+1)
C          Turn off calling user subroutine:
           CALL FMSIST ('MDATAU',  0)
C
C          Allocate memory to store the submatrix:
           CALL FMSCMG (CMD, L_ASUB, NSUB*NSUB)
C
C          Extract the substructure matrix:
           CALL CNDEX (LUA, CMD(L_ASUB), NSUB, NSUB)
C
C          Form an incore FMS matrix for [ASUB]:
           CALL CNDANN (CMD(L_ASUB), NSUB, NSUB, LUASUB)
C
C          Allocate memory to store the submatrix vector:
           CALL FMSCMG (CMD, L_XSUB, NSUB*NRHS)
C
C          Read the reduced vectors {BSUB}:
           LBUF  = L_X    - 1 + (NEQSUB-1)
           LX    = L_XSUB - 1
           LDISK = 1
           DO IRHS = 1,NRHS
              CALL FMSRED (LUX(1), LDISK, CMD(L_X), LUX(4))
              DO I=1,NSUB
                 CMD(LX+I) = CMD(LBUF+I)
              END DO
              LDISK = LDISK + LUX(4)
              LX    = LX    + NSUB
           END DO
C
C          Form an incore FMS vector for {XSUB}:
           CALL FMSOV2 (NSUB, IDTYPE, NRHS, CMD(L_XSUB), NSUB, LUXSUB)
C
C          Factor the submatrix:
           NUMRED = 0
           CALL CNDF (LUASUB, LUASUB, LUXSUB, LUXSUB, NUMRED)
C
C          Perform forward reduction and back substitution on {XSUB}:
           ISKIP = 0
           CALL CNDS (LUASUB, LUXSUB, LUXSUB, NRHS, ISKIP)
C
```

```
C          Close substructure matrix and return storage:
           CALL FMSCM  (LUASUB)
           CALL FMSCMR (CMD, L_ASUB, NSUB*NSUB)
C
C          Place the solution {XSUB} back in {X}:
C
C          Read the reduced vectors {BSUB}:
           LBUF  = L_X    - 1 + (NEQSUB-1)
           LX    = L_XSUB - 1
           LDISK = 1
           DO IRHS = 1,NRHS
              CALL FMSRED (LUX(1), LDISK, CMD(L_X), LUX(4))
              DO I=1,NSUB
                 CMD(LBUF+I) = CMD(LX+I)
              END DO
              CALL FMSWRT (LUX(1), LDISK, CMD(L_X), LUX(4))
              LDISK = LDISK + LUX(4)
              LX    = LX    + NSUB
           END DO
C
C          Close substructure vector file and return storage:
           CALL FMSCV  (LUXSUB)
           CALL FMSCMR (CMD, L_XSUB, NSUB*NRHS)
C
C          Restore the value of NEQSUB:
           CALL FMSIST ('NEQSUB', NEQSUB)
        END IF
C
C       Complete the back-substitution for {X1}:
        ISKIP = 1
        CALL CNDS (LUA, LUX, LUX, NRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Check the answer:
        ERROR = 0.0D0
        LDINC  = LUX(4)
        LDISK  = 1 - LDINC
        DO 60 NV = 1,NRHS
           LDISK  = LDISK  + LDINC
           CALL FMSRED (LUX(1), LDISK, CMD(L_X), LUX(4))
           DO 50 I = 1,N
              EI = ABS(CMD(L_X-1+I) - 1.0D0)
              IF(EI .GT. ERROR) ERROR = EI
   50      CONTINUE
   60   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUX)
        CALL FMSCMR (CMD, L_X, LENX)
        CALL FMSPOP (MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), ONE
        PARAMETER  (ONE=(1.0D0,1.0D0))
        COMMON /MYDATA/N, NEQSUB, NRHS
C
C       Populate the diagonal with test data:
```

```
         IF(IROW2 .EQ. JCOL2) THEN
C           This is a diagonal block:
            DO 10 I = IROW1,IROW2
               D(I) = ONE
   10       CONTINUE
            IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
         END IF
C
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
         DO 20 I = IROW1,IROW2
            J = LOWEQ(I)
            IF( (J .GE. JCOL1) .AND.
     1          (J .LE. JCOL2) .AND.
     2          (J .LT.      I) ) A(LOCEQ(I) + IJSTEP*J) = -ONE
   20    CONTINUE
         RETURN
         END
C=======================================================================
         SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         COMPLEX*16 A(0:*), D(*), ONE
         PARAMETER  (ONE=(1.0D0,1.0D0))
C
C        Populate profile of [AU] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
         DO 10 J = JCOL1,JCOL2
            I = LOWEQ(J)
            IF( (I .GE. IROW1) .AND.
     1          (I .LE. IROW2) .AND.
     2          (I .LT.      J) ) A(LOCEQ(J) + IJSTEP*I) = -ONE
   10    CONTINUE
         RETURN
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
         CHARACTER* (*) STRING
         WRITE(6,2000) STRING
         READ (5,*) ASK_I
         RETURN
 2000    FORMAT (1X,A,'>')
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 8. USING FMSROW

```
C       E X A M P L E     F M S R O W
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_8')
C
        PARAMETER   (NMAX = 1000)
        COMMON      A(NMAX), X(NMAX)
        COMPLEX*16  A, X, ZERO, ONE, ALPHA(1)
        REAL*8      EI, ERROR
        INTEGER     LUA(25), LUX(25), LUS0(25), LUA0(25)
C
C       Profile vector for a full matrix:
        INTEGER     LOWEQ(1)
        DATA LOWEQ/-1/
        DATA ZERO /(0.D0,0.D0)/
        DATA ONE  /(1.D0,1.D0)/
        DATA ALPHA/(1.D0,0.D0)/
        DATA LUA0(1)/0/
        DATA LUS0(1)/0/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
        N = NMAX
C
C (2)   Open FMS files:
        CALL CNDI (LOWEQ, N, 'LUA', LUA)
        CALL FMSOV2 (N, 2, 1, X, N, LUX)
        DO 21 I = 1,N
           X(I) = ZERO
   21   CONTINUE
        X(1) =  ONE
C
```

```
C (3)   Write data to FMS files:
C       Initialize FMSROW
        CALL FMSROW (0, A, LUA)
C       Write first row
        DO 31 I = 2,N
           A(I) = -ONE
   31   CONTINUE
        A(1) = DCMPLX(N,N)
        CALL FMSROW (1, A, LUA)
C       Write remaining rows
        DO 32 I = 2,N
           A(I) = ZERO
   32   CONTINUE
        A(1) = -ONE
        DO 33 I = N,2,-1
           A(I) = ONE
           CALL FMSROW (I, A, LUA)
           A(I) = ZERO
   33   CONTINUE
C       End FMSROW
        CALL FMSROW (N+1, A, LUA)
C
C (4)   Perform matrix algebra:
        CALL CNDAF (LUA, ALPHA, 1, LUS0, 0, LUA0,
     1   LUA, LUX, LUX, 0)
        CALL CNDS  (LUA, LUX, LUX, 1, 0)
C
C (5))  Read data from FMS files:
C       (Not required).
C
C       Check the answer:
        ERROR = 0.0D0
        DO 50 I = 1,N
           EI = ABS(X(I) - 1.0D0)
           IF(EI .GT. ERROR) ERROR = EI
   50   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C       Close FMS files:
```

```
      CALL FMSCM (LUA)
      CALL FMSCV (LUX)
      CALL FMSPOP(MYNAME)
      CALL FMSEND
      END
```

Copyright © Multipath Corporation

```
C       E X A M P L E     F M S R O W
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_8')
C
        PARAMETER   (NMAX = 1000)
        COMMON      A(NMAX), X(NMAX)
        COMPLEX*16  A, X, ZERO, ONE, ALPHA(1)
        REAL*8      EI, ERROR
        INTEGER     LUA(25), LUX(25), LUS0(25), LUA0(25)
C
C       Profile vector for a full matrix:
        INTEGER     LOWEQ(1)
        DATA LOWEQ/-1/
        DATA ZERO /(0.D0,0.D0)/
        DATA ONE  /(1.D0,1.D0)/
        DATA ALPHA/(1.D0,0.D0)/
        DATA LUA0(1)/0/
        DATA LUS0(1)/0/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        N = NMAX
C
C (2)   Open FMS files:
        CALL CNDI (LOWEQ, N, 'LUA', LUA)
        CALL FMSOV2 (N, 2, 1, X, N, LUX)
        DO 21 I = 1,N
           X(I) = ZERO
   21   CONTINUE
        X(1) =  ONE
C
C (3)   Write data to FMS files:
C       Initialize FMSROW
        CALL FMSROW (0, A, LUA)
C       Write first row
        DO 31 I = 2,N
           A(I) = -ONE
   31   CONTINUE
        A(1) = DCMPLX(N,N)
        CALL FMSROW (1, A, LUA)
C       Write remaining rows
        DO 32 I = 2,N
           A(I) = ZERO
   32   CONTINUE
        A(1) = -ONE
        DO 33 I = N,2,-1
           A(I) = ONE
           CALL FMSROW (I, A, LUA)
           A(I) = ZERO
   33   CONTINUE
C       End FMSROW
        CALL FMSROW (N+1, A, LUA)
C
C (4)   Perform matrix algebra:
        CALL CNDAF (LUA, ALPHA, 1, LUS0, 0, LUA0,
     1   LUA, LUX, LUX, 0)
        CALL CNDS  (LUA, LUX, LUX, 1, 0)
C
C (5))  Read data from FMS files:
C       (Not required).
```

```
C
C       Check the answer:
        ERROR = 0.0D0
        DO 50 I = 1,N
           EI = ABS(X(I) - 1.0D0)
           IF(EI .GT. ERROR) ERROR = EI
   50   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C       Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUX)
        CALL FMSPOP (MYNAME)
        CALL FMSEND
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 9. RADAR CROSS SECTION SAMPLE PROGRAM

This example illustrates how to use ***FMS*** in a typical Radar Cross Section program. The example includes using the ***FMS*** parallel processing utilities to generate the matrix coefficients in parallel.

```
        PROGRAM RCSFMS
C-----------------------------------------------------------------------
C       This is a sample program that illustrates the following:
C
C       * How to use the FMS out-of-core solver in a RCS program,
C
C       * How to generate matrix elements efficiently,
C
C       * How to use the FMS parallel processing utilities to generate
C         matrix elements in parallel.
C
C       It is assumed that the model consists of a mesh composed of
C       triangles, and the unknowns (equations) are the values on the
C       triangle edges.  It is also assumed that each triangle impacts
C       every other triangle, producing a full matrix.
C
C       It is also assumed that the data is double precision complex,
C       (COMPLEX*16).
C
C       The test matrix produced by this example has a solution vector
C       of all (0.5,-0.5)'s.
C
C-----------------------------------------------------------------------
C       DECLARATIONS
C-----------------------------------------------------------------------
C
C       Program name:
        CHARACTER*9 MYNAME
        PARAMETER (MYNAME='EXAMPLE_9')
C
```

```
C       Input data functions:
        INTEGER ASK_I
C
C       Subroutines called in parallel (passed to FMSPAR) must be
C       declared external.
        EXTERNAL FILL
C
C       FMS file object handles (attribute lists)
C       Matrix file:
        INTEGER LUA(25)
C       Vector file:
        INTEGER LUX(25)
C
C       Data type:
        INTEGER IDTYPE
C       Complex*16:
        PARAMETER (IDTYPE=2)
C
C       FMS profile vector:
C       LOWEQ(1)=-1 flags a full matrix:
        INTEGER LOWEQ(1)
        DATA LOWEQ/-1/
C
C       Constants used for this test matrix:
        COMPLEX*16 CZERO, CONE, ANSWER
        PARAMETER (CZERO  = (0.0D0, 0.0D0))
        PARAMETER (CONE   = (1.0D0, 0.0D0))
        PARAMETER (ANSWER = (0.5D0,-0.5D0))
C
C       Variables used to check answer:
        REAL*8 ERROR, ETEST
C
C       Local variables:
        INTEGER L_X, LENX
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
```

```
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C-----------------------------------------------------------------------
C (1)   Initialize FMS:
C-----------------------------------------------------------------------
        CALL FMSINI
C       This must be the first statement in your program.
C       It performs the following functions:
C       * Initializes FMS,
C       * Sets FMS Parameters to default values,
C       * If children processes start at the beginning of the program,
C         they are put to sleep.
C
        CALL FMSPSH(MYNAME)
C
C       The following calls to subroutine FMSIGT obtain the pointers
C       to the FMS memory management arrays IMD, RMD and CMD:
C
C       FMS contains a list of environment variables called FMS
C       Parameters.  These parameters control special features.
C       You may modify the default value of the FMS parameters in one
C       of 3 ways:
C       * Call FMSSET to interactively modify parameters at run time.
C       * Include FMSSET, Parameter statements and RETURN in the license
C         file to modify the default values,
C       * Call FMSIST, FMSRST or FMSCST in the program.
C
C       Here are some examples:
C
C       If you want to process this matrix with partial pivoting,
C       include the following:
C       CALL FMSIST ('MFMAT',3)
C
C       or
C
C       If you want to average the upper and lower triangle parts to
C       produce a symmetric matrix, include the following:
```

```
C       CALL FMSIST ('LUOK', 1)
C       NOTE: For symmetric processing, you must also change the
C       complex nonsymmetric FMS calls (beginning with CN) to
C       complex symmetric (beginning with CS).
C
C       Loop back to here to do next problem:
  100   CONTINUE
C
C       Allow for interactive modification of the FMS parameters:
C       If you do not want to do this, remove this call.
        PRINT *
        PRINT *, 'You may now set any FMS parameter.'
        PRINT *, 'When you have finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       You may also obtain the value of any FMS parameter by calling
C       FMSIGT, FMSRGT or FMSCGT.  Here we get the number of processors.
        CALL FMSIGT ('MAXCPU', MAXCPU)
C-----------------------------------------------------------------------
C       Application Specific
C-----------------------------------------------------------------------
C
C       Generate a model
C       ================
C       For this sample problem, we will generate a mesh of a cylinder.
C
C       Find out how many equations and RHS vectors to use:
        PRINT *
        PRINT *, 'Enter the model parameters below.'
        PRINT *, 'If you want to stop, enter a value of 0'
        PRINT *
        MAXEQU = ASK_I ('Enter the number of equations')
        MAXEQU = MAX0(MAXEQU,6)
        IF(MAXEQU .LE. 0) GO TO 200
        NUMVEC = ASK_I ('Enter the number of vectors')
        IF(NUMVEC .LE. 0) GO TO 200
```

```
C
      PRINT *
      PRINT *,'Start of generating model'
C
C     FMS contains memory management routines.  Subroutines FMSIMG,
C     FMSRMG and FMSCMG allocate INTEGER, REAL*8 and COMPLEX*16
C     memory respectively.  The arguments include a reference
C     array, the returned offset, and the number of words requested.
C     Allocate memory for edge list:
      CALL FMSIMG (IMD, L_IDEDGE, 3*MAXEQU)
C     CALL FMSCST ('SHOW','MEMORY')
C
C     Generate the mesh:
      NEQEST = MAXEQU
      CALL MGEN (NEQEST, NFACES, NEDGES, IMD(L_IDEDGE))
C
      PRINT *,'Number of faces.....................=', NFACES
      PRINT *,'Number of edges.....................=', NEDGES
      PRINT *,'End of generating model'
C
C     For this problem, the number of equations is the number of
C     edges.
      NUMEQ  = NEDGES
C
C     Determine order for processing faces
C     ====================================
      PRINT *,'Start of computing face order'
C
C     The amount of storage required to accumulate matrix data before
C     transferring to FMS can be reduced by changing the order of
C     processing faces.  The following code determines the optimum
C     order, IORDER(NFACES) and the number of vectors required
C     for accumulation, MAXVEC.
C
C     Allocate storage for sorting faces:
      CALL FMSIMG (IMD, L_IORDER, NFACES)
      CALL FMSIMG (IMD, L_MAXUSE, NEDGES)
      CALL FMSIMG (IMD, L_NLEFT,  NEDGES)
      CALL FMSIMG (IMD, L_IUSE  , NFACES)
```

```
        CALL FMSIMG (IMD, L_IAGE  , NEDGES)
C
        CALL ORDERF (
     1   NFACES      , NEDGES       , IMD(L_IDEDGE),
     2   MAXVEC      , IMD(L_IORDER), IMD(L_MAXUSE),
     3   IMD(L_NLEFT), IMD(L_IUSE  ), IMD(L_IAGE  ))
C
        PRINT *,'Maximum number of active vectors...=', MAXVEC
C
C       Return scratch storage:
        CALL FMSIMR (IMD, L_IAGE  , NEDGES)
        CALL FMSIMR (IMD, L_IUSE  , NFACES)
        PRINT *,'End of computing face order'
C-----------------------------------------------------------------------
C (2)   Open FMS files:
C-----------------------------------------------------------------------
        CALL CNDI  (LOWEQ, NUMEQ, 'Matrix', LUA)
        CALL FMSOV (NUMEQ, IDTYPE, NUMVEC, 'Vectors', LUX)
C-----------------------------------------------------------------------
C (3)   Write data to FMS files:
C-----------------------------------------------------------------------
        PRINT *
        PRINT *,'Starting fill'
C
C       Allocate storage for computing matrix elements:
C       ===============================================
C
C       MAXVEC shared vectors for accumulators:
C       Increase MAXVEC to allow processes to operate independently:
        MAXVEC = MAXVEC + 3*MAXCPU
        CALL FMSCMG (CMD, L_VGLOB, MAXVEC*NUMEQ)
C
C       Accumulator vector identifiers:
C       (Lists which global equations are currently held)
        CALL FMSIMG (IMD, L_IDGLOB, MAXVEC)
C
C       Process status (private for each process)
        CALL FMSIMG (IMD, L_ISTAT, MAXCPU)
C
```

```
C       Store scalar arguments in shared memory
C       =======================================
C       This is necessary if children are forked or spawned.
C       This is not necessary (but doesn't hurt) if children are threads.
        CALL FMSIMG (IMD, L_NXFACE, 6)
        L_NAVAIL = L_NXFACE + 1
        L_NFACES = L_NAVAIL + 1
        L_NEDGES = L_NFACES + 1
        L_MAXVEC = L_NEDGES + 1
        L_LUA    = L_MAXVEC + 1
        IMD(L_NXFACE) = 0
        IMD(L_NAVAIL) = MAXVEC
        IMD(L_NFACES) = NFACES
        IMD(L_NEDGES) = NEDGES
        IMD(L_MAXVEC) = MAXVEC
C
C       Initialize values:
C       ==================
C       Initialize accumulators to available:
        DO I=1,MAXVEC
           IMD(L_IDGLOB + I - 1) = 0
        END DO
C
C       Initialize NLEFT to MAXUSE:
        DO I=1,NEDGES
           IMD(L_NLEFT-1+I) = IMD(L_MAXUSE-1+I)
        END DO
C
C       Initialize addresses of private data:
        LL_ISTAT  = L_ISTAT
C
C       Leave room for local vectors:
        CALL FMSIGT ('MAXMD', MAXMD)
        LENC16 = 2*(3*NUMEQ)
        LENC16 = LENC16 + 2048
        MD_NEW = MAXMD - MAXCPU*LENC16
        CALL FMSIST ('MAXMD', MAXMD)
C
C       Initialize FMSROW:
```

```
C       Use all remaining memory for buffers:
C       NOTE: This is a common set of buffers for all processes.
C       Therefore FMSROW is initialized once by parent before starting
C       children.
        CALL FMSROW (-1, CMD(L_VGLOB), LUA)
C
C       Reset MAXMD to allow space for arrays allocated by children:
        CALL FMSIST ('MAXMD', MAXMD)
C
C       Copy LUA(25) to shared memory for children:
C       Note: If children are threads, you may use LUA directly.
        DO I = 1,25
           IMD(L_LUA+I-1) = LUA(I)
        END DO
C
C       Generate matrix elements in parallel
C       ====================================
C       Loop over children processes:
        DO ICPU = 2,MAXCPU
C          Increment addresses of private data:
           LL_ISTAT  = LL_ISTAT  + 1
C          Place work for children in a queue:
           CALL FMSPAR (13, FILL,
     1        IMD(L_NXFACE ), IMD(L_NAVAIL ),
     2        IMD(L_NFACES ), IMD(L_NEDGES ), IMD(L_MAXVEC ),
     3        IMD(L_LUA    ), IMD(L_IDEDGE ), IMD(L_IORDER ),
     4        IMD(L_MAXUSE ), IMD(L_NLEFT  ),
     5        CMD(L_VGLOB  ), IMD(L_IDGLOB ), IMD(LL_ISTAT ))
        END DO
C
C       Start the children running:
        IF(MAXCPU .GT. 1) CALL FMSRUN
C
C       Do parent's part:
        LL_ISTAT  = L_ISTAT
        CALL FILL(
     1     IMD(L_NXFACE ), IMD(L_NAVAIL ),
     2     IMD(L_NFACES ), IMD(L_NEDGES ), IMD(L_MAXVEC ),
```

```
     3     LUA           , IMD(L_IDEDGE ), IMD(L_IORDER ),
     4     IMD(L_MAXUSE ), IMD(L_NLEFT  ),
     5     CMD(L_VGLOB  ), IMD(L_IDGLOB ), IMD(LL_ISTAT ))
C
C       Wait for the children to complete:
        IF(MAXCPU .GT. 1) CALL FMSYNC
C
C       Check the status of the processes:
        WRITE(6,2002)
        NTOTAL = 0
        LL_ISTAT = L_ISTAT
        DO I=1,MAXCPU
           WRITE(6,2004) I, IMD(LL_ISTAT)
           IF(IMD(LL_ISTAT) .LT. 0) STOP'Error during fill'
           NTOTAL   = NTOTAL   + IMD(LL_ISTAT)
           LL_ISTAT = LL_ISTAT + 1
        END DO
        WRITE(6,2003) NTOTAL
        IF(NTOTAL .NE. NFACES) STOP'Not all faces processed'
C
C       End FMSROW:
        CALL FMSROW (NUMEQ+1, CMD(L_VGLOB), LUA)
C
C       Free storage used for fill
C       ==========================
        CALL FMSIMR (IMD, L_NXFACE, 5 + 25)
        CALL FMSIMR (IMD, L_ISTAT,  MAXCPU)
        CALL FMSIMR (IMD, L_IDGLOB, MAXVEC)
        CALL FMSCMR (CMD, L_VGLOB,  MAXVEC*NUMEQ)
        CALL FMSIMR (IMD, L_NLEFT,  NEDGES)
        CALL FMSIMR (IMD, L_MAXUSE, NEDGES)
        CALL FMSIMR (IMD, L_IORDER, NFACES)
        CALL FMSIMR (IMD, L_IDEDGE, 3*MAXEQU)
        PRINT *,'Ending fill'
C
C       Write RHS vector
C       ================
        LENX = LUX(4)/2
        CALL FMSCMG (CMD, L_RHS, LENX)
```

```
         CMD(L_RHS) = CONE
         DO I=2,NUMEQ
            CMD(L_RHS+I-1) = CZERO
         END DO
         LOCD = 1
         DO IVEC = 1,NUMVEC
            CALL FMSWRT (LUX, LOCD, CMD(L_RHS), LUX(4))
            LOCD = LOCD + LUX(4)
         END DO
         CALL FMSCMR (CMD, L_RHS, LENX)
C--------------------------------------------------------------------
C (4)    Perform matrix algebra:
C--------------------------------------------------------------------
         CALL CNDF (LUA, LUA, LUX, LUX, 0)
         CALL CNDS (LUA, LUX, LUX, NUMVEC, 0)
C--------------------------------------------------------------------
C (5)    Read data from FMS files:
C--------------------------------------------------------------------
         CALL FMSCMG (CMD, L_X, LENX)
         LOCD = 1
         ERROR = 0.0D0
         DO IVEC = 1,NUMVEC
            CALL FMSRED (LUX, LOCD, CMD(L_X), LUX(4))
            LOCD  = LOCD + LUX(4)
            DO I=1,NUMEQ
               ETEST = ABS( CMD(L_X + I - 1) - ANSWER )
               IF(ETEST .GT. ERROR) ERROR = ETEST
            END DO
         END DO
         PRINT *,'MAXIMUM ERROR =', ERROR
         CALL FMSCMR (CMD, L_X, LENX)
C--------------------------------------------------------------------
C (6)    End FMS:
C--------------------------------------------------------------------
C        Do the next problem.
         CALL FMSCV (LUX)
         CALL FMSCM (LUA)
         GO TO 100
   200   CONTINUE
```

```
        CALL FMSPOP(MYNAME)
        CALL FMSEND
C-------------------------------------------------------------------------
C       FORMAT STATEMENTS
C-------------------------------------------------------------------------
 2002   FORMAT (/
     1  ' Number of faces computed by each process'/
     2  ' PROCESS     FACES'/
     3  ' =======     =====')
 2003   FORMAT (
     1  '             -----'/
     2  ' TOTAL', I12)
 2004   FORMAT (I5,I13)
        END
C=========================================================================
        SUBROUTINE ORDERF
     1   (NFACES, NEDGES, IDEDGE,
     2    MAXVEC, IORDER, MAXUSE,
     3    NLEFT, IUSE, IAGE)
C=========================================================================
C       DESCRIPTION:
C
C       This next section of code computes the order of processing
C       the faces, IORDER(NFACES) that results in the minimum amount
C       of storage required to accumulate the matrix elements.
C       NOTE: For this problem, each matrix element has 2 contributions,
C       one from each face.  Before writing the matrix data to FMS,
C       we must compute both contributions to make the term complete.
C       We also note that the matrix terms are computed by integrals
C       over the triangle faces.  Therefore the order of computing the
C       matrix terms is determined by the order of processing the
C       faces.
C
C       This approach uses a reasonable amount of memory to accumulate
C       face integrals and then writes the parts of the matrix that
C       become complete to FMS.  The benefits are:
C       * the integrals only need to be computed once
C       * there is no rereading of data from disk to do the
C         accumulation.
```

```
C       * the algorithm easily runs in parallel.
C
C       The general scheme is:
C       COMPLEX*16 Z(3,3)
C       COMPLEX*16 VGLOB (NUMEQ,MAXVEC)
C       INTEGER    IORDER(NFACES)
C       Outer loop over faces:
C       DO IFACE=1,NFACES
C          Change to optimized processing order:
C          MYFACE = IORDER(IFACE)
C          DO JFACE=1,NFACES
C             Compute the integral of MYFACE due to JFACE and store
C             the results in VLOCAL.
C          END DO
C          The vectors in VLOCAL are now complete.  They contain the
C          contributions to MYFACE from all other faces.
C
C          Before the vectors can be written to FMS, we must wait for
C          their second contribution (each vector will appear twice,
C          unless it is on an edge).  We therefore store the vectors
C          into VGLOB, until they become complete.
C
C          Add VLOCAL TO VGLOB.
C          IF(Any of the VLOCAL vectors are complete) THEN
C             Write complete VGLOB vector to FMS
C             Free VGLOB storage for another vector
C          END IF
C       END DO
C
C       By properly picking the face processing order, we can reduce
C       the number of VGLOB vectors to a reasonable level.
C       The following code determins this level:
C
C       FORMAL PARAMETERS:
C          (R ) NFACES = Integer scalar
C               Number of faces
C
C          (R ) NEDGES = Integer scalar
C               Number of edges
```

```
C
C          (R ) IDEDGE(3,NFACES) = Integer array.
C               Lists the edges associated with a face
C
C          (RW) IORDER(NFACES) = Integer array.
C               On input, specifies the list of starting faces.
C               At least one face must be specified in IORDER(1).
C               The value following the last face specified must be 0.
C               On output, specifies the order for processing faces.
C
C          ( W) MAXVEC = Integer array.
C               Maximum number of active edges.  This determines the
C               number of vectors which must be held in memory to
C               accumulate results.  The objective of this subroutine
C               is to select the IORDER array to minimize this value.
C
C          ( W) MAXUSE(NEDGES) = Integer array.
C               Maximum number of faces using each edge.  For interior
C               edges, the value will be 2.  For edges on the boundary,
C               the value will be 1.
C
C          (  ) NLEFT(NEDGES) = Integer array.
C               Scratch storage used to hold the number of times the
C               edge must be found before it can be eliminated.
C               This vector is initialized to MAXUSE.
C
C          (  ) IUSE(NFACES) = Integer array.
C               Scratch storage used to flag available faces.
C
C          (  ) IAGE(NEDGES) = Integer array.
C               Scratch storage used to hold the age of active edges.
C-----------------------------------------------------------------------
C       Formal Parameters
C-----------------------------------------------------------------------
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='ORDERF')
        INTEGER NFACES
        INTEGER NEDGES
        INTEGER IDEDGE(3,NFACES)
```

```
      INTEGER IORDER(NFACES)
      INTEGER MAXVEC
      INTEGER MAXUSE(NEDGES)
      INTEGER IUSE  (NFACES)
      INTEGER IAGE  (NEDGES)
      INTEGER NLEFT (NEDGES)
C-----------------------------------------------------------------------
C     Local Variables
C-----------------------------------------------------------------------
      INTEGER IFACE
      INTEGER JFACE
      INTEGER IEDGE
      INTEGER MYEDGE
      INTEGER MAXAGE
      INTEGER NUMACT
C-----------------------------------------------------------------------
      CALL FMSPSH(MYNAME)
C
C     Initialize starting face:
C     You should pick the face that is the furthest from the center.
C     Here we will do something simple:
      IORDER(1) = 1
      IORDER(2) = 0
C
C ( 1)  Initialize IUSE to unused:
      DO IFACE = 1,NFACES
         IUSE(IFACE) = 0
      END DO
C
C ( 2)  Find the maximum number of times each edge is used:
      DO IEDGE = 1,NEDGES
         MAXUSE(IEDGE) = 0
      END DO
      DO IFACE = 1,NFACES
         DO I = 1,3
            IEDGE = IDEDGE(I,IFACE)
            MAXUSE(IEDGE) = MAXUSE(IEDGE) + 1
         END DO
```

```
         END DO
C
C ( 3)  Initialize the number of times each edge must be found before
C       it can be eliminated.
         DO IEDGE = 1,NEDGES
            NLEFT(IEDGE) = MAXUSE(IEDGE)
         END DO
C
C ( 4)  Initialize the age of each edge to inactive:
         DO IEDGE = 1,NEDGES
            IAGE(IEDGE) = 0
         END DO
C
C ( 5)  Find the face order:
         NUMACT = 0
         MAXVEC = 0
         NSTART = 0
         DO IFACE = 1,NFACES
            IF(NSTART .EQ. 0) THEN
               IF(IORDER(IFACE) .EQ. 0) THEN
C                 This is the first non-specified face:
                  NSTART = IFACE - 1
                  IF(NSTART .EQ. 0)
     1               STOP 'ORDERF: No starting faces specified'
               ELSE
C                 This face is specified:
                  GO TO 100
               END IF
            END IF
C
C           Find the oldest edge.
            MAXAGE = -1
            MYEDGE = 0
            DO IEDGE = 1,NEDGES
               IF( IAGE(IEDGE) .GT. MAXAGE ) THEN
                  MYEDGE = IEDGE
                  MAXAGE = IAGE(IEDGE)
               END IF
            END DO
```

```
            IF(MYEDGE .EQ. 0) STOP 'Oldest edge not found'
C
C           Find a face which uses the oldest edge:
            DO JFACE = 1,NFACES
               IF( IUSE(JFACE) .EQ. 0 ) THEN
                  DO I= 1,3
                     IF(IDEDGE(I,JFACE) .EQ. MYEDGE) THEN
                        IORDER(IFACE) = JFACE
                        GO TO 100
                     END IF
                  END DO
               END IF
            END DO
            STOP 'Face not found which uses oldest edge'
C
C           The new face is IORDER(IFACE):
  100       CONTINUE
            JFACE = IORDER(IFACE)
C           PRINT *,'ORDERF: IORDER(',IFACE,') =',JFACE
            IUSE(JFACE) = 1
C
C           Age all the active edges:
            DO IEDGE = 1,NEDGES
               IF(IAGE(IEDGE) .GT. 0) THEN
C                 Edge is active:
                  IAGE(IEDGE) = IAGE(IEDGE) + 1
               END IF
            END DO
C
C           Start new edges:
            DO I = 1,3
               IEDGE = IDEDGE(I,JFACE)
               IF(IAGE(IEDGE) .EQ. 0) THEN
C                 New active edge:
                  IAGE(IEDGE) = 1
                  NUMACT = NUMACT + 1
               END IF
            END DO
            IF(NUMACT .GT. MAXVEC) MAXVEC = NUMACT
```

```
C          Eliminate edges which are now complete:
           DO I = 1,3
              IEDGE = IDEDGE(I,JFACE)
              NLEFT(IEDGE) = NLEFT(IEDGE) - 1
              IF(NLEFT(IEDGE) .EQ. 0) THEN
C                Edge is complete:
                 IAGE(IEDGE) = 0
                 NUMACT = NUMACT - 1
              END IF
           END DO
C
C          Do the next face:
        END DO
        CALL FMSPOP(MYNAME)
        RETURN
        END
C=======================================================================
        SUBROUTINE FILL (
     1   NXFACE, NAVAIL,
     2   NFACES, NEDGES, MAXVEC,
     3   LUA   , IDEDGE, IORDER,
     4   MAXUSE, NLEFT ,
     5   VGLOB , IDGLOB, ISTAT)
C=======================================================================
C       DESCRIPTION:
C          This subroutine computes the matrix elements.
C          It is designed to be run in parallel.
C
C       FORMAL PARAMETERS:
C          (RW) NXFACE = Integer scalar (shared, read, write)
C               Outer face loop counter.
C
C          (RW) NAVAIL = Integer scalar (shared, read, write)
C               Number of global vectors available.
C               (Initialized to MAXVEC)
C
C          (R ) NFACES = Integer scalar (shared, read only)
C               Number of faces
C
```

```
C          (R ) NEDGES = Integer scalar (shared, read only)
C               Number of edges
C
C          (R ) MAXVEC = Integer scalar. (shared, read only)
C               Number of vectors in memory
C
C          (R ) LUA(25) = Integer array.
C               Matrix file attributes.
C
C          (R ) IDEDGE(3,NFACES) = Integer array. (shared, read only)
C               Lists the edges associated with a face.
C
C          (R ) IORDER(NFACES) = Integer array (shared, read only)
C               Lists the order for processing faces
C
C          (R ) MAXUSE(NEDGES) = Integer array (shared, read only)
C               Lists the number of times each edge must be used.
C
C          (RW) NLEFT(NEDGES) = Integer array (shared, read, write)
C               Scratch storage, initialized to MAXUSE
C
C          (RW) VGLOB(NEDGES,MAXVEC) = Complex*16 array (shared, r, w)
C               Scratch storage for accumulating vectors
C
C          (RW) IDGLOB(MAXVEC) = Integer array (shared, r, w)
C               Scratch storage for vector numbers in V.
C
C          ( W) ISTAT = Integer scalar (private, write)
C                       Return status.
C                     =-1, Error
C                     >=0, Number of faces processed.
C-----------------------------------------------------------------------
C       Formal Parameters
C-----------------------------------------------------------------------
        CHARACTER*4 MYNAME
        PARAMETER (MYNAME='FILL')
        INTEGER    NXFACE
        INTEGER    NAVAIL
        INTEGER    NFACES
```

```
        INTEGER    NEDGES
        INTEGER    IDEDGE(3,NFACES)
        INTEGER    LUA(25)
        INTEGER    MAXVEC
        INTEGER    IORDER(NFACES)
        INTEGER    MAXUSE(NEDGES)
        INTEGER    NLEFT (NEDGES)
        COMPLEX*16 VGLOB(NEDGES, MAXVEC)
        INTEGER    IDGLOB(MAXVEC)
        INTEGER    ISTAT
C       VOLATILE   NXFACE, NAVAIL
C-----------------------------------------------------------------------
C       Local Variables
C-----------------------------------------------------------------------
        INTEGER    IFACE
        INTEGER    JFACE
        INTEGER    ITEMP
        COMPLEX*16 CMD(0:1)
        POINTER (CMD_PTR,CMD)
C-----------------------------------------------------------------------
        CALL FMSPSH(MYNAME)
        ISTAT = 0
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        CALL FMSCMG (CMD, L_VLOCAL, 3*NEDGES)
C
C       Get your starting face number:
C       Get exclusive use:
        CALL FMSONE
C          NXFACE = NXFACE + 1
C          MYFACE = NXFACE
           MYFACE = INTINC(NXFACE)
        CALL FMSALL
C       NOTE: We saved the face number NXFACE in a local variable
C       MYFACE, because NXFACE may be changed by another process
C       as soon as we called FMSALL.
C
C       Outer loop over faces:
  100   CONTINUE
```

```
            IF(MYFACE .GT. NFACES) THEN
C              This process is done.
               CALL FMSCMR (CMD, L_VLOCAL, 3*NEDGES)
         CALL FMSPOP(MYNAME)
               RETURN
            END IF
            IFACE  = IORDER(MYFACE)
            ISTAT  = ISTAT  + 1
C
C           Inner loop over faces:
C           NOTE: I may process these in any order, because I am doing
C                 all of them and I have room to store the results.
            DO JFACE = 1,NFACES
C
C              Compute the matrix relating IFACE to JFACE:
C              There will need to be a lot more information passed
C              through to this routine.  The following is used to
C              generate a test matrix only.
C
               CALL FILL2 (IFACE, JFACE, IDEDGE, MAXUSE, NEDGES,
     1                     CMD(L_VLOCAL))
            END DO
C
C           The 3 vectors VLOCAL are now complete.  They contain
C           1/2 of the terms for interior edges.
C           We now need to accumulate these results into the global
C           arrays VGLOB.
C
C           Critical Section
C           ================
            CALL FMSONE
C
C              This code writes to shared variables.  Therefore it can
C              be executed by only one process at a time.
C
C              Transfer private vectors to global arrays:
               L_V = L_VLOCAL - NEDGES
               DO I = 1,3
```

```
                  L_V   = L_V + NEDGES
                  IEQN  = IDEDGE(I,IFACE)
C                 Look for vector IEQN in the accumulation list:
                  MYVEC = 0
                  DO IVEC = 1,MAXVEC
                     IF(IEQN .EQ. IDGLOB(IVEC)) THEN
C                       Add to existing vector:
                        MYVEC  = IVEC
                        CALL CVADD (VGLOB(1,MYVEC), CMD(L_V), NEDGES)
                        GO TO 220
                     END IF
                     IF(IDGLOB(IVEC) .EQ. 0) THEN
C                       This slot is free:
                        IF(MYVEC .EQ. 0) MYVEC = IVEC
                     END IF
                  END DO
                  IF(MYVEC .EQ. 0) THEN
C                    Error:
                     WRITE(6,2000) MYNODE, IDEDGE(I,IFACE),
     1                  (II, IDGLOB(II),II=1,MAXVEC)
                     ISTAT = -1
                     CALL FMSCMR (CMD, L_VLOCAL, 3*NEDGES)
         CALL FMSPOP(MYNAME)
                     RETURN
                  END IF
C                 New vector:
                  ITEMP = INTDEC(NAVAIL)
                  IDGLOB(MYVEC) = IEQN
                  CALL CVMOVE (VGLOB(1,MYVEC), CMD(L_V), NEDGES)
  220             CONTINUE
                  NLEFT(IEQN) = NLEFT(IEQN) - 1
                  IF(NLEFT(IEQN) .EQ. 0) THEN
C                    Vector is ready to be written out:
                     CALL FMSROW (IEQN, VGLOB(1,MYVEC), LUA)
C                    Free the storage:
                     IDGLOB(MYVEC) = 0
                     ITEMP  = INTINC(NAVAIL)
                  END IF
               END DO
```

```
C
C             Get your next face number:
C             NXFACE = NXFACE + 1
C             MYFACE = NXFACE
              MYFACE = INTINC(NXFACE)
           CALL FMSALL
C
C          End of Critical Section
C          =======================
C
C          Do the next face:
         GO TO 100
 2000    FORMAT (/
     1   ' FILL: No global vectors available for process',I5/
     2   '       Trying to store vector',I10/
     3   '       Vector     Contents'/
     4   '       ======     ========'/
     5   (I13,I13))
         END
C=======================================================================
         SUBROUTINE FILL2 (IFACE, JFACE, IDEDGE, MAXUSE, NUMEQ, VLOCAL)
C=======================================================================
C        This is a dummy subroutine to generate test matrix elements.
         INTEGER IFACE, JFACE, IDEDGE(3,*), NUMEQ, MAXUSE(NUMEQ)
         REAL*8 TEMP, DTEMP
         COMPLEX*16 ZIJ
         COMPLEX*16 VLOCAL(NUMEQ,3)
         CHARACTER*5 MYNAME
         PARAMETER (MYNAME='FILL2')
         CALL FMSPSH(MYNAME)
         DO I=1,3
            IGLOB = IDEDGE(I,IFACE)
            TEMP  = (1.0D0)/(DFLOAT(MAXUSE(IGLOB)))
            DO J=1,3
               JGLOB = IDEDGE(J,JFACE)
               IF( IGLOB .EQ. JGLOB ) THEN
C                 This is a diagonal element:
                  IF ( IGLOB .EQ. 1) THEN
                     DTEMP = DFLOAT(NUMEQ)*TEMP
```

```
                     ZIJ = DCMPLX(DTEMP,DTEMP)
                  ELSE
                     ZIJ = DCMPLX(TEMP,TEMP)
                  END IF
               ELSE IF( (IGLOB .EQ. 1)  .OR. (JGLOB .EQ. 1) ) THEN
C                 Term is in first row or column:
                  ZIJ = DCMPLX(-TEMP,-TEMP)
               ELSE
C                 Interior element:
                  ZIJ = DCMPLX(0.0D0,0.0D0)
               END IF
               VLOCAL(JGLOB,I) = ZIJ
            END DO
         END DO
         CALL FMSPOP(MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE MGEN (NEQEST, NFACES, NEDGES, IDEDGE)
C=======================================================================
         INTEGER NEQEST
         INTEGER NFACES
         INTEGER NEDGES
         INTEGER IDEDGE(3,NEQEST)
         REAL*8  RN
         CHARACTER*4 MYNAME
         PARAMETER (MYNAME='MGEN')
         CALL FMSPSH(MYNAME)
C        Generate a mesh of quadrilaterals that is
C        approximately square.  Each quadrilateral will
C        be divided into four triangles.  Each interior
C        quadrilateral will add 6 equations (4 interior
C        edges and 2 sides).
C        Compute the number of quadrilaterals:
         RN = NEQEST/6
C        Compute the number of quadrilaterals on an edge:
         RN = SQRT(RN)
         N  = RN
         N2 = N  + N
```

```
         N3 = N2 + N
         N4 = N3 + N
         N6 = N4 + N2
         M  = (NEQEST + N2)/N6
C        PRINT *,'MGEN: Mesh is ',N,' by ',M
         NFACES = 0
         NEDGES = 0
C        Loop down the cylinder:
         DO J = 1,M
C           Loop around the cylinder:
            DO I = 1,N
               I2 = I + I
C              Generate the 4 face numbers in the square:
               IFACE = NFACES + I
               JFACE = NFACES + N + I2 - 1
               KFACE = JFACE  + 1
               LFACE = NFACES + N3 + I
C              Generate the edge list for each face:
               IDEDGE(1,IFACE) = NEDGES + I
               IDEDGE(2,IFACE) = NEDGES + N + I2
               IDEDGE(3,IFACE) = IDEDGE(2,IFACE) - 1
               IDEDGE(1,JFACE) = NEDGES + N3 + I
               IDEDGE(2,JFACE) = IDEDGE(3,IFACE)
               IDEDGE(3,JFACE) = NEDGES + N4 + I2 - 1
               IDEDGE(1,KFACE) = IDEDGE(2,IFACE)
               IDEDGE(2,KFACE) = IDEDGE(1,JFACE) + 1
               IDEDGE(3,KFACE) = IDEDGE(3,JFACE) + 1
               IDEDGE(1,LFACE) = IDEDGE(3,JFACE)
               IDEDGE(2,LFACE) = IDEDGE(3,KFACE)
               IDEDGE(3,LFACE) = NEDGES + N6 + I
            END DO
C           Wrap for a cylinder:
            IDEDGE(2,KFACE) = IDEDGE(2,KFACE) - N
            NFACES = NFACES + N4
            NEDGES = NEDGES + N6
         END DO
C        Leave cylinder open:
         NEDGES = NEDGES + N
C
```

```
C       WRITE(6,5000) (I,(IDEDGE(J,I), J=1,3), I=1,NFACES)
        CALL FMSPOP(MYNAME)
        RETURN
C5000   FORMAT(/
C    1  ' FACE   EDGE 1   EDGE 2   EDGE 3'/
C    2  ' ====   ======   ======   ======'/
C    3  (I5,3I9))
        END
C=======================================================================
        SUBROUTINE CVMOVE (A, B, N)
C=======================================================================
        COMPLEX*16 A(N), B(N)
        DO I = 1,N
           A(I) = B(I)
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE CVADD (A, B, N)
C=======================================================================
        COMPLEX*16 A(N), B(N)
        DO I = 1,N
           A(I) = A(I) + B(I)
        END DO
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION INTINC (I)
C=======================================================================
C       This function increments a volatile shared variable.  It is
C       placed in a subroutine to prevent some compilers from storing
C       the value in a register and not updating it.
        INTEGER I
        I = I + 1
        INTINC = I
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION INTDEC (I)
```

```
C=======================================================================
C       This function decrements a volatile shared variable.  It is
C       placed in a subroutine to prevent some compilers from storing
C       the value in a register and not updating it.
        INTEGER I
        I = I - 1
        INTDEC = I
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

```
      PROGRAM RCSFMS
C-----------------------------------------------------------------------
C     This is a sample program that illustrates the following:
C
C     * How to use the FMS out-of-core solver in a RCS program,
C
C     * How to generate matrix elements efficiently,
C
C     * How to use the FMS parallel processing utilities to generate
C       matrix elements in parallel.
C
C     It is assumed that the model consists of a mesh composed of
C     triangles, and the unknowns (equations) are the values on the
C     triangle edges.  It is also assumed that each triangle impacts
C     every other triangle, producing a full matrix.
C
C     It is also assumed that the data is double precision complex,
C     (COMPLEX*16).
C
C     The test matrix produced by this example has a solution vector
C     of all (0.5,-0.5)'s.
C
C-----------------------------------------------------------------------
C     DECLARATIONS
C-----------------------------------------------------------------------
C
C     Program name:
      CHARACTER*9 MYNAME
      PARAMETER (MYNAME='EXAMPLE_9')
C
C     Input data functions:
      INTEGER ASK_I
C
C     Subroutines called in parallel (passed to FMSPAR) must be
C     declared external.
      EXTERNAL FILL
C
C     FMS file object handles (attribute lists)
C     Matrix file:
      INTEGER LUA(25)
C     Vector file:
      INTEGER LUX(25)
C
C     Data type:
      INTEGER IDTYPE
C     Complex*16:
      PARAMETER (IDTYPE=2)
C
C     FMS profile vector:
C     LOWEQ(1)=-1 flags a full matrix:
      INTEGER LOWEQ(1)
      DATA LOWEQ/-1/
C
C     Constants used for this test matrix:
      COMPLEX*16 CZERO, CONE, ANSWER
      PARAMETER (CZERO  = (0.0D0, 0.0D0))
      PARAMETER (CONE   = (1.0D0, 0.0D0))
      PARAMETER (ANSWER = (0.5D0,-0.5D0))
C
C     Variables used to check answer:
      REAL*8 ERROR, ETEST
C
C     Local variables:
```

```
        INTEGER L_X, LENX
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C-----------------------------------------------------------------------
C (1)   Initialize FMS:
C-----------------------------------------------------------------------
        CALL FMSINI
C       This must be the first statement in your program.
C       It performs the following functions:
C       * Initializes FMS,
C       * Sets FMS Parameters to default values,
C       * If children processes start at the beginning of the program,
C         they are put to sleep.
C
        CALL FMSPSH (MYNAME)
C
C       The following calls to subroutine FMSIGT obtain the pointers
C       to the FMS memory management arrays IMD, RMD and CMD:
C
C       FMS contains a list of environment variables called FMS
C       Parameters.  These parameters control special features.
C       You may modify the default value of the FMS parameters in one
C       of 3 ways:
C       * Call FMSSET to interactively modify parameters at run time.
C       * Include FMSSET, Parameter statements and RETURN in the license
C         file to modify the default values,
C       * Call FMSIST, FMSRST or FMSCST in the program.
C
C       Here are some examples:
C
C       If you want to process this matrix with partial pivoting,
C       include the following:
C       CALL FMSIST ('MFMAT',3)
C
C       or
C
C       If you want to average the upper and lower triangle parts to
C       produce a symmetric matrix, include the following:
C       CALL FMSIST ('LUOK', 1)
C       NOTE: For symmetric processing, you must also change the
C       complex nonsymmetric FMS calls (beginning with CN) to
C       complex symmetric (beginning with CS).
C
C       Loop back to here to do next problem:
  100   CONTINUE
C
C       Allow for interactive modification of the FMS parameters:
C       If you do not want to do this, remove this call.
        PRINT *
        PRINT *, 'You may now set any FMS parameter.'
        PRINT *, 'When you have finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       You may also obtain the value of any FMS parameter by calling
C       FMSIGT, FMSRGT or FMSCGT.  Here we get the number of processors.
        CALL FMSIGT ('MAXCPU', MAXCPU)
```

```
C-----------------------------------------------------------------------
C       Application Specific
C-----------------------------------------------------------------------
C
C       Generate a model
C       ================
C       For this sample problem, we will generate a mesh of a cylinder.
C
C       Find out how many equations and RHS vectors to use:
        PRINT *
        PRINT *, 'Enter the model parameters below.'
        PRINT *, 'If you want to stop, enter a value of 0'
        PRINT *
        MAXEQU = ASK_I ('Enter the number of equations')
        MAXEQU = MAX0(MAXEQU,6)
        IF(MAXEQU .LE. 0) GO TO 200
        NUMVEC = ASK_I ('Enter the number of vectors')
        IF(NUMVEC .LE. 0) GO TO 200
C
        PRINT *
        PRINT *,'Start of generating model'
C
C       FMS contains memory management routines.  Subroutines FMSIMG,
C       FMSRMG and FMSCMG allocate INTEGER, REAL*8 and COMPLEX*16
C       memory respectively.  The arguments include a reference
C       array, the returned offset, and the number of words requested.
C       Allocate memory for edge list:
        CALL FMSIMG (IMD, L_IDEDGE, 3*MAXEQU)
C       CALL FMSCST ('SHOW','MEMORY')
C
C       Generate the mesh:
        NEQEST = MAXEQU
        CALL MGEN (NEQEST, NFACES, NEDGES, IMD(L_IDEDGE))
C
        PRINT *,'Number of faces.....................=', NFACES
        PRINT *,'Number of edges.....................=', NEDGES
        PRINT *,'End of generating model'
C
C       For this problem, the number of equations is the number of
C       edges.
        NUMEQ  = NEDGES
C
C       Determine order for processing faces
C       ====================================
        PRINT *,'Start of computing face order'
C
C       The amount of storage required to accumulate matrix data before
C       transferring to FMS can be reduced by changing the order of
C       processing faces.  The following code determines the optimum
C       order, IORDER(NFACES) and the number of vectors required
C       for accumulation, MAXVEC.
C
C       Allocate storage for sorting faces:
        CALL FMSIMG (IMD, L_IORDER, NFACES)
        CALL FMSIMG (IMD, L_MAXUSE, NEDGES)
        CALL FMSIMG (IMD, L_NLEFT,  NEDGES)
        CALL FMSIMG (IMD, L_IUSE  , NFACES)
        CALL FMSIMG (IMD, L_IAGE  , NEDGES)
C
        CALL ORDERF (
     1   NFACES      , NEDGES       , IMD(L_IDEDGE),
     2   MAXVEC      , IMD(L_IORDER), IMD(L_MAXUSE),
     3   IMD(L_NLEFT), IMD(L_IUSE  ), IMD(L_IAGE  ))
C
        PRINT *,'Maximum number of active vectors...=', MAXVEC
```

```
C
C       Return scratch storage:
        CALL FMSIMR (IMD, L_IAGE  , NEDGES)
        CALL FMSIMR (IMD, L_IUSE  , NFACES)
        PRINT *,'End of computing face order'
C-----------------------------------------------------------------------
C (2)   Open FMS files:
C-----------------------------------------------------------------------
        CALL CNDI  (LOWEQ, NUMEQ, 'Matrix', LUA)
        CALL FMSOV (NUMEQ, IDTYPE, NUMVEC, 'Vectors', LUX)
C-----------------------------------------------------------------------
C (3)   Write data to FMS files:
C-----------------------------------------------------------------------
        PRINT *
        PRINT *,'Starting fill'
C
C       Allocate storage for computing matrix elements:
C       ===============================================
C
C       MAXVEC shared vectors for accumulators:
C       Increase MAXVEC to allow processes to operate independently:
        MAXVEC = MAXVEC + 3*MAXCPU
        CALL FMSCMG (CMD, L_VGLOB, MAXVEC*NUMEQ)
C
C       Accumulator vector identifiers:
C       (Lists which global equations are currently held)
        CALL FMSIMG (IMD, L_IDGLOB, MAXVEC)
C
C       Process status (private for each process)
        CALL FMSIMG (IMD, L_ISTAT, MAXCPU)
C
C       Store scalar arguments in shared memory
C       =======================================
C       This is necessary if children are forked or spawned.
C       This is not necessary (but doesn't hurt) if children are threads.
        CALL FMSIMG (IMD, L_NXFACE, 5 + 25)
        L_NAVAIL = L_NXFACE + 1
        L_NFACES = L_NAVAIL + 1
        L_NEDGES = L_NFACES + 1
        L_MAXVEC = L_NEDGES + 1
        L_LUA    = L_MAXVEC + 1
        IMD(L_NXFACE) = 0
        IMD(L_NAVAIL) = MAXVEC
        IMD(L_NFACES) = NFACES
        IMD(L_NEDGES) = NEDGES
        IMD(L_MAXVEC) = MAXVEC
C
C       Initialize values:
C       ==================
C       Initialize accumulators to available:
        DO I=1,MAXVEC
           IMD(L_IDGLOB + I - 1) = 0
        END DO
C
C       Initialize NLEFT to MAXUSE:
        DO I=1,NEDGES
           IMD(L_NLEFT-1+I) = IMD(L_MAXUSE-1+I)
        END DO
C
C       Initialize addresses of private data:
        LL_ISTAT  = L_ISTAT
C
C       Leave room for local vectors:
        CALL FMSIGT ('MAXMD', MAXMD)
```

```
      LENC16 = 2*(3*NUMEQ)
      LENC16 = LENC16 + 2048
      MD_NEW = MAXMD - MAXCPU*LENC16
      CALL FMSIST ('MAXMD', MD_NEW)
C
C     Initialize FMSROW:
C     Use all remaining memory for buffers:
C     NOTE: This is a common set of buffers for all processes.
C     Therefore FMSROW is initialized once by parent before starting
C     children.
      CALL FMSROW (-1, CMD(L_VGLOB), LUA)
C
C     Reset MAXMD to allow space for arrays allocated by children:
      CALL FMSIST('MAXMD', MAXMD)
C
C     Copy LUA(25) to shared memory for children:
C     Note: If children are threads, you may use LUA directly.
      DO I = 1,25
         IMD(L_LUA+I-1) = LUA(I)
      END DO
C
C     Generate matrix elements in parallel
C     ====================================
C     Loop over children processes:
      DO ICPU = 2,MAXCPU
C        Increment addresses of private data:
         LL_ISTAT  = LL_ISTAT  + 1
C        Place work for children in a queue:
         CALL FMSPAR (13, FILL,
     1      IMD(L_NXFACE ), IMD(L_NAVAIL ),
     2      IMD(L_NFACES ), IMD(L_NEDGES ), IMD(L_MAXVEC ),
     3      IMD(L_LUA    ), IMD(L_IDEDGE ), IMD(L_IORDER ),
     4      IMD(L_MAXUSE ), IMD(L_NLEFT  ),
     5      CMD(L_VGLOB  ), IMD(L_IDGLOB ), IMD(LL_ISTAT ))
      END DO
C
C     Start the children running:
      IF(MAXCPU .GT. 1) CALL FMSRUN
C
C     Do parent's part:
      LL_ISTAT  = L_ISTAT
      CALL FILL(
     1   IMD(L_NXFACE ), IMD(L_NAVAIL ),
     2   IMD(L_NFACES ), IMD(L_NEDGES ), IMD(L_MAXVEC ),
     3   LUA           , IMD(L_IDEDGE ), IMD(L_IORDER ),
     4   IMD(L_MAXUSE ), IMD(L_NLEFT  ),
     5   CMD(L_VGLOB  ), IMD(L_IDGLOB ), IMD(LL_ISTAT ))
C
C     Wait for the children to complete:
      IF(MAXCPU .GT. 1) CALL FMSYNC
C
C     Check the status of the processes:
      WRITE(6,2002)
      NTOTAL = 0
      LL_ISTAT = L_ISTAT
      DO I=1,MAXCPU
         WRITE(6,2004) I, IMD(LL_ISTAT)
         IF(IMD(LL_ISTAT) .LT. 0) STOP'Error during fill'
         NTOTAL   = NTOTAL   + IMD(LL_ISTAT)
         LL_ISTAT = LL_ISTAT + 1
      END DO
      WRITE(6,2003) NTOTAL
      IF(NTOTAL .NE. NFACES) STOP'Not all faces processed'
C
C     End FMSROW:
```

```
         CALL FMSROW (NUMEQ+1, CMD(L_VGLOB), LUA)
C
C        Free storage used for fill
C        ==========================
         CALL FMSIMR (IMD, L_NXFACE, 5 + 25)
         CALL FMSIMR (IMD, L_ISTAT,  MAXCPU)
         CALL FMSIMR (IMD, L_IDGLOB, MAXVEC)
         CALL FMSCMR (CMD, L_VGLOB,  MAXVEC*NUMEQ)
         CALL FMSIMR (IMD, L_NLEFT,  NEDGES)
         CALL FMSIMR (IMD, L_MAXUSE, NEDGES)
         CALL FMSIMR (IMD, L_IORDER, NFACES)
         CALL FMSIMR (IMD, L_IDEDGE, 3*MAXEQU)
         PRINT *,'Ending fill'
C
C        Write RHS vector
C        ================
         LENX = LUX(4)/2
         CALL FMSCMG (CMD, L_RHS, LENX)
         CMD(L_RHS) = CONE
         DO I=2,NUMEQ
            CMD(L_RHS+I-1) = CZERO
         END DO
         LOCD = 1
         DO IVEC = 1,NUMVEC
            CALL FMSWRT (LUX, LOCD, CMD(L_RHS), LUX(4))
            LOCD = LOCD + LUX(4)
         END DO
         CALL FMSCMR (CMD, L_RHS, LENX)
C-----------------------------------------------------------------------
C (4)    Perform matrix algebra:
C-----------------------------------------------------------------------
         CALL CNDF (LUA, LUA, LUX, LUX, 0)
         CALL CNDS (LUA, LUX, LUX, NUMVEC, 0)
C-----------------------------------------------------------------------
C (5)    Read data from FMS files:
C-----------------------------------------------------------------------
         CALL FMSCMG (CMD, L_X, LENX)
         LOCD = 1
         ERROR = 0.0D0
         DO IVEC = 1,NUMVEC
            CALL FMSRED (LUX, LOCD, CMD(L_X), LUX(4))
            LOCD  = LOCD + LUX(4)
            DO I=1,NUMEQ
               ETEST = ABS( CMD(L_X + I - 1) - ANSWER )
               IF(ETEST .GT. ERROR) ERROR = ETEST
            END DO
         END DO
         PRINT *,'MAXIMUM ERROR =', ERROR
         CALL FMSCMR (CMD, L_X, LENX)
C-----------------------------------------------------------------------
C (6)    End FMS:
C-----------------------------------------------------------------------
C        Do the next problem.
         CALL FMSCV (LUX)
         CALL FMSCM (LUA)
         GO TO 100
  200    CONTINUE
         CALL FMSPOP (MYNAME)
         CALL FMSEND
C-----------------------------------------------------------------------
C        FORMAT STATEMENTS
C-----------------------------------------------------------------------
 2002    FORMAT (/
     1   ' Number of faces computed by each process'/
     2   ' PROCESS     FACES'/
```

```
     3  ' =======     =====')
 2003   FORMAT (
     1  '             -----'/
     2  ' TOTAL', I12)
 2004   FORMAT (I5,I13)
        END
C=======================================================================
        SUBROUTINE ORDERF
     1   (NFACES, NEDGES, IDEDGE,
     2    MAXVEC, IORDER, MAXUSE,
     3    NLEFT, IUSE, IAGE)
C=======================================================================
C       DESCRIPTION:
C
C       This next section of code computes the order of processing
C       the faces, IORDER(NFACES) that results in the minimum amount
C       of storage required to accumulate the matrix elements.
C       NOTE: For this problem, each matrix element has 2 contributions,
C       one from each face.  Before writing the matrix data to FMS,
C       we must compute both contributions to make the term complete.
C       We also note that the matrix terms are computed by integrals
C       over the triangle faces.  Therefore the order of computing the
C       matrix terms is determined by the order of processing the
C       faces.
C
C       This approach uses a reasonable amount of memory to accumulate
C       face integrals and then writes the parts of the matrix that
C       become complete to FMS.  The benefits are:
C       * the integrals only need to be computed once
C       * there is no rereading of data from disk to do the
C         accumulation.
C       * the algorithm easily runs in parallel.
C
C       The general scheme is:
C       COMPLEX*16 Z(3,3)
C       COMPLEX*16 VGLOB (NUMEQ,MAXVEC)
C       INTEGER    IORDER(NFACES)
C       Outer loop over faces:
C       DO IFACE=1,NFACES
C          Change to optimized processing order:
C          MYFACE = IORDER(IFACE)
C          DO JFACE=1,NFACES
C             Compute the integral of MYFACE due to JFACE and store
C             the results in VLOCAL.
C          END DO
C          The vectors in VLOCAL are now complete.  They contain the
C          contributions to MYFACE from all other faces.
C
C          Before the vectors can be written to FMS, we must wait for
C          their second contribution (each vector will appear twice,
C          unless it is on an edge).  We therefore store the vectors
C          into VGLOB, until they become complete.
C
C          Add VLOCAL TO VGLOB.
C          IF(Any of the VLOCAL vectors are complete) THEN
C             Write complete VGLOB vector to FMS
C             Free VGLOB storage for another vector
C          END IF
C       END DO
C
C       By properly picking the face processing order, we can reduce
C       the number of VGLOB vectors to a reasonable level.
C       The following code determins this level:
C
```

```
C       FORMAL PARAMETERS:
C          (R ) NFACES = Integer scalar
C               Number of faces
C
C          (R ) NEDGES = Integer scalar
C               Number of edges
C
C          (R ) IDEDGE(3,NFACES) = Integer array.
C               Lists the edges associated with a face
C
C          (RW) IORDER(NFACES) = Integer array.
C               On input, specifies the list of starting faces.
C               At least one face must be specified in IORDER(1).
C               The value following the last face specified must be 0.
C               On output, specifies the order for processing faces.
C
C          ( W) MAXVEC = Integer array.
C               Maximum number of active edges.  This determines the
C               number of vectors which must be held in memory to
C               accumulate results.  The objective of this subroutine
C               is to select the IORDER array to minimize this value.
C
C          ( W) MAXUSE(NEDGES) = Integer array.
C               Maximum number of faces using each edge.  For interior
C               edges, the value will be 2.  For edges on the boundary,
C               the value will be 1.
C
C          (  ) NLEFT(NEDGES) = Integer array.
C               Scratch storage used to hold the number of times the
C               edge must be found before it can be eliminated.
C               This vector is initialized to MAXUSE.
C
C          (  ) IUSE(NFACES) = Integer array.
C               Scratch storage used to flag available faces.
C
C          (  ) IAGE(NEDGES) = Integer array.
C               Scratch storage used to hold the age of active edges.
C-----------------------------------------------------------------------
C       Formal Parameters
C-----------------------------------------------------------------------
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='ORDERF')
        INTEGER NFACES
        INTEGER NEDGES
        INTEGER IDEDGE(3,NFACES)
        INTEGER IORDER(NFACES)
        INTEGER MAXVEC
        INTEGER MAXUSE(NEDGES)
        INTEGER IUSE  (NFACES)
        INTEGER IAGE  (NEDGES)
        INTEGER NLEFT (NEDGES)
C-----------------------------------------------------------------------
C       Local Variables
C-----------------------------------------------------------------------
        INTEGER IFACE
        INTEGER JFACE
        INTEGER IEDGE
        INTEGER MYEDGE
        INTEGER MAXAGE
        INTEGER NUMACT
C-----------------------------------------------------------------------
        CALL FMSPSH (MYNAME)
C
C       Initialize starting face:
C       You should pick the face that is the furthest from the center.
```

```
C       Here we will do something simple:
        IORDER(1) = 1
        IORDER(2) = 0
C
C ( 1)  Initialize IUSE to unused:
        DO IFACE = 1,NFACES
           IUSE(IFACE) = 0
        END DO
C
C ( 2)  Find the maximum number of times each edge is used:
        DO IEDGE = 1,NEDGES
           MAXUSE(IEDGE) = 0
        END DO
        DO IFACE = 1,NFACES
           DO I = 1,3
              IEDGE = IDEDGE(I,IFACE)
              MAXUSE(IEDGE) = MAXUSE(IEDGE) + 1
           END DO
        END DO
C
C ( 3)  Initialize the number of times each edge must be found before
C       it can be eliminated.
        DO IEDGE = 1,NEDGES
           NLEFT(IEDGE) = MAXUSE(IEDGE)
        END DO
C
C ( 4)  Initialize the age of each edge to inactive:
        DO IEDGE = 1,NEDGES
           IAGE(IEDGE) = 0
        END DO
C
C ( 5)  Find the face order:
        NUMACT = 0
        MAXVEC = 0
        NSTART = 0
        DO IFACE = 1,NFACES
           IF(NSTART .EQ. 0) THEN
              IF(IORDER(IFACE) .EQ. 0) THEN
C                This is the first non-specified face:
                 NSTART = IFACE - 1
                 IF(NSTART .EQ. 0)
     1              STOP 'ORDERF: No starting faces specified'
              ELSE
C                This face is specified:
                 GO TO 100
              END IF
           END IF
C
C          Find the oldest edge.
           MAXAGE = -1
           MYEDGE = 0
           DO IEDGE = 1,NEDGES
              IF( IAGE(IEDGE) .GT. MAXAGE ) THEN
                 MYEDGE = IEDGE
                 MAXAGE = IAGE(IEDGE)
              END IF
           END DO
           IF(MYEDGE .EQ. 0) STOP 'Oldest edge not found'
C
C          Find a face which uses the oldest edge:
           DO JFACE = 1,NFACES
              IF( IUSE(JFACE) .EQ. 0 ) THEN
                 DO I= 1,3
                    IF(IDEDGE(I,JFACE) .EQ. MYEDGE) THEN
                       IORDER(IFACE) = JFACE
```

```
                        GO TO 100
                     END IF
                  END DO
               END IF
            END DO
            STOP 'Face not found which uses oldest edge'
C
C           The new face is IORDER(IFACE):
  100       CONTINUE
            JFACE = IORDER(IFACE)
C           PRINT *,'ORDERF: IORDER(',IFACE,') =',JFACE
            IUSE(JFACE) = 1
C
C           Age all the active edges:
            DO IEDGE = 1,NEDGES
               IF(IAGE(IEDGE) .GT. 0) THEN
C                 Edge is active:
                  IAGE(IEDGE) = IAGE(IEDGE) + 1
               END IF
            END DO
C
C           Start new edges:
            DO I = 1,3
               IEDGE = IDEDGE(I,JFACE)
               IF(IAGE(IEDGE) .EQ. 0) THEN
C                 New active edge:
                  IAGE(IEDGE) = 1
                  NUMACT = NUMACT + 1
               END IF
            END DO
            IF(NUMACT .GT. MAXVEC) MAXVEC = NUMACT
C           Eliminate edges which are now complete:
            DO I = 1,3
               IEDGE = IDEDGE(I,JFACE)
               NLEFT(IEDGE) = NLEFT(IEDGE) - 1
               IF(NLEFT(IEDGE) .EQ. 0) THEN
C                 Edge is complete:
                  IAGE(IEDGE) = 0
                  NUMACT = NUMACT - 1
               END IF
            END DO
C
C           Do the next face:
         END DO
         CALL FMSPOP (MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE FILL (
     1    NXFACE, NAVAIL,
     2    NFACES, NEDGES, MAXVEC,
     3    LUA   , IDEDGE, IORDER,
     4    MAXUSE, NLEFT ,
     5    VGLOB , IDGLOB, ISTAT)
C=======================================================================
C        DESCRIPTION:
C           This subroutine computes the matrix elements.
C           It is designed to be run in parallel.
C
C        FORMAL PARAMETERS:
C           (RW) NXFACE = Integer scalar (shared, read, write)
C                Outer face loop counter.
C
C           (RW) NAVAIL = Integer scalar (shared, read, write)
```

```
C               Number of global vectors available.
C               (Initialized to MAXVEC)
C
C          (R ) NFACES = Integer scalar (shared, read only)
C               Number of faces
C
C          (R ) NEDGES = Integer scalar (shared, read only)
C               Number of edges
C
C          (R ) MAXVEC = Integer scalar. (shared, read only)
C               Number of vectors in memory
C
C          (R ) LUA(25) = Integer array.
C               Matrix file attributes.
C
C          (R ) IDEDGE(3,NFACES) = Integer array. (shared, read only)
C               Lists the edges associated with a face.
C
C          (R ) IORDER(NFACES) = Integer array (shared, read only)
C               Lists the order for processing faces
C
C          (R ) MAXUSE(NEDGES) = Integer array (shared, read only)
C               Lists the number of times each edge must be used.
C
C          (RW) NLEFT(NEDGES) = Integer array (shared, read, write)
C               Scratch storage, initialized to MAXUSE
C
C          (RW) VGLOB(NEDGES,MAXVEC) = Complex*16 array (shared, r, w)
C               Scratch storage for accumulating vectors
C
C          (RW) IDGLOB(MAXVEC) = Integer array (shared, r, w)
C               Scratch storage for vector numbers in V.
C
C          ( W) ISTAT = Integer scalar (private, write)
C                       Return status.
C                     =-1, Error
C                     >=0, Number of faces processed.
C-----------------------------------------------------------------------
C       Formal Parameters
C-----------------------------------------------------------------------
        CHARACTER*4 MYNAME
        PARAMETER (MYNAME='FILL')
        INTEGER    NXFACE
        INTEGER    NAVAIL
        INTEGER    NFACES
        INTEGER    NEDGES
        INTEGER    IDEDGE(3,NFACES)
        INTEGER    LUA(25)
        INTEGER    MAXVEC
        INTEGER    IORDER(NFACES)
        INTEGER    MAXUSE(NEDGES)
        INTEGER    NLEFT (NEDGES)
        COMPLEX*16 VGLOB(NEDGES, MAXVEC)
        INTEGER    IDGLOB(MAXVEC)
        INTEGER    ISTAT
C       VOLATILE   NXFACE, NAVAIL
C-----------------------------------------------------------------------
C       Local Variables
C-----------------------------------------------------------------------
        INTEGER    IFACE
        INTEGER    JFACE
        INTEGER    ITEMP
        COMPLEX*16 CMD(0:1)
        POINTER (CMD_PTR,CMD)
C-----------------------------------------------------------------------
```

```
            CALL FMSPSH (MYNAME)
            ISTAT = 0
            CALL FMSIGT ('MYNODE', MYNODE)
            CALL FMSIGT ('MEMPTR', CMD_PTR)
            CALL FMSCMG (CMD, L_VLOCAL, 3*NEDGES)
C
C           Get your starting face number:
C           Get exclusive use:
            CALL FMSONE
C              NXFACE = NXFACE + 1
C              MYFACE = NXFACE
               MYFACE = INTINC(NXFACE)
            CALL FMSALL
C           NOTE: We saved the face number NXFACE in a local variable
C           MYFACE, because NXFACE may be changed by another process
C           as soon as we called FMSALL.
C
C           Outer loop over faces:
  100       CONTINUE
               IF(MYFACE .GT. NFACES) THEN
C                 This process is done.
                  CALL FMSCMR (CMD, L_VLOCAL, 3*NEDGES)
            CALL FMSPOP (MYNAME)
                  RETURN
               END IF
               IFACE  = IORDER(MYFACE)
               ISTAT  = ISTAT  + 1
C
C              Inner loop over faces:
C              NOTE: I may process these in any order, because I am doing
C                    all of them and I have room to store the results.
               DO JFACE = 1,NFACES
C
C                 Compute the matrix relating IFACE to JFACE:
C                 There will need to be a lot more information passed
C                 through to this routine.  The following is used to
C                 generate a test matrix only.
C
                  CALL FILL2 (IFACE, JFACE, IDEDGE, MAXUSE, NEDGES,
     1                        CMD(L_VLOCAL))
               END DO
C
C              The 3 vectors VLOCAL are now complete.  They contain
C              1/2 of the terms for interior edges.
C              We now need to accumulate these results into the global
C              arrays VGLOB.
C
C              Critical Section
C              ================
               CALL FMSONE
C
C                 This code writes to shared variables.  Therefore it can
C                 be executed by only one process at a time.
C
C                 Transfer private vectors to global arrays:
                  L_V = L_VLOCAL - NEDGES
                  DO I = 1,3
                     L_V   = L_V + NEDGES
                     IEQN  = IDEDGE(I,IFACE)
C                    Look for vector IEQN in the accumulation list:
                     MYVEC = 0
                     DO IVEC = 1,MAXVEC
                        IF(IEQN .EQ. IDGLOB(IVEC)) THEN
C                          Add to existing vector:
                           MYVEC  = IVEC
```

```
                        CALL CVADD (VGLOB(1,MYVEC), CMD(L_V), NEDGES)
                        GO TO 220
                     END IF
                     IF(IDGLOB(IVEC) .EQ. 0) THEN
C                       This slot is free:
                        IF(MYVEC .EQ. 0) MYVEC = IVEC
                     END IF
                  END DO
                  IF(MYVEC .EQ. 0) THEN
C                    Error:
                     WRITE(6,2000) MYNODE, IDEDGE(I,IFACE),
     1                  (II, IDGLOB(II),II=1,MAXVEC)
                     ISTAT = -1
                     CALL FMSCMR (CMD, L_VLOCAL, 3*NEDGES)
         CALL FMSPOP (MYNAME)
                     RETURN
                  END IF
C                 New vector:
                  ITEMP = INTDEC(NAVAIL)
                  IDGLOB(MYVEC) = IEQN
                  CALL CVMOVE (VGLOB(1,MYVEC), CMD(L_V), NEDGES)
   220            CONTINUE
                  NLEFT(IEQN) = NLEFT(IEQN) - 1
                  IF(NLEFT(IEQN) .EQ. 0) THEN
C                    Vector is ready to be written out:
                     CALL FMSROW (IEQN, VGLOB(1,MYVEC), LUA)
C                    Free the storage:
                     IDGLOB(MYVEC) = 0
                     ITEMP  = INTINC(NAVAIL)
                  END IF
               END DO
C
C              Get your next face number:
C              NXFACE = NXFACE + 1
C              MYFACE = NXFACE
               MYFACE = INTINC(NXFACE)
            CALL FMSALL
C
C           End of Critical Section
C           ======================
C
C           Do the next face:
         GO TO 100
 2000    FORMAT (/
     1   ' FILL: No global vectors available for process',I5/
     2   '       Trying to store vector',I10/
     3   '       Vector     Contents'/
     4   '       ======     ========'/
     5   (I13,I13))
         END
C=======================================================================
         SUBROUTINE FILL2 (IFACE, JFACE, IDEDGE, MAXUSE, NUMEQ, VLOCAL)
C=======================================================================
C        This is a dummy subroutine to generate test matrix elements.
         INTEGER IFACE, JFACE, IDEDGE(3,*), NUMEQ, MAXUSE(NUMEQ)
         REAL*8 TEMP, DTEMP
         COMPLEX*16 ZIJ
         COMPLEX*16 VLOCAL(NUMEQ,3)
         CHARACTER*5 MYNAME
         PARAMETER (MYNAME='FILL2')
         CALL FMSPSH (MYNAME)
         DO I=1,3
            IGLOB = IDEDGE(I,IFACE)
            TEMP  = (1.0D0)/(DFLOAT(MAXUSE(IGLOB)))
```

```
            DO J=1,3
               JGLOB = IDEDGE(J,JFACE)
               IF( IGLOB .EQ. JGLOB ) THEN
C                 This is a diagonal element:
                  IF ( IGLOB .EQ. 1) THEN
                     DTEMP = DFLOAT(NUMEQ)*TEMP
                     ZIJ = DCMPLX(DTEMP,DTEMP)
                  ELSE
                     ZIJ = DCMPLX(TEMP,TEMP)
                  END IF
               ELSE IF( (IGLOB .EQ. 1)  .OR. (JGLOB .EQ. 1) ) THEN
C                 Term is in first row or column:
                  ZIJ = DCMPLX(-TEMP,-TEMP)
               ELSE
C                 Interior element:
                  ZIJ = DCMPLX(0.0D0,0.0D0)
               END IF
               VLOCAL(JGLOB,I) = ZIJ
            END DO
         END DO
         CALL FMSPOP (MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE MGEN (NEQEST, NFACES, NEDGES, IDEDGE)
C=======================================================================
         INTEGER NEQEST
         INTEGER NFACES
         INTEGER NEDGES
         INTEGER IDEDGE(3,NEQEST)
         REAL*8  RN
         CHARACTER*4 MYNAME
         PARAMETER (MYNAME='MGEN')
         CALL FMSPSH (MYNAME)
C        Generate a mesh of quadrilaterals that is
C        approximately square.  Each quadrilateral will
C        be divided into four triangles.  Each interior
C        quadrilateral will add 6 equations (4 interior
C        edges and 2 sides).
C        Compute the number of quadrilaterals:
         RN = NEQEST/6
C        Compute the number of quadrilaterals on an edge:
         RN = SQRT(RN)
         N  = RN
         N2 = N  + N
         N3 = N2 + N
         N4 = N3 + N
         N6 = N4 + N2
         M  = (NEQEST + N2)/N6
C        PRINT *,'MGEN: Mesh is ',N,' by ',M
         NFACES = 0
         NEDGES = 0
C        Loop down the cylinder:
         DO J = 1,M
C           Loop around the cylinder:
            DO I = 1,N
               I2 = I + I
C              Generate the 4 face numbers in the square:
               IFACE = NFACES + I
               JFACE = NFACES + N + I2 - 1
               KFACE = JFACE  + 1
               LFACE = NFACES + N3 + I
C              Generate the edge list for each face:
               IDEDGE(1,IFACE) = NEDGES + I
               IDEDGE(2,IFACE) = NEDGES + N + I2
```

```
               IDEDGE(3,IFACE) = IDEDGE(2,IFACE) - 1
               IDEDGE(1,JFACE) = NEDGES + N3 + I
               IDEDGE(2,JFACE) = IDEDGE(3,IFACE)
               IDEDGE(3,JFACE) = NEDGES + N4 + I2 - 1
               IDEDGE(1,KFACE) = IDEDGE(2,IFACE)
               IDEDGE(2,KFACE) = IDEDGE(1,JFACE) + 1
               IDEDGE(3,KFACE) = IDEDGE(3,JFACE) + 1
               IDEDGE(1,LFACE) = IDEDGE(3,JFACE)
               IDEDGE(2,LFACE) = IDEDGE(3,KFACE)
               IDEDGE(3,LFACE) = NEDGES + N6 + I
            END DO
C           Wrap for a cylinder:
            IDEDGE(2,KFACE) = IDEDGE(2,KFACE) - N
            NFACES = NFACES + N4
            NEDGES = NEDGES + N6
         END DO
C        Leave cylinder open:
         NEDGES = NEDGES + N
C
C        WRITE(6,5000) (I,(IDEDGE(J,I), J=1,3), I=1,NFACES)
         CALL FMSPOP (MYNAME)
         RETURN
C5000    FORMAT(/
C     1  ' FACE   EDGE 1   EDGE 2   EDGE 3'/
C     2  ' ====   ======   ======   ======'/
C     3  (I5,3I9))
         END
C=======================================================================
         SUBROUTINE CVMOVE (A, B, N)
C=======================================================================
         COMPLEX*16 A(N), B(N)
         DO I = 1,N
            A(I) = B(I)
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE CVADD (A, B, N)
C=======================================================================
         COMPLEX*16 A(N), B(N)
         DO I = 1,N
            A(I) = A(I) + B(I)
         END DO
         RETURN
         END
C=======================================================================
         INTEGER FUNCTION INTINC (I)
C=======================================================================
C        This function increments a volatile shared variable.  It is
C        placed in a subroutine to prevent some compilers from storing
C        the value in a register and not updating it.
         INTEGER I
         I = I + 1
         INTINC = I
         RETURN
         END
C=======================================================================
         INTEGER FUNCTION INTDEC (I)
C=======================================================================
C        This function decrements a volatile shared variable.  It is
C        placed in a subroutine to prevent some compilers from storing
C        the value in a register and not updating it.
         INTEGER I
         I = I - 1
         INTDEC = I
```

```
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 10. COMPLEX SYMMETRIC OUT-OF-CORE MATRIX

This example solves a complex symmetric system of equations where the number of equations and right-hand side vectors are input. Subroutines CSUBLK is used to define matrix coefficients in the lower triangle and diagonal. The matrix size is only limited by the available disk space.

```
C       E X A M P L E   10
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_10')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
```

```
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       Local variables:
        INTEGER     N, NRHS, I, LX, LENX, LDISK, NV
        REAL*8      EI, ERROR
C
C       Common block to communicate with CSUBLK:
        COMMON /MYDATA/NUMEQ
        DATA        LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
        CALL FMSIST ('MDATAU', 2)
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        N    = ASK_I ('Enter the number of equations')
        NRHS = ASK_I ('Enter the number of solution vectors')
C
C (2)   Open FMS files:
        CALL CSDI (LOWEQ, N, 'LUF', LUF)
        CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C       Populate test vector:
C       Allocate enough space to hold a vector image on disk:
        LENX = LUX(4)/2
        CALL FMSCMG (CMD, LX, LENX)
        NUMEQ  = N
        DO 10 I = 1,LENX
           CMD(LX-1+I) = (0.0D0,0.0D0)
   10   CONTINUE
        CMD(LX) = (1.0D0,1.0D0)
C
C (3)   Write data to FMS files:
        LDISK  = 1
        DO 30 NV = 1,NRHS
```

```
            CALL FMSWRT (LUX(1), LDISK, CMD(LX), LUX(4))
            LDISK = LDISK + LUX(4)
   30    CONTINUE
C
C (4)    Perform matrix algebra:
         CALL CSDF (LUF, LUF, LUX, LUX,  NUMRED)
         CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
         LDISK  = 1
         DO 60 NV = 1,NRHS
            CALL FMSRED (LUX(1), LDISK, CMD(LX), IDTYPE*N)
            LDISK  = LDISK  + LUX(4)
            DO 50 I = 1,N
               EI = ABS(CMD(LX-1+I) - 1.0D0)
               IF(EI .GT. ERROR) ERROR = EI
   50       CONTINUE
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUF)
         CALL FMSCV (LUX)
         CALL FMSCMR (CMD, LX, LENX)
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         COMPLEX*16 A(0:*), D(*), ONE
         PARAMETER  (ONE=(1.0D0,1.0D0))
         COMMON /MYDATA/NUMEQ
C
```

```
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = ONE
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(NUMEQ,NUMEQ)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = -ONE
   20   CONTINUE
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

Copyright © Multipath Corporation

```
C       E X A M P L E   10
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_10')
C
C       Input data functions:
        INTEGER ASK_I
C
C       Data type = complex:
        PARAMETER (IDTYPE = 2)
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       Local variables:
        INTEGER     N, NRHS, I, LX, LENX, LDISK, NV
        REAL*8      EI, ERROR
C
C       Common block to communicate with CSUBLK:
        COMMON /MYDATA/NUMEQ
        DATA        LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('MDATAU', 2)
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        N    = ASK_I ('Enter the number of equations')
        NRHS = ASK_I ('Enter the number of solution vectors')
C
C (2)   Open FMS files:
        CALL CSDI (LOWEQ, N, 'LUF', LUF)
        CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C       Populate test vector:
C       Allocate enough space to hold a vector image on disk:
        LENX = LUX(4)/2
        CALL FMSCMG (CMD, LX, LENX)
        NUMEQ  = N
        DO 10 I = 1,LENX
           CMD(LX-1+I) = (0.0D0,0.0D0)
   10   CONTINUE
```

```
        CMD(LX) = (1.0D0,1.0D0)
C
C (3)   Write data to FMS files:
        LDISK  = 1
        DO 30 NV = 1,NRHS
           CALL FMSWRT (LUX(1), LDISK, CMD(LX), LUX(4))
           LDISK = LDISK + LUX(4)
   30   CONTINUE
C
C (4)   Perform matrix algebra:
        CALL CSDF (LUF, LUF, LUX, LUX,  NUMRED)
        CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Check the answer:
        ERROR = 0.0D0
        LDISK  = 1
        DO 60 NV = 1,NRHS
           CALL FMSRED (LUX(1), LDISK, CMD(LX), LUX(4))
           LDISK  = LDISK  + LUX(4)
           DO 50 I = 1,N
              EI = ABS(CMD(LX-1+I) - 1.0D0)
              IF(EI .GT. ERROR) ERROR = EI
   50      CONTINUE
   60   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUF)
        CALL FMSCV (LUX)
        CALL FMSCMR (CMD, LX, LENX)
        CALL FMSPOP (MYNAME)
        CALL FMSEND
        END
C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), ONE
        PARAMETER  (ONE=(1.0D0,1.0D0))
        COMMON /MYDATA/NUMEQ
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = ONE
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(NUMEQ,NUMEQ)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = -ONE
   20   CONTINUE
        RETURN
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
```

```
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 11. GENERAL OUT-OF-CORE MATRIX

This example solves a system of equations where the following may be specified as input:

- ***FMS*** Module

  RS = Real Symmetric

  RN = Real Nonsymmetric

  CH = Complex Hermitian

  CS = Complex Symmetric

  CN = Complex Nonsymmetric

- Number of equations
- Number of solution vectors
- Any FMS Parameter

This example is designed to support pre-sales benchmarking of machines.

```
C       E X A M P L E   11
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_11')
C
C       Subroutines called in parallel:
        EXTERNAL WRITEV
C
C       Input data functions:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
```

```
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       FMS Parameter values:
        INTEGER MAXCPU
        INTEGER LOGTIM
C
C       Local variables:
        INTEGER     IDTYPE, I, LX, LENX, LDISK, NV
        REAL*8      EI, ERROR
C
C       Common block to communicate with fill routines:
        COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
        COMMON /SHARE/IWORK
        DATA LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
C       Default to generating iwatch html pages:
        CALL FMSIST ('IWATCH', 99)
        CALL FMSPSH(MYNAME)
        CALL FMSIST ('MDATAU', 2)
        CALL FMSIGT ('LOGTIM', LOGTIM)
        IF(LOGTIM .LT. 3) CALL FMSIST ('LOGTIM',    3)
        CALL FMSIGT ('IPRF', IPRF)
        IF(IAND(IPRF,   2) .EQ. 0) IPRF = IPRF + 2
```

```
        IF(IAND(IPRF,1024) .EQ. 0) IPRF = IPRF + 1024
        CALL FMSIST ('IPRF'  , IPRF)
C       Default to using direct I/O:
        CALL FMSIST ('IOTYPE', 2)
    1   CONTINUE
        WRITE (6,*) 'Enter the FMS module number (1 to 5)'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        READ  (5,*) MOD
        IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
        N    = ASK_I('Enter the number of equations')
        NRHS = ASK_I('Enter the number of solution vectors')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        CALL FMSIGT ('MAXCPU', MAXCPU)
        IF(MOD .LE. 2) THEN
           IDTYPE = 1
        ELSE
           IDTYPE = 2
        END IF
        NUMEQ  = N
C
C       Allocate processor local workspace:
        LENLOC = NUMEQ
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRLG (RMD, LCPU1, LENLOC)
        ELSE
           CALL FMSCLG (CMD, LCPU1, LENLOC)
        END IF
C
C (2)   Open FMS files:
        IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUF', LUF)
```

```
      IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C (3)   Write data to FMS files:
C
C       Fill the RHS vectors in parallel:
      WRITE (6,2000)
      IWORK = 0
      DO ICPU = 2,MAXCPU
         CALL FMSPAR (1, WRITEV, LUX)
      END DO
      IF(MAXCPU .GT. 1) CALL FMSRUN
      CALL WRITEV(LUX)
      IF(MAXCPU .GT. 1) CALL FMSYNC
C
C       Return processor local workspace:
      IF(IDTYPE .EQ. 1) THEN
         CALL FMSRLR (RMD, LCPU1, LENLOC)
      ELSE
         CALL FMSCLR (RMD, LCPU1, LENLOC)
      END IF
C
C (4)   Perform matrix algebra:
      IF(MOD.EQ.1) CALL RSDF (LUF, LUF, LUX, LUX,  NUMRED)
      IF(MOD.EQ.2) CALL RNDF (LUF, LUF, LUX, LUX,  NUMRED)
      IF(MOD.EQ.3) CALL CHDF (LUF, LUF, LUX, LUX,  NUMRED)
      IF(MOD.EQ.4) CALL CSDF (LUF, LUF, LUX, LUX,  NUMRED)
      IF(MOD.EQ.5) CALL CNDF (LUF, LUF, LUX, LUX,  NUMRED)
C
      CALL FMSIGT ('LOGTIM', LOGTIM)
      IF(LOGTIM .GE. 3) CALL FMSCST ('SHOW', 'CALLS')
      IF(MOD.EQ.1) CALL RSDS (LUF, LUX, LUX, NRHS, ISKIP)
      IF(MOD.EQ.2) CALL RNDS (LUF, LUX, LUX, NRHS, ISKIP)
      IF(MOD.EQ.3) CALL CHDS (LUF, LUX, LUX, NRHS, ISKIP)
      IF(MOD.EQ.4) CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
      IF(MOD.EQ.5) CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
```

```
C
C (5)   Read data from FMS files:
C       Allocate a work array for reading the results:
        LENVEC = LUX(4)
        LDX    = LENVEC/IDTYPE
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMG (RMD, LX, LDX)
        ELSE
           CALL FMSCMG (CMD, LX, LDX)
        END IF
C       Check the answer:
        ERROR = 0.0D0
        LDISK  = 1
        DO 60 NV = 1,NRHS
           IF(IDTYPE .EQ. 1) THEN
              CALL FMSRED (LUX(1), LDISK, RMD(LX), LENVEC)
              DO 50 I = 1,N
                 EI = ABS(RMD(LX-1+I) - 1.0D0)
                 IF(EI .GT. ERROR) ERROR = EI
   50         CONTINUE
           ELSE
              CALL FMSRED (LUX(1), LDISK, CMD(LX), LENVEC)
              DO 51 I = 1,N
                 EI = ABS(CMD(LX-1+I) - 1.0D0)
                 IF(EI .GT. ERROR) ERROR = EI
   51         CONTINUE
           END IF
           LDISK  = LDISK  + LENVEC
   60   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)   Close FMS files:
C       Return the work array:
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMR (RMD, LX, LDX)
        ELSE
           CALL FMSCMR (CMD, LX, LDX)
        END IF
        CALL FMSCM (LUF)
```

```
         CALL FMSCV (LUX)
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP(MYNAME)
         CALL FMSEND
 2000    FORMAT (/
     1   ' Writing the RHS vectors in parallel'/
     2   ' ==================================')
         END

C        ================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*), DIA, OFFDIA
         PARAMETER  (DIA   = 1.0D0)
         PARAMETER  (OFFDIA=-1.0D0)
         COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
C
C        Populate the diagonal with test data:
         IF(IROW2 .EQ. JCOL2) THEN
C           This is a diagonal block:
            DO 10 I = IROW1,IROW2
               D(I) = DIA
   10       CONTINUE
            IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
         END IF
C
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
         DO 20 I = IROW1,IROW2
            J = LOWEQ(I)
            IF( (J .GE. JCOL1) .AND.
     1          (J .LE. JCOL2) .AND.
     2          (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20    CONTINUE
         RETURN
         END
```

```
C       ================================================================
        SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), OFFDIA
        PARAMETER  (OFFDIA=-1.0D0)
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 20 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
   20   CONTINUE
        RETURN
        END

C       ================================================================
        SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     D(*), DIA
        COMPLEX*16 A(0:*), OFFDIA
        PARAMETER  (DIA   =  1.0D0        )
        PARAMETER  (OFFDIA=(-1.0D0,0.0D0))
        COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
```

```
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20   CONTINUE
        RETURN
        END

C       ================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), DIA, OFFDIA
        PARAMETER  (DIA   =( 1.0D0, 1.0D0))
        PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
        COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
```

```
            IF( (J .GE. JCOL1) .AND.
     1          (J .LE. JCOL2) .AND.
     2          (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20    CONTINUE
         RETURN
         END

C        ================================================================
         SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         COMPLEX*16 A(0:*), D(*), OFFDIA
         PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
         DO 20 J = JCOL1,JCOL2
            I = LOWEQ(J)
            IF( (I .GE. IROW1) .AND.
     1          (I .LE. IROW2) .AND.
     2          (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
   20    CONTINUE
         RETURN
         END

C        ================================================================
         SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         REAL*8     A(0:*), DIA, OFFDIA
         PARAMETER  (DIA   = 1.0D0)
         PARAMETER  (OFFDIA=-1.0D0)
C
C        Fill first row and diagonal:
         LU1    = LUFLAG(1)
         LI1    = LOCI(1)
```

```
      DO 10 J = JEQN1,JEQN2
         LIJ = LI1 + LOCJ(J,LU1)
         A(LIJ) = OFFDIA
         LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
         A(LD ) = DIA
10    CONTINUE
C
C     Fill column 1 if this is the first slab:
      IF(JEQN1 .EQ. 1) THEN
         DO 20 I = 1,NUMEQ
            LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
            A(LIJ) = OFFDIA
20       CONTINUE
         LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
         A(LD) = DFLOAT(NUMEQ)
      END IF
      RETURN
      END

C     ================================================================
      SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C     ================================================================
      INTEGER    JEQN1, JEQN2, NUMEQ
      INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
      COMPLEX*16 A(0:*), DIA, OFFDIA
      PARAMETER  (DIA   =( 1.0D0, 1.0D0))
      PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C     Fill first row and diagonal:
      LU1    = LUFLAG(1)
      LI1    = LOCI(1)
      DO 10 J = JEQN1,JEQN2
         LIJ = LI1 + LOCJ(J,LU1)
         A(LIJ) = OFFDIA
         LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
         A(LD ) = DIA
10    CONTINUE
C
C     Fill column 1 if this is the first slab:
```

```
         IF(JEQN1 .EQ. 1) THEN
            DO 20 I = 1,NUMEQ
               LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
               A(LIJ) = OFFDIA
   20       CONTINUE
            LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
            A(LD) = DCMPLX(NUMEQ,NUMEQ)
         END IF
         RETURN
         END
C=======================================================================
         SUBROUTINE WRITEV (LUX)
C=======================================================================
C
C        DESCRIPTION:
C           This subroutine runs in parallel. For each vector
C           that it finds, it fills in the terms and calls FMSWRT to
C           write the vector to the vector file LUX.
C
C        FORMAL PARAMETERS:
C           (R ) LUX(25) = FMS vector file attributes
C-----------------------------------------------------------------------
C
C        Formal Parameters:
         INTEGER    LUX(25)
C
C        Local Variables:
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='WRITEV')
         INTEGER    MYNODE
         INTEGER    LUPR
         INTEGER    NUMEQ
         INTEGER    NUMVEC
         INTEGER    NROWS, NCOLS
         INTEGER    IEQN1, IEQN2, JEQN1, JEQN2
         INTEGER    MY_WORK
         INTEGER    MY_TOTAL
         INTEGER    LX
         REAL*8     RMD(0:1)
```

```
      COMPLEX*16 CMD(0:1)
      POINTER    (RMD_PTR,RMD)
      POINTER    (CMD_PTR,CMD)
      COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
      COMMON/SHARE/IWORK
      INTEGER    PARENT
      PARAMETER (PARENT = 0)
C
      CALL FMSPSH(MYNAME)
C     Get your CPU number (0 to MAXCPU-1)
      CALL FMSIGT ('MYNODE', MYNODE)
      CALL FMSIGT ('LUPR',   LUPR)
      NUMEQ  = LUX( 3)
      LENVEC = LUX( 4)
      IDTYPE = LUX( 5)
      NUMVEC = LUX( 6)
      NROWS  = NUMEQ
      NCOLS  = 1
      IEQN1  = 1
      IEQN2  = NUMEQ
      LDX    = LENVEC/IDTYPE
C
C     Use the processor local storage for the data you generate:
C     For this example, we will fill [B] vector by vector.
      LX = LCPU1 + LENLOC*MYNODE
      IF(IDTYPE .EQ. 1) THEN
         CALL FMSIGT ('MEMPTR', RMD_PTR)
         DO I=1,NUMEQ
            RMD(LX+I-1) = 0.0D0
         END DO
         RMD(LX) = 1.0D0
      ELSE
         CALL FMSIGT ('MEMPTR', CMD_PTR)
         DO I=1,NUMEQ
            CMD(LX+I-1) = (0.0D0,0.0D0)
         END DO
         IF(MOD .EQ. 3) THEN
            CMD(LX) = (1.0D0,0.0D0)
         ELSE
```

```
               CMD(LX) = (1.0D0,1.0D0)
            END IF
         END IF
C
C        Find the next vector to fill:
         MY_TOTAL = 0
   10    CONTINUE
            CALL FMSONE
C              IWORK   = IWORK + 1
C              MY_WORK = IWORK
               MY_WORK = INTINC(IWORK)
            CALL FMSALL
            IF(MY_WORK .GT. NUMVEC) GO TO 100
            MY_TOTAL = MY_TOTAL + 1
            LDISK  = 1 + LENVEC*(MY_WORK-1)
C
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSWRT (LUX, LDISK, RMD(LX), LENVEC)
            ELSE
               CALL FMSWRT (LUX, LDISK, CMD(LX), LENVEC)
            END IF
         GO TO 10
  100    CONTINUE
C        Your part of the work is done:
C
C        Report your total work:
         IF(MY_TOTAL .GT. 0) THEN
            CALL FMSONE
            WRITE(LUPR,2000) MYNODE, MY_TOTAL
            CALL FMSALL
         END IF
         CALL FMSPOP(MYNAME)
         RETURN
 2000    FORMAT (' Process',I5,' computed',I5,' vectors.')
 2001    FORMAT (' Process',I5,' is computing vector ',I5)
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
```

```
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        INTEGER FUNCTION INTINC (I)
C-----------------------------------------------------------------------
C       This function increments a volatile shared variable.  It is
C       placed in a subroutine to prevent some compilers from storing
C       the value in a register and not updating it.
        INTEGER I
        I = I + 1
        INTINC = I
        RETURN
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   11
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_11')
C
C       Subroutines called in parallel:
        EXTERNAL WRITEV
C
C       Input data functions:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       FMS Parameter values:
        INTEGER MAXCPU
        INTEGER LOGTIM
C
C       Local variables:
        INTEGER     IDTYPE, I, LX, LENX, LDISK, NV
        REAL*8      EI, ERROR
C
C       Common block to communicate with fill routines:
        COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
        COMMON /SHARE/IWORK
        DATA LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
C       Default to generating iwatch html pages:
        CALL FMSIST ('IWATCH'  , 99)
        CALL FMSPSH(MYNAME)
        CALL FMSIST ('MDATAU', 2)
        CALL FMSIGT ('LOGTIM', LOGTIM)
        IF(LOGTIM .LT. 3) CALL FMSIST ('LOGTIM',    3)
        CALL FMSIGT ('IPRF', IPRF)
        IF(IAND(IPRF,   2) .EQ. 0) IPRF = IPRF + 2
        IF(IAND(IPRF,1024) .EQ. 0) IPRF = IPRF + 1024
        CALL FMSIST ('IPRF'  , IPRF)
C       Default to using direct I/O:
        CALL FMSIST ('IOTYPE'  , 2)
    1   CONTINUE
        WRITE (6,*) 'Enter the FMS module number (1 to 5)'
```

```
         WRITE (6,*) '   1 = Real Symmetric'
         WRITE (6,*) '   2 = Real Nonsymmetric'
         WRITE (6,*) '   3 = Complex Hermitian'
         WRITE (6,*) '   4 = Complex Symmetric'
         WRITE (6,*) '   5 = Complex Nonsymmetric'
         READ  (5,*) MOD
         IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
         N    = ASK_I('Enter the number of equations')
         NRHS = ASK_I('Enter the number of solution vectors')
         WRITE (6,*) 'You may now alter any FMS parameter.'
         WRITE (6,*) 'When you are finished, type the letters RETURN'
         CALL FMSSET
         CALL FMSIGT ('MEMPTR', IMD_PTR)
         CALL FMSIGT ('MEMPTR', RMD_PTR)
         CALL FMSIGT ('MEMPTR', CMD_PTR)
         CALL FMSIGT ('MAXCPU', MAXCPU)
         IF(MOD .LE. 2) THEN
            IDTYPE = 1
         ELSE
            IDTYPE = 2
         END IF
         NUMEQ  = N
C
C        Allocate processor local workspace:
         LENLOC = NUMEQ
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRLG (RMD, LCPU1, LENLOC)
         ELSE
            CALL FMSCLG (CMD, LCPU1, LENLOC)
         END IF
C
C (2)    Open FMS files:
         IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C (3)    Write data to FMS files:
C
C        Fill the RHS vectors in parallel:
         WRITE (6,2000)
         IWORK = 0
         DO ICPU = 2,MAXCPU
            CALL FMSPAR (1, WRITEV, LUX)
         END DO
         IF(MAXCPU .GT. 1) CALL FMSRUN
         CALL WRITEV(LUX)
         IF(MAXCPU .GT. 1) CALL FMSYNC
C
C        Return processor local workspace:
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRLR (RMD, LCPU1, LENLOC)
         ELSE
            CALL FMSCLR (CMD, LCPU1, LENLOC)
         END IF
C
C (4)    Perform matrix algebra:
         IF(MOD.EQ.1) CALL RSDF (LUF, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.2) CALL RNDF (LUF, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.3) CALL CHDF (LUF, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.4) CALL CSDF (LUF, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.5) CALL CNDF (LUF, LUF, LUX, LUX,  NUMRED)
C
```

```
         CALL FMSIGT ('LOGTIM', LOGTIM)
         IF(LOGTIM .GE. 3) CALL FMSCST ('SHOW', 'CALLS')
         IF(MOD.EQ.1) CALL RSDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.3) CALL CHDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.4) CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C (5)    Read data from FMS files:
C        Allocate a work array for reading the results:
         LENVEC = LUX(4)
         LDX    = LENVEC/IDTYPE
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMG (RMD, LX, LDX)
         ELSE
            CALL FMSCMG (CMD, LX, LDX)
         END IF
C        Check the answer:
         ERROR = 0.0D0
         LDISK  = 1
         DO 60 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRED (LUX(1), LDISK, RMD(LX), LENVEC)
               DO 50 I = 1,N
                  EI = ABS(RMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   50          CONTINUE
            ELSE
               CALL FMSRED (LUX(1), LDISK, CMD(LX), LENVEC)
               DO 51 I = 1,N
                  EI = ABS(CMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   51          CONTINUE
            END IF
            LDISK  = LDISK  + LENVEC
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
C        Return the work array:
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LX, LDX)
         ELSE
            CALL FMSCMR (CMD, LX, LDX)
         END IF
         CALL FMSCM (LUF)
         CALL FMSCV (LUX)
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP(MYNAME)
         CALL FMSEND
 2000    FORMAT (/
     1   ' Writing the RHS vectors in parallel'/
     2   ' ===================================')
         END

C        ================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*), DIA, OFFDIA
         PARAMETER  (DIA   = 1.0D0)
         PARAMETER  (OFFDIA=-1.0D0)
         COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
```

```
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20   CONTINUE
        RETURN
        END

C       ================================================================
        SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), OFFDIA
        PARAMETER  (OFFDIA=-1.0D0)
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 20 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
   20   CONTINUE
        RETURN
        END

C       ================================================================
        SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     D(*), DIA
        COMPLEX*16 A(0:*), OFFDIA
        PARAMETER  (DIA   =  1.0D0        )
        PARAMETER  (OFFDIA=(-1.0D0,0.0D0))
        COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
```

```
      DO 20 I = IROW1,IROW2
         J = LOWEQ(I)
         IF( (J .GE. JCOL1) .AND.
     1       (J .LE. JCOL2) .AND.
     2       (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20 CONTINUE
      RETURN
      END

C     ================================================================
      SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                   JCOL1, JCOL2, IJSTEP)
C     ================================================================
      INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
      INTEGER    LOWEQ(*), LOCEQ(*)
      COMPLEX*16 A(0:*), D(*), DIA, OFFDIA
      PARAMETER  (DIA   =( 1.0D0, 1.0D0))
      PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
      COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
C
C     Populate the diagonal with test data:
      IF(IROW2 .EQ. JCOL2) THEN
C        This is a diagonal block:
         DO 10 I = IROW1,IROW2
            D(I) = DIA
   10    CONTINUE
         IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
      END IF
C
C     Populate profile of [AL] with test data:
C     The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
      DO 20 I = IROW1,IROW2
         J = LOWEQ(I)
         IF( (J .GE. JCOL1) .AND.
     1       (J .LE. JCOL2) .AND.
     2       (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20 CONTINUE
      RETURN
      END

C     ================================================================
      SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                   JCOL1, JCOL2, IJSTEP)
C     ================================================================
      INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
      INTEGER    LOWEQ(*), LOCEQ(*)
      COMPLEX*16 A(0:*), D(*), OFFDIA
      PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C     Populate profile of [AL] with test data:
C     The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
      DO 20 J = JCOL1,JCOL2
         I = LOWEQ(J)
         IF( (I .GE. IROW1) .AND.
     1       (I .LE. IROW2) .AND.
     2       (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
   20 CONTINUE
      RETURN
      END

C     ================================================================
      SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C     ================================================================
      INTEGER    JEQN1, JEQN2, NUMEQ
      INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
```

```
      REAL*8     A(0:*), DIA, OFFDIA
      PARAMETER  (DIA   = 1.0D0)
      PARAMETER  (OFFDIA=-1.0D0)
C
C     Fill first row and diagonal:
      LU1    = LUFLAG(1)
      LI1    = LOCI(1)
      DO 10 J = JEQN1,JEQN2
         LIJ = LI1 + LOCJ(J,LU1)
         A(LIJ) = OFFDIA
         LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
         A(LD ) = DIA
10    CONTINUE
C
C     Fill column 1 if this is the first slab:
      IF(JEQN1 .EQ. 1) THEN
         DO 20 I = 1,NUMEQ
            LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
            A(LIJ) = OFFDIA
20       CONTINUE
         LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
         A(LD) = DFLOAT(NUMEQ)
      END IF
      RETURN
      END

C     ================================================================
      SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C     ================================================================
      INTEGER    JEQN1, JEQN2, NUMEQ
      INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
      COMPLEX*16 A(0:*), DIA, OFFDIA
      PARAMETER  (DIA   =( 1.0D0, 1.0D0))
      PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C     Fill first row and diagonal:
      LU1    = LUFLAG(1)
      LI1    = LOCI(1)
      DO 10 J = JEQN1,JEQN2
         LIJ = LI1 + LOCJ(J,LU1)
         A(LIJ) = OFFDIA
         LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
         A(LD ) = DIA
10    CONTINUE
C
C     Fill column 1 if this is the first slab:
      IF(JEQN1 .EQ. 1) THEN
         DO 20 I = 1,NUMEQ
            LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
            A(LIJ) = OFFDIA
20       CONTINUE
         LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
         A(LD) = DCMPLX(NUMEQ,NUMEQ)
      END IF
      RETURN
      END
C=======================================================================
      SUBROUTINE WRITEV (LUX)
C=======================================================================
C
C     DESCRIPTION:
C        This subroutine runs in parallel. For each vector
C        that it finds, it fills in the terms and calls FMSWRT to
C        write the vector to the vector file LUX.
C
```

```
C       FORMAL PARAMETERS:
C          (R ) LUX(25) = FMS vector file attributes
C-----------------------------------------------------------------------
C
C       Formal Parameters:
        INTEGER    LUX(25)
C
C       Local Variables:
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='WRITEV')
        INTEGER    MYNODE
        INTEGER    LUPR
        INTEGER    NUMEQ
        INTEGER    NUMVEC
        INTEGER    NROWS, NCOLS
        INTEGER    IEQN1, IEQN2, JEQN1, JEQN2
        INTEGER    MY_WORK
        INTEGER    MY_TOTAL
        INTEGER    LX
        REAL*8     RMD(0:1)
        COMPLEX*16 CMD(0:1)
        POINTER    (RMD_PTR,RMD)
        POINTER    (CMD_PTR,CMD)
        COMMON /MYDATA/MOD, N, NRHS, LCPU1, LENLOC
        COMMON/SHARE/IWORK
        INTEGER    PARENT
        PARAMETER (PARENT = 0)
C
        CALL FMSPSH(MYNAME)
C       Get your CPU number (0 to MAXCPU-1)
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('LUPR',   LUPR  )
        NUMEQ  = LUX( 3)
        LENVEC = LUX( 4)
        IDTYPE = LUX( 5)
        NUMVEC = LUX( 6)
        NROWS  = NUMEQ
        NCOLS  = 1
        IEQN1  = 1
        IEQN2  = NUMEQ
        LDX    = LENVEC/IDTYPE
C
C       Use the processor local storage for the data you generate:
C       For this example, we will fill [B] vector by vector.
        LX = LCPU1 + LENLOC*MYNODE
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSIGT ('MEMPTR', RMD_PTR)
           DO I=1,NUMEQ
              RMD(LX+I-1) = 0.0D0
           END DO
           RMD(LX) = 1.0D0
        ELSE
           CALL FMSIGT ('MEMPTR', CMD_PTR)
           DO I=1,NUMEQ
              CMD(LX+I-1) = (0.0D0,0.0D0)
           END DO
           IF(MOD .EQ. 3) THEN
              CMD(LX) = (1.0D0,0.0D0)
           ELSE
              CMD(LX) = (1.0D0,1.0D0)
           END IF
        END IF
C
C       Find the next vector to fill:
```

```
         MY_TOTAL = 0
    10   CONTINUE
            CALL FMSONE
C              IWORK   = IWORK + 1
C              MY_WORK = IWORK
               MY_WORK = INTINC(IWORK)
            CALL FMSALL
            IF(MY_WORK .GT. NUMVEC) GO TO 100
            MY_TOTAL = MY_TOTAL + 1
            LDISK  = 1 + LENVEC*(MY_WORK-1)
C
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSWRT (LUX, LDISK, RMD(LX), LENVEC)
            ELSE
               CALL FMSWRT (LUX, LDISK, CMD(LX), LENVEC)
            END IF
         GO TO 10
   100   CONTINUE
C        Your part of the work is done:
C
C        Report your total work:
         IF(MY_TOTAL .GT. 0) THEN
            CALL FMSONE
            WRITE(LUPR,2000) MYNODE, MY_TOTAL
            CALL FMSALL
         END IF
         CALL FMSPOP(MYNAME)
         RETURN
 2000    FORMAT (' Process',I5,' computed',I5,' vectors.')
 2001    FORMAT (' Process',I5,' is computing vector ',I5)
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
         END IF
         RETURN
 1000    FORMAT (A)
 2000    FORMAT (1X,A,' (y,n)>')
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
         CHARACTER* (*) STRING
         WRITE(6,2000) STRING
         READ (5,*) ASK_I
         RETURN
 2000    FORMAT (1X,A,'>')
         END
C=======================================================================
         INTEGER FUNCTION INTINC (I)
C-----------------------------------------------------------------------
C        This function increments a volatile shared variable.  It is
C        placed in a subroutine to prevent some compilers from storing
C        the value in a register and not updating it.
         INTEGER I
         I = I + 1
         INTINC = I
```

```
        RETURN
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 12. GENERAL OUT-OF-CORE MATRIX

This example solves a system of equations **[A]{X}={B}** and then computes the error **{E}=[A]{X}-{B}**, where the following may be specified as input:

- ***FMS*** Module

  RS = Real Symmetric

  RN = Real Nonsymmetric

  CH = Complex Hermitian

  CS = Complex Symmetric

  CN = Complex Nonsymmetric

- Number of equations
- Number of solution vectors
- Any FMS Parameter

This example is designed to support pre-sales benchmarking of machines.

```
C       E X A M P L E   12
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_12')
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25), LUB(25)
        INTEGER     LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
```

```
      POINTER (RMD_PTR, RMD)
      POINTER (IMD_PTR, IMD)
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C
C     Input data:
      LOGICAL ASK
      INTEGER ASK_I
C
C     Profile vector for a full matrix:
      INTEGER LOWEQ(1)
C
C     Local variables:
      INTEGER     IDTYPE, ISTYPE, I, LB, LAX, LDISK, NV
      INTEGER     LENX, LENF
      REAL*8      EI, ERROR
C
C     Common block to communicate with fill routines:
      COMMON /MYDATA/MOD, N, NRHS
      DATA LOWEQ/-1/
C
C (1) Initialize FMS:
      CALL FMSINI
      CALL FMSPSH(MYNAME)
      CALL FMSIST ('MDATAU', 1)
    1 CONTINUE
      WRITE (6,*) 'Enter the FMS module number (1 to 5)'
      WRITE (6,*) '   1 = Real Symmetric'
      WRITE (6,*) '   2 = Real Nonsymmetric'
      WRITE (6,*) '   3 = Complex Hermitian'
      WRITE (6,*) '   4 = Complex Symmetric'
      WRITE (6,*) '   5 = Complex Nonsymmetric'
      READ  (5,*) MOD
      IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
      N      = ASK_I('Enter the number of equations')
      NRHS   = ASK_I('Enter the number of solution vectors')
      WRITE (6,*) 'You may now alter any FMS parameter.'
      WRITE (6,*) 'When you are finished, type the letters RETURN'
```

```
      CALL FMSSET
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
      IDTYPE = 1
      ISTYPE = 1
      IF(MOD .GE. 3) IDTYPE = 2
      IF(MOD .EQ. 2) ISTYPE = 2
      IF(MOD .EQ. 3) ISTYPE = 3
      IF(MOD .EQ. 5) ISTYPE = 2
C
C (2)   Open FMS files:
      IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUB', LUB)
C
C       Populate test vector:
      IF(IDTYPE .EQ. 1) THEN
         LENX = LUX(4)
         CALL FMSRMG (RMD, LB,  LENX)
         CALL FMSRMG (RMD, LAX, LENX)
         DO I = 0,(N-1)
            RMD(LB+I) = 0.0D0
         END DO
         RMD(LB) = 1.0D0
      ELSE
         LENX = LUX(4)/2
         CALL FMSCMG (CMD, LB,  LENX)
         CALL FMSCMG (CMD, LAX, LENX)
         DO I = 0,(N-1)
            CMD(LB+I) = (0.0D0,0.0D0)
         END DO
         IF(MOD.EQ.3) THEN
            CMD(LB) = (1.0D0,0.0D0)
         ELSE
```

```
               CMD(LB) = (1.0D0,1.0D0)
            END IF
         END IF
C
C (3)    Write data to FMS files:
C
C        Matrix file:
         IF(MOD.EQ.1) CALL RSDA (LUS0, 0, LUF)
         IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
         IF(MOD.EQ.3) CALL CHDA (LUS0, 0, LUF)
         IF(MOD.EQ.4) CALL CSDA (LUS0, 0, LUF)
         IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C        Vector file:
         LDISK  = 1
         DO NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSWRT (LUB(1), LDISK, RMD(LB), LUB(4))
            ELSE
               CALL FMSWRT (LUB(1), LDISK, CMD(LB), LUB(4))
            END IF
            LDISK = LDISK + LUB(4)
         END DO
C
C (4)    Perform matrix algebra:
C
C        Factor matrix [A]:
         IF(MOD.EQ.1) CALL RSDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.2) CALL RNDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.3) CALL CHDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.4) CALL CSDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.5) CALL CNDF (LUF, LUF, LUB, LUX,  NUMRED)
C
C        Solve [A]{X} = {B}:
         IF(MOD.EQ.1) CALL RSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.3) CALL CHDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.4) CALL CSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUB, LUX, NRHS, ISKIP)
```

```
C
C       Compute the enengy [F] = {X}T{B}:
        IF(ISTYPE .EQ. 1) THEN
           LENF = NRHS*(NRHS+1)/2
        ELSE
           LENF = NRHS*NRHS
        END IF
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMG (RMD, LB,  LENF)
        ELSE
           CALL FMSCMG (CMD, LF, LENF)
        END IF
        IF(MOD.EQ.1) CALL RSDVVM (LUX, LUB, RMD(LF), NRHS)
        IF(MOD.EQ.2) CALL RNDVVM (LUX, LUB, RMD(LF), NRHS, NRHS)
        IF(MOD.EQ.3) CALL CHDVVM (LUX, LUB, CMD(LF), NRHS)
        IF(MOD.EQ.4) CALL CSDVVM (LUX, LUB, CMD(LF), NRHS)
        IF(MOD.EQ.5) CALL CNDVVM (LUX, LUB, CMD(LF), NRHS, NRHS)
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMR (RMD, LB,  LENF)
        ELSE
           CALL FMSCMR (CMD, LF, LENF)
        END IF
C
C       Repopulate matrix [A]:
        CALL FMSIST ('MDATAU',  1)
        IF(MOD.EQ.1) CALL RSDA (LUS0, 0, LUF)
        IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
        IF(MOD.EQ.3) CALL CHDA (LUS0, 0, LUF)
        IF(MOD.EQ.4) CALL CSDA (LUS0, 0, LUF)
        IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C       Multiply {B} = [A]{X}:
        IF(MOD.EQ.1) CALL RSDMVM (LUF, LUX, LUB, NRHS)
        IF(MOD.EQ.2) CALL RNDMVM (LUF, LUX, LUB, NRHS)
        IF(MOD.EQ.3) CALL CHDMVM (LUF, LUX, LUB, NRHS)
        IF(MOD.EQ.4) CALL CSDMVM (LUF, LUX, LUB, NRHS)
        IF(MOD.EQ.5) CALL CNDMVM (LUF, LUX, LUB, NRHS)


C
```

```
C (5)   Read data from FMS files:
C       Check the answer:
        ERROR = 0.0D0
        LDISK  = 1
        DO NV = 1,NRHS
           IF(IDTYPE .EQ. 1) THEN
              CALL FMSRED (LUB(1), LDISK, RMD(LAX), N)
              DO I = 0,(N-1)
                 EI = ABS(RMD(LAX+I) - RMD(LB+I))
                 IF(EI .GT. ERROR) ERROR = EI
              END DO
           ELSE
              CALL FMSRED (LUB(1), LDISK, CMD(LAX), 2*N)
              DO I = 0,(N-1)
                 EI = ABS(CMD(LAX+I) - CMD(LB+I))
                 IF(EI .GT. ERROR) ERROR = EI
              END DO
           END IF
           LDISK  = LDISK  + LUB(4)
        END DO
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUF)
        CALL FMSCV (LUX)
        CALL FMSCV (LUB)
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMR (RMD, LB,  LENX)
           CALL FMSRMR (RMD, LAX, LENX)
        ELSE
           CALL FMSCMR (CMD, LB,  LENX)
           CALL FMSCMR (CMD, LAX, LENX)
        END IF
        IF(ASK('Do you want another solution?')) GO TO 1
        CALL FMSPOP(MYNAME)
        CALL FMSEND
        END
C=======================================================================
        REAL*8 FUNCTION AIJNUM(I,J)
```

```
C=======================================================================
        INTEGER I, J, IJ
        REAL*8 AI, AJ, AN
        COMMON /MYDATA/MOD, N, NRHS
C       Generate a matrix term for row I, column J that is
C       reproducible and will cause pivoting.
        AI  = DFLOAT(I)
        AJ  = DFLOAT(J)
        AN  = DFLOAT(N)
        AIJNUM = SQRT(2.0*AI*AJ)
        AIJNUM = SQRT(10.0*(AIJNUM - INT(AIJNUM)))
        RETURN
        END
C       ================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*)
        LOGICAL    IDIAG
C
D       PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
        IF(IROW1 .EQ. JCOL1) THEN
           IDIAG = .TRUE.
        ELSE
           IDIAG = .FALSE.
        END IF
        DO I = IROW1,IROW2
           J1 = MAX0(LOWEQ(I),JCOL1)
           J2 = MIN0(I-1,JCOL2)
           DO J = J1,J2
              A(LOCEQ(I)+IJSTEP*J) = AIJNUM(I,J)
           END DO
           IF(IDIAG) THEN
              D(I) = AIJNUM(I,I)
           END IF
        END DO
```

```
      RETURN
D2000 FORMAT('RSUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
      END
C     ================================================================
      SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                   JCOL1, JCOL2, IJSTEP)
C     ================================================================
      INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
      INTEGER    LOWEQ(*), LOCEQ(*)
      REAL*8     A(0:*), D(*)
C
D     PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
      DO J = JCOL1,JCOL2
         I1 = MAX0(LOWEQ(J),IROW1)
         I2 = MIN0(J-1,IROW2)
         DO I = I1,I2
            A(LOCEQ(J)+IJSTEP*I) = AIJNUM(I,J)
         END DO
      END DO
      RETURN
D2000 FORMAT('RNUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
      END
C     ================================================================
      SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                   JCOL1, JCOL2, IJSTEP)
C     ================================================================
      INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
      INTEGER    LOWEQ(*), LOCEQ(*)
      REAL*8     D(*), AIJ_RE, AIJ_IM
      COMPLEX*16 A(0:*)
      LOGICAL    IDIAG
C
D     PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
      IF(IROW1 .EQ. JCOL1) THEN
         IDIAG = .TRUE.
      ELSE
         IDIAG = .FALSE.
      END IF
      DO I = IROW1,IROW2
```

```
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            DO J = J1,J2
               AIJ_RE = AIJNUM(I,J)
               AIJ_IM = - AIJ_RE
               A(LOCEQ(I)+IJSTEP*J) = DCMPLX(AIJ_RE,AIJ_IM)
            END DO
            IF(IDIAG) THEN
               D(I) = AIJNUM(I,I)
            END IF
         END DO
         RETURN
D2000    FORMAT('CHUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
         END
C        ================================================================
         SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     AIJ_RE, AIJ_IM
         COMPLEX*16 A(0:*), D(*)
         LOGICAL    IDIAG
C
D        PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
         IF(IROW1 .EQ. JCOL1) THEN
            IDIAG = .TRUE.
         ELSE
            IDIAG = .FALSE.
         END IF
         DO I = IROW1,IROW2
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            DO J = J1,J2
               AIJ_RE = AIJNUM(I,J)
               AIJ_IM = - AIJ
               A(LOCEQ(I)+IJSTEP*J) = DCMPLX(AIJ_RE,AIJ_IM)
            END DO
            IF(IDIAG) THEN
```

```
              AIJ_RE = AIJNUM(I,I)
              AIJ_IM = - AIJ_RE
              D(I) = DCMPLX(AIJ_RE,AIJ_IM)
           END IF
        END DO
        RETURN
D2000   FORMAT('CSUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
        END
C       ===============================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ===============================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     AIJ_RE, AIJ_IM
        COMPLEX*16 A(0:*), D(*)
C
D       PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
        DO J = JCOL1,JCOL2
           I1 = MAX0(LOWEQ(J),IROW1)
           I2 = MIN0(J-1,IROW2)
           DO I = I1,I2
              AIJ_RE = AIJNUM(I,J)
              AIJ_IM = - AIJ_RE
              A(LOCEQ(J)+IJSTEP*I) = DCMPLX(AIJ_RE,AIJ_IM)
           END DO
        END DO
        RETURN
D2000   FORMAT('CNUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
        END
C       ===============================================================
        SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C       ===============================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        REAL*8     A(0:*)
C
D       PRINT 2000, NUMEQ, JEQN1, JEQN2
        DO J=JEQN1,JEQN2
```

```
            DO I=1,NUMEQ
               LIJ = LOCI(I) + LOCJ(J,LUFLAG(I))
               A(LIJ) = AIJNUM(I,J)
            END DO
         END DO
         RETURN
D2000    FORMAT('RNUSLB filling A(1:',I6,',',I6,':',I6,')')
         END
C        ================================================================
         SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         COMPLEX*16 A(0:*)
         REAL*8     AIJ_RE, AIJ_IM
C
D        PRINT 2000, NUMEQ, JEQN1, JEQN2
         DO J=JEQN1,JEQN2
            DO I=1,NUMEQ
               LIJ = LOCI(I) + LOCJ(J,LUFLAG(I))
               AIJ_RE = AIJNUM(I,J)
               AIJ_IM = - AIJ_RE
               A(LIJ) = DCMPLX(AIJ_RE,AIJ_IM)
            END DO
         END DO
         RETURN
D2000    FORMAT('CNUSLB filling A(1:',I6,',',I6,':',I6,')')
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
```

```
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

Copyright © Multipath Corporation

```
C       E X A M P L E   12
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_12')
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUX(25), LUB(25)
        INTEGER     LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Input data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       Local variables:
        INTEGER     IDTYPE, ISTYPE, I, LB, LAX, LDISK, NV
        INTEGER     LENX, LENF
        REAL*8      EI, ERROR
C
C       Common block to communicate with fill routines:
        COMMON /MYDATA/MOD, N, NRHS
        DATA LOWEQ/-1/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('MDATAU', 1)
    1   CONTINUE
        WRITE (6,*) 'Enter the FMS module number (1 to 5)'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        READ  (5,*) MOD
        IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
        N      = ASK_I('Enter the number of equations')
        NRHS   = ASK_I('Enter the number of solution vectors')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        IDTYPE = 1
```

```
         ISTYPE = 1
         IF(MOD .GE. 3) IDTYPE = 2
         IF(MOD .EQ. 2) ISTYPE = 2
         IF(MOD .EQ. 3) ISTYPE = 3
         IF(MOD .EQ. 5) ISTYPE = 2
C
C (2)    Open FMS files:
         IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUB', LUB)
C
C        Populate test vector:
         IF(IDTYPE .EQ. 1) THEN
            LENX = LUX(4)
            CALL FMSRMG (RMD, LB,  LENX)
            CALL FMSRMG (RMD, LAX, LENX)
            DO I = 0,(N-1)
               RMD(LB+I) = 0.0D0
            END DO
            RMD(LB) = 1.0D0
         ELSE
            LENX = LUX(4)/2
            CALL FMSCMG (CMD, LB,  LENX)
            CALL FMSCMG (CMD, LAX, LENX)
            DO I = 0,(N-1)
               CMD(LB+I) = (0.0D0,0.0D0)
            END DO
            IF(MOD.EQ.3) THEN
               CMD(LB) = (1.0D0,0.0D0)
            ELSE
               CMD(LB) = (1.0D0,1.0D0)
            END IF
         END IF
C
C (3)    Write data to FMS files:
C
C        Matrix file:
         IF(MOD.EQ.1) CALL RSDA (LUS0, 0, LUF)
         IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
         IF(MOD.EQ.3) CALL CHDA (LUS0, 0, LUF)
         IF(MOD.EQ.4) CALL CSDA (LUS0, 0, LUF)
         IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C        Vector file:
         LDISK  = 1
         DO NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSWRT (LUB(1), LDISK, RMD(LB), LUB(4))
            ELSE
               CALL FMSWRT (LUB(1), LDISK, CMD(LB), LUB(4))
            END IF
            LDISK = LDISK + LUB(4)
         END DO
C
C (4)    Perform matrix algebra:
C
C        Factor matrix [A]:
         IF(MOD.EQ.1) CALL RSDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.2) CALL RNDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.3) CALL CHDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.4) CALL CSDF (LUF, LUF, LUB, LUX,  NUMRED)
```

```
         IF(MOD.EQ.5) CALL CNDF (LUF, LUF, LUB, LUX,  NUMRED)
C
C        Solve [A]{X} = {B}:
         IF(MOD.EQ.1) CALL RSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.3) CALL CHDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.4) CALL CSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUB, LUX, NRHS, ISKIP)
C
C        Compute the enengy [F] = {X}T{B}:
         IF(ISTYPE .EQ. 1) THEN
            LENF = NRHS*(NRHS+1)/2
         ELSE
            LENF = NRHS*NRHS
         END IF
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMG (RMD, LF, LENF)
         ELSE
            CALL FMSCMG (CMD, LF, LENF)
         END IF
         IF(MOD.EQ.1) CALL RSDVVM (LUX, LUB, RMD(LF), NRHS)
         IF(MOD.EQ.2) CALL RNDVVM (LUX, LUB, RMD(LF), NRHS, NRHS)
         IF(MOD.EQ.3) CALL CHDVVM (LUX, LUB, CMD(LF), NRHS)
         IF(MOD.EQ.4) CALL CSDVVM (LUX, LUB, CMD(LF), NRHS)
         IF(MOD.EQ.5) CALL CNDVVM (LUX, LUB, CMD(LF), NRHS, NRHS)
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LF, LENF)
         ELSE
            CALL FMSCMR (CMD, LF, LENF)
         END IF
C
C        Repopulate matrix [A]:
         CALL FMSIST ('MDATAU',  1)
         IF(MOD.EQ.1) CALL RSDA (LUS0, 0, LUF)
         IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
         IF(MOD.EQ.3) CALL CHDA (LUS0, 0, LUF)
         IF(MOD.EQ.4) CALL CSDA (LUS0, 0, LUF)
         IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C        Multiply {B} = [A]{X}:
         IF(MOD.EQ.1) CALL RSDMVM (LUF, LUX, LUB, NRHS)
         IF(MOD.EQ.2) CALL RNDMVM (LUF, LUX, LUB, NRHS)
         IF(MOD.EQ.3) CALL CHDMVM (LUF, LUX, LUB, NRHS)
         IF(MOD.EQ.4) CALL CSDMVM (LUF, LUX, LUB, NRHS)
         IF(MOD.EQ.5) CALL CNDMVM (LUF, LUX, LUB, NRHS)

C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
         LDISK  = 1
         DO NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRED (LUB(1), LDISK, RMD(LAX), N)
               DO I = 0,(N-1)
                  EI = ABS(RMD(LAX+I) - RMD(LB+I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            ELSE
               CALL FMSRED (LUB(1), LDISK, CMD(LAX), 2*N)
               DO I = 0,(N-1)
                  EI = ABS(CMD(LAX+I) - CMD(LB+I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END IF
```

```
            LDISK  = LDISK  + LUB(4)
         END DO
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUF)
         CALL FMSCV (LUX)
         CALL FMSCV (LUB)
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LB,  LENX)
            CALL FMSRMR (RMD, LAX, LENX)
         ELSE
            CALL FMSCMR (CMD, LB,  LENX)
            CALL FMSCMR (CMD, LAX, LENX)
         END IF
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP (MYNAME)
         CALL FMSEND
         END
C=======================================================================
         REAL*8 FUNCTION AIJNUM(I,J)
C=======================================================================
         INTEGER I, J, IJ
         REAL*8 AI, AJ, AN
         COMMON /MYDATA/MOD, N, NRHS
C        Generate a matrix term for row I, column J that is
C        reproducible and will cause pivoting.
         AI  = DFLOAT(I)
         AJ  = DFLOAT(J)
         AN  = DFLOAT(N)
         AIJNUM = SQRT(2.0*AI*AJ)
         AIJNUM = SQRT(10.0*(AIJNUM - INT(AIJNUM)))
         RETURN
         END
C        ===============================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ===============================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*)
         LOGICAL    IDIAG
C
D        PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
         IF(IROW1 .EQ. JCOL1) THEN
            IDIAG = .TRUE.
         ELSE
            IDIAG = .FALSE.
         END IF
         DO I = IROW1,IROW2
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            DO J = J1,J2
               A(LOCEQ(I)+IJSTEP*J) = AIJNUM(I,J)
            END DO
            IF(IDIAG) THEN
               D(I) = AIJNUM(I,I)
            END IF
         END DO
         RETURN
D2000    FORMAT('RSUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
         END
C        ===============================================================
         SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
```

```
     1                    JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*)
C
D       PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
        DO J = JCOL1,JCOL2
           I1 = MAX0(LOWEQ(J),IROW1)
           I2 = MIN0(J-1,IROW2)
           DO I = I1,I2
              A(LOCEQ(J)+IJSTEP*I) = AIJNUM(I,J)
           END DO
        END DO
        RETURN
D2000   FORMAT('RNUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
        END
C       ================================================================
        SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                    JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     D(*), AIJ_RE, AIJ_IM
        COMPLEX*16 A(0:*)
        LOGICAL    IDIAG
C
D       PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
        IF(IROW1 .EQ. JCOL1) THEN
           IDIAG = .TRUE.
        ELSE
           IDIAG = .FALSE.
        END IF
        DO I = IROW1,IROW2
           J1 = MAX0(LOWEQ(I),JCOL1)
           J2 = MIN0(I-1,JCOL2)
           DO J = J1,J2
              AIJ_RE = AIJNUM(I,J)
              AIJ_IM = - AIJ_RE
              A(LOCEQ(I)+IJSTEP*J) = DCMPLX(AIJ_RE,AIJ_IM)
           END DO
           IF(IDIAG) THEN
              D(I) = AIJNUM(I,I)
           END IF
        END DO
        RETURN
D2000   FORMAT('CHUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
        END
C       ================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                    JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     AIJ_RE, AIJ_IM
        COMPLEX*16 A(0:*), D(*)
        LOGICAL    IDIAG
C
D       PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
        IF(IROW1 .EQ. JCOL1) THEN
           IDIAG = .TRUE.
        ELSE
           IDIAG = .FALSE.
        END IF
        DO I = IROW1,IROW2
```

```
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            DO J = J1,J2
               AIJ_RE = AIJNUM(I,J)
               AIJ_IM = - AIJ
               A(LOCEQ(I)+IJSTEP*J) = DCMPLX(AIJ_RE,AIJ_IM)
            END DO
            IF(IDIAG) THEN
               AIJ_RE = AIJNUM(I,I)
               AIJ_IM = - AIJ_RE
               D(I) = DCMPLX(AIJ_RE,AIJ_IM)
            END IF
         END DO
         RETURN
D2000    FORMAT('CSUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
         END
C        ================================================================
         SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     AIJ_RE, AIJ_IM
         COMPLEX*16 A(0:*), D(*)
C
D        PRINT 2000, IROW1, IROW2, JCOL1, JCOL2
         DO J = JCOL1,JCOL2
            I1 = MAX0(LOWEQ(J),IROW1)
            I2 = MIN0(J-1,IROW2)
            DO I = I1,I2
               AIJ_RE = AIJNUM(I,J)
               AIJ_IM = - AIJ_RE
               A(LOCEQ(J)+IJSTEP*I) = DCMPLX(AIJ_RE,AIJ_IM)
            END DO
         END DO
         RETURN
D2000    FORMAT('CNUBLK filling A(',I6,':',I6,',',I6,':',I6,')')
         END
C        ================================================================
         SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         REAL*8     A(0:*)
C
D        PRINT 2000, NUMEQ, JEQN1, JEQN2
         DO J=JEQN1,JEQN2
            DO I=1,NUMEQ
               LIJ = LOCI(I) + LOCJ(J,LUFLAG(I))
               A(LIJ) = AIJNUM(I,J)
            END DO
         END DO
         RETURN
D2000    FORMAT('RNUSLB filling A(1:',I6,',',I6,':',I6,')')
         END
C        ================================================================
         SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         COMPLEX*16 A(0:*)
         REAL*8     AIJ_RE, AIJ_IM
C
D        PRINT 2000, NUMEQ, JEQN1, JEQN2
         DO J=JEQN1,JEQN2
```

```
            DO I=1,NUMEQ
               LIJ = LOCI(I) + LOCJ(J,LUFLAG(I))
               AIJ_RE = AIJNUM(I,J)
               AIJ_IM = - AIJ_RE
               A(LIJ) = DCMPLX(AIJ_RE,AIJ_IM)
            END DO
         END DO
         RETURN
D2000    FORMAT('CNUSLB filling A(1:',I6,',',I6,':',I6,')')
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
         END IF
         RETURN
 1000    FORMAT (A)
 2000    FORMAT (1X,A,' (y,n)>')
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
         CHARACTER* (*) STRING
         WRITE(6,2000) STRING
         READ (5,*) ASK_I
         RETURN
 2000    FORMAT (1X,A,'>')
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 13. USING FMSPUT

This example illustrates how to fill a matrix in parallel using subroutine FMSPUT.

```
C       E X A M P L E  13
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_13')
C
C       Subroutines called in parallel:
        EXTERNAL WRITEM
        EXTERNAL WRITEV
C
C       FMS Module number (1=RS, 2=RN, 3=CH, 4=CS, 5=CN)
        INTEGER MOD
C
C       Number of equations:
        INTEGER N
C
C       Number of R.H.S. vectors:
        INTEGER NRHS
C
C       Size of block to fill matrix:
        INTEGER NROWS, NCOLS
C
C       Problem data:
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
C       Work queue index (shared):
        COMMON/SHARE/IWORK
        INTEGER IWORK
C
C       FMS matrix and vector file attributes:
        INTEGER     LUA(15), LUF(25), LUA0(25)
        DATA LUA0(1)/0/
```

```
      INTEGER     LUX(25)
      INTEGER     LUS0(25)
C
C     FMS memory management requires the following arrays:
      POINTER (CMD_PTR, CMD)
      POINTER (RMD_PTR, RMD)
      POINTER (IMD_PTR, IMD)
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C
C     Input Data:
      LOGICAL ASK
      INTEGER ASK_I
C
C     FMS Parameters:
      INTEGER     MDATAU
C
C     Profile vector for a full matrix:
      INTEGER     LOWEQ(1)
      DATA        LOWEQ/-1/
C
C     Local variables:
      INTEGER     MAXCPU
      INTEGER     IDTYPE
      INTEGER     ISTYPE
      INTEGER     I, LX, LDISK, NV
      REAL*8      EI
      LOGICAL     ITEST
C
C     Number of vectors to reduce during factoring:
      PARAMETER (NUMRED = 0)
C
C     Skip operations during solving (no):
      PARAMETER (ISKIP  = 0)
C
C     Constants:
      REAL*8     RALPHA(1)
      DATA RALPHA/1.0D0/
```

```
      COMPLEX*16 CALPHA(1)
      DATA CALPHA/(1.0D0,0.0D0)/
C
C (1)   Initialize FMS:
      CALL FMSINI
      CALL FMSPSH(MYNAME)
      CALL FMSIST ('IPRF'  , 1026)
    1 CONTINUE
      WRITE (6,*) 'Enter the FMS module number (1 to 5)'
      WRITE (6,*) '   1 = Real Symmetric'
      WRITE (6,*) '   2 = Real Nonsymmetric'
      WRITE (6,*) '   3 = Complex Hermitian'
      WRITE (6,*) '   4 = Complex Symmetric'
      WRITE (6,*) '   5 = Complex Nonsymmetric'
      READ  (5,*) MOD
      IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
      N    = ASK_I('Enter the number of equations')
      ISCALE = 1
      DO I=1,15
         IF(10**ISCALE .GT. N) GO TO 11
         ISCALE = ISCALE + 1
      END DO
   11 CONTINUE
      NRHS = ASK_I('Enter the number of solution vectors')
      NROWS= ASK_I('Enter the number of rows per block (0=NEQBIO)')
      NCOLS= ASK_I('Enter the number of columns per block (0=NEQBIO)')
      IF(ASK('Do you want to check the matrix data?')) THEN
         MDATAU = 2
      ELSE
         MDATAU = 0
      END IF
      CALL FMSIST ('MDATAU', MDATAU)
      WRITE (6,*) 'You may now alter any FMS parameter.'
      WRITE (6,*) 'When you are finished, type the letters RETURN'
      CALL FMSSET
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
      CALL FMSIGT ('MAXCPU', MAXCPU)
```

```
         IF(MOD .LE. 2) THEN
            IDTYPE = 1
         ELSE
            IDTYPE = 2
         END IF
         ISTYPE = 1
         IF(MOD .EQ. 2) ISTYPE = 2
         IF(MOD .EQ. 3) ISTYPE = 3
         IF(MOD .EQ. 5) ISTYPE = 2
C
C (2)    Open FMS files:
         CALL FMSIST ('IOKIDS', MAXCPU-1)
         IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C (3)    Write data to FMS files:
C
C        Fill the RHS vectors in parallel:
C        Initialize FMSPUT:
            CALL FMSPUT (LUX, 0, 0, 0, 0, RMD, 1)
C        Generate and write the vector data in parallel:
         IWORK  = 0
         DO 30 ICPU=2,MAXCPU
            CALL FMSPAR (1, WRITEV, LUX)
   30    CONTINUE
         IF(MAXCPU .GT. 1) CALL FMSRUN
         CALL WRITEV (LUX)
         IF(MAXCPU .GT. 1) CALL FMSYNC
C
C        End FMSPUT:
            CALL FMSPUT (LUX, 0, 0, N+1, 0, RMD, 1)
C
C        If testing the matrix fill, first fill with one processor:
         IF(MDATAU .NE. 0) THEN
            IF(ISTYPE .EQ. 2) THEN
```

```
C             This matrix is nonsymmetric:
              CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
           ELSE
C             This matrix is symmetric or Hermitian:
C             Initialize FMSPUT to define the matrix terms in the upper
C             triangle.
              CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
           END IF
           ISIGN  = -1
           IWORK  = 0
           CALL WRITEM (NROWS, NCOLS, LUA)
C          End FMSPUT:
           CALL FMSPUT (LUA, 0, 0, N+1, 0, RMD, 1)
        END IF
C
C       Fill the matrix in parallel:
C
C       Initialize FMSPUT:
        IF(ISTYPE .EQ. 2) THEN
C          This matrix is nonsymmetric:
           CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
        ELSE
C          This matrix is symmetric or Hermitian:
C          Initialize FMSPUT to define the matrix terms in the upper
C          triangle.
           CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
        END IF
        ISIGN  = 1
        IWORK  = 0
        DO 35 ICPU=2,MAXCPU
           CALL FMSPAR (3, WRITEM, NROWS, NCOLS, LUA)
   35   CONTINUE
        IF(MAXCPU .GT. 1) CALL FMSRUN
        CALL WRITEM (NROWS, NCOLS, LUA)
        IF(MAXCPU .GT. 1) CALL FMSYNC
C
C       End FMSPUT:
        CALL FMSPUT (LUA, 0, 0, N+1, 0, RMD, 1)
C
```

```
C (4)   Perform matrix algebra:
C
C       This example offers two testing options:
C       MDATAU=0; Factor the matrix with valid test data;
C       MDATAU=2; Check the matrix data and quit.
        ERROR = 0.0D0
        IF(MDATAU .EQ. 0) THEN
           DO I=1,25
              LUF(I) = LUA(I)
           END DO
        ELSE
           LUF(1) = 0
        END IF
        IF(MOD.EQ.1) CALL RSDAF (LUA, RALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,
NUMRED)
        IF(MOD.EQ.2) CALL RNDAF (LUA, RALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,
NUMRED)
        IF(MOD.EQ.3) CALL CHDAF (LUA, RALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,
NUMRED)
        IF(MOD.EQ.4) CALL CSDAF (LUA, CALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,
NUMRED)
        IF(MOD.EQ.5) CALL CNDAF (LUA, CALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,
NUMRED)
C
        IF(MDATAU .EQ. 2) GO TO 70
        IF(MOD.EQ.1) CALL RSDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.2) CALL RNDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.3) CALL CHDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.4) CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.5) CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Check the answer:
C
C       Allocate a vector for reading {X}:
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMG (RMD, LX, N)
        ELSE
```

```
            CALL FMSCMG (CMD, LX, N)
         END IF
         LDISK  = 1
         DO 60 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRED (LUX(1), LDISK, RMD(LX), N)
               DO 50 I = 1,N
                  EI = ABS(RMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   50          CONTINUE
            ELSE
               CALL FMSRED (LUX(1), LDISK, CMD(LX), 2*N)
               DO 51 I = 1,N
                  EI = ABS(CMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   51          CONTINUE
            END IF
            LDISK  = LDISK  + LUX(4)
   60    CONTINUE
   70    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUF)
         CALL FMSCV (LUX)
         IF(MDATAU .EQ. 0) THEN
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRMR (RMD, LX, N)
            ELSE
               CALL FMSCMR (CMD, LX, N)
            END IF
         END IF
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE WRITEV (LUX)
C=======================================================================
```

```
C
C       DESCRIPTION:
C          This subroutine runs in parallel. For each vector
C          that it finds, it fills in the terms and calls FMSPUT to
C          write the vector to the vector file LUX.
C
C       FORMAL PARAMETERS:
C          (R ) LUX(25) = FMS vector file attributes
C-----------------------------------------------------------------------
C
C       Formal Parameters:
        INTEGER    LUX(25)
C
C       Local Variables:
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='WRITEV')
        INTEGER    MYNODE
        INTEGER    LUPR
        INTEGER    NUMEQ
        INTEGER    NUMVEC
        INTEGER    NROWS, NCOLS
        INTEGER    IEQN1, IEQN2, JEQN1, JEQN2
        INTEGER    MY_WORK
        INTEGER    MY_TOTAL
        INTEGER    LX
        REAL*8     RMD(0:1)
        COMPLEX*16 CMD(0:1)
        POINTER    (RMD_PTR,RMD)
        POINTER    (CMD_PTR,CMD)
        COMMON/SHARE/IWORK
        INTEGER    PARENT
        PARAMETER (PARENT = 0)
C
        CALL FMSPSH(MYNAME)
C       Get your CPU number (0 to MAXCPU-1)
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('LUPR',   LUPR)
        NUMEQ  = LUX( 3)
        LENVEC = LUX( 4)
```

```
         IDTYPE = LUX( 5)
         NUMVEC = LUX( 6)
         NROWS  = NUMEQ
         NCOLS  = 1
         IEQN1  = 1
         IEQN2  = NUMEQ
         LDX    = LENVEC/IDTYPE
C
C        Allocate a work array for storing the data you generate:
C        For this example, we will fill [B] vector by vector.
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSIGT ('MEMPTR', RMD_PTR)
            CALL FMSRMG (RMD, LX, NUMEQ)
         ELSE
            CALL FMSIGT ('MEMPTR', CMD_PTR)
            CALL FMSCMG (CMD, LX, NUMEQ)
         END IF
C
C        Find the next vector to fill:
         MY_TOTAL = 0
   10    CONTINUE
            CALL FMSONE
C              IWORK   = IWORK + 1
C              MY_WORK = IWORK
               MY_WORK = INTINC(IWORK)
C              Write message under protection of lock:
               IF(MY_WORK .LE. NUMVEC) WRITE(LUPR,2001) MYNODE, MY_WORK
            CALL FMSALL
            IF(MY_WORK .GT. NUMVEC) GO TO 100
            MY_TOTAL = MY_TOTAL + 1
            JEQN1  = MY_WORK
            JEQN2  = JEQN1
C
            IF(IDTYPE .EQ. 1) THEN
               CALL RVFILL (RMD(LX), IEQN1, IEQN2, JEQN1, JEQN2, NUMEQ)
               CALL FMSPUT (LUX, NROWS, NCOLS, IEQN1, JEQN1, RMD(LX),
     1                      LDX)
            ELSE
               CALL CVFILL (CMD(LX), IEQN1, IEQN2, JEQN1, JEQN2, NUMEQ)
```

```
               CALL FMSPUT (LUX, NROWS, NCOLS, IEQN1, JEQN1, CMD(LX),
     1                      LDX)
            END IF
         GO TO 10
  100    CONTINUE
C        Your part of the work is done:
C
C        Return work array:
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LX, N)
         ELSE
            CALL FMSCMR (CMD, LX, N)
         END IF
C
C        Report your total work:
         CALL FMSONE
            WRITE(LUPR,2000) MYNODE, MY_TOTAL
         CALL FMSALL
         CALL FMSPOP(MYNAME)
         RETURN
 2000    FORMAT (' Process',I3,' computed',I5,' vectors.')
 2001    FORMAT (' Process',I3,' is computing vector ',I5)
         END
C=======================================================================
         SUBROUTINE WRITEM (NROWS, NCOLS, LUA)
C=======================================================================
C
C        DESCRIPTION:
C           This subroutine runs in parallel. For each block
C           A(NROWS, NCOLS) that it finds, it fills in the terms and
C           calls FMSPUT to write the block to the matrix file LUA.
C
C        FORMAL PARAMETERS:
C           (R ) NROWS   = Number of rows per block
C           (R ) NCOLS   = Number of columns per block
C           (R ) LUA(25) = FMS matrix file attributes
C-----------------------------------------------------------------------
C
C        Formal Parameters:
```

```
      INTEGER    NROWS
      INTEGER    NCOLS
      INTEGER    LUA(25)
C
C     Local Variables:
      CHARACTER*6 MYNAME
      PARAMETER (MYNAME='WRITEM')
      INTEGER    NEQROW, NEQCOL
      INTEGER    NBKROW, NBKCOL
      INTEGER    IEQN1, IEQN2, JEQN1, JEQN2
      INTEGER    IBLOCK, JBLOCK
      INTEGER    NUMBLK
      INTEGER    MYNODE
      INTEGER    MY_WORK
      INTEGER    MY_TOTAL
      INTEGER    LUPR
      INTEGER    NUMEQ
      INTEGER    NEQBIO
      INTEGER    NUMSEG
      INTEGER    IDTYPE
      INTEGER    LA
      REAL*8     RMD(0:1)
      COMPLEX*16 CMD(0:1)
      POINTER    (RMD_PTR,RMD)
      POINTER    (CMD_PTR,CMD)
      COMMON/SHARE/IWORK
      INTEGER    PARENT
      PARAMETER (PARENT = 0)
C
      CALL FMSPSH(MYNAME)
C     Get your CPU number (0 to MAXCPU-1)
      CALL FMSIGT ('MYNODE', MYNODE)
      CALL FMSIGT ('LUPR',   LUPR)
      NUMEQ  = LUA( 8)
      NUMSEG = LUA( 6)
      IDTYPE = LUA(11)
      ISTYPE = LUA(12)
      MFMAT  = LUA(13)
      NEQBIO = LUA(21)
```

```
         NEQROW = NROWS
         NEQCOL = NCOLS
         IF(NEQROW .EQ. 0) THEN
            IF(MFMAT .EQ. 1) THEN
               NEQROW = NUMEQ/5
            ELSE
               NEQROW = NEQBIO
            END IF
         END IF
         IF(NEQCOL .EQ. 0) THEN
            IF(MFMAT .EQ. 1) THEN
               NEQCOL = NUMEQ/5
            ELSE
               NEQCOL = NEQBIO
            END IF
         END IF
         NBKROW = (NUMEQ+NEQROW-1)/NEQROW
         NBKCOL = (NUMEQ+NEQCOL-1)/NEQCOL
         NUMBLK = NBKROW*NBKCOL
         LDA    = NEQROW
C
C        Allocate a work array for storing the data you generate:
         CALL FMSIGT ('MDATAU', MDATAU)
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSIGT ('MEMPTR', RMD_PTR)
            CALL FMSRMG (RMD, LA, NEQROW*NEQCOL)
         ELSE
            CALL FMSIGT ('MEMPTR', CMD_PTR)
            CALL FMSCMG (CMD, LA, NEQROW*NEQCOL)
         END IF
C
C        Fill the matrix:
         MY_TOTAL = 0
   10    CONTINUE
            CALL FMSONE
C              IWORK   = IWORK + 1
C              MY_WORK = IWORK
               MY_WORK = INTINC(IWORK)
```

```
C             Write message under protection of lock:
              JBLOCK = (MY_WORK + NBKROW - 1)/NBKROW
              IBLOCK =  MY_WORK - NBKROW*(JBLOCK-1)
              IF(MY_WORK .LE. NUMBLK)
     1           WRITE(LUPR,2001) MYNODE, IBLOCK, JBLOCK
            CALL FMSALL
            IF(MY_WORK .GT. NUMBLK) GO TO 100
            MY_TOTAL = MY_TOTAL + 1
C
            IEQN1 = 1 + NEQROW*(IBLOCK-1)
            JEQN1 = 1 + NEQCOL*(JBLOCK-1)
            IEQN2 = IEQN1 + NEQROW - 1
            IEQN2 = MIN0(IEQN2,NUMEQ)
            JEQN2 = JEQN1 + NEQCOL - 1
            JEQN2 = MIN0(JEQN2,NUMEQ)
            NR    = IEQN2 - IEQN1 + 1
            NC    = JEQN2 - JEQN1 + 1
            IF(ISTYPE .NE. 2) THEN
C             This problem is symmetric.
C             Skip blocks entirely in [L]:
              IF(IEQN1 .GT. JEQN2) GO TO 10
            END IF
            IF(IDTYPE .EQ. 1) THEN
              CALL RMFILL (RMD(LA), NR, NC, IEQN1, JEQN1, NUMEQ)
              IF(MDATAU .GT. 0)
     1        CALL RCHECK (RMD(LA), NR, NC, IEQN1, JEQN1)
              CALL FMSPUT (LUA,  NR, NC, IEQN1, JEQN1, RMD(LA), NR)
            ELSE
              CALL CMFILL (CMD(LA), NR, NC, IEQN1, JEQN1, NUMEQ)
              IF(MDATAU .GT. 0)
     1        CALL CCHECK (CMD(LA), NR, NC, IEQN1, JEQN1)
              CALL FMSPUT (LUA,  NR, NC, IEQN1, JEQN1, CMD(LA), NR)
            END IF
          GO TO 10
  100     CONTINUE
C         Your part of the work is done:
C
C         Return work array:
          IF(IDTYPE .EQ. 1) THEN
```

```
            CALL FMSRMR (RMD, LX, NEQROW*NEQCOL)
         ELSE
            CALL FMSCMR (CMD, LX, NEQROW*NEQCOL)
         END IF
C
C        Report your total work:
         CALL FMSONE
            WRITE(LUPR,2000) MYNODE, MY_TOTAL
         CALL FMSALL
         CALL FMSPOP(MYNAME)
         RETURN
 2000    FORMAT (' Process',I3,' computed',I5,' blocks.')
 2001    FORMAT (' Process',I3,' is computing block (',I3,',',I3,')')
         END
C=======================================================================
         SUBROUTINE RVFILL (X, IROW1, IROW2, JCOL1, JCOL2, NUMEQ)
C=======================================================================
C        Fill a test vector with REAL*8 data:
         REAL*8  X(IROW1:IROW2,JCOL1:JCOL2)
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='RVFILL')
         INTEGER IROW1
         INTEGER IROW2
         INTEGER JCOL1
         INTEGER JCOL2
         REAL*8  ZERO, X1
         DATA    ZERO/0.0D0/
         DATA    X1  /1.0D0/
C
         CALL FMSPSH(MYNAME)
C        Zero out [X]
         DO IROW=IROW1,IROW2
            DO JCOL=JCOL1,JCOL2
               X(IROW,JCOL) = ZERO
            END DO
         END DO
C
C        Set first row to X1:
         IF(IROW1 .EQ. 1) THEN
```

```
            DO JCOL=JCOL1,JCOL2
               X(1,JCOL) = X1
            END DO
         END IF
         CALL FMSPOP(MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE CVFILL (X, IROW1, IROW2, JCOL1, JCOL2, NUMEQ)
C=======================================================================
C        Fill a test vector with COMPLEX*16 data:
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='CVFILL')
         COMPLEX*16 X(IROW1:IROW2,JCOL1:JCOL2)
         INTEGER IROW1
         INTEGER IROW2
         INTEGER JCOL1
         INTEGER JCOL2
         COMPLEX*16 ZERO, X1
         DATA       ZERO/(0.0D0, 0.0D0)/
         COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
         REAL*8 ERROR
C
         CALL FMSPSH(MYNAME)
         IF(MOD .EQ. 3) THEN
            X1 = (1.0D0,0.0D0)
         ELSE
            X1 = (1.0D0,1.0D0)
         END IF
C
C        Zero out [X]
         DO IROW=IROW1,IROW2
            DO JCOL=JCOL1,JCOL2
               X(IROW,JCOL) = ZERO
            END DO
         END DO
C
C        Set first row to X1:
         IF(IROW1 .EQ. 1) THEN
```

```
            DO JCOL=JCOL1,JCOL2
               X(1,JCOL) = X1
            END DO
         END IF
         CALL FMSPOP(MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE RMFILL (A, NR, NC, IROW1, JCOL1, NUMEQ)
C=======================================================================
C        Fill a matrix block with REAL*8 data:
         REAL*8  A(NR,NC)
         INTEGER NR
         INTEGER NC
         INTEGER IROW1
         INTEGER JCOL1
         INTEGER I, J, IROW, JCOL
         REAL*8  OFFDIA, ZERO, DIA, AI, AJ, SCALE
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='RMFILL')
         COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
         REAL*8 ERROR
         DATA    OFFDIA/-1.0D0/
         DATA    ZERO  / 0.0D0/
         DATA    DIA   / 1.0D0/
C
         CALL FMSPSH(MYNAME)
C        Get MDATAU:
         CALL FMSIGT ('MDATAU', MDATAU)
C
         IF(MDATAU .EQ. 0) THEN
C           Fill the block with test data:
C           Zero out [A]
            DO I=1,NR
               DO J=1,NC
                  A(I,J) = ZERO
               END DO
            END DO
C
```

```
C          Set first column to OFFDIA:
           IF(JCOL1 .EQ. 1) THEN
              DO I=1,MIN0(NR,NUMEQ-IROW1+1)
                 A(I,1) = OFFDIA
              END DO
           END IF
C
C          Set first row to OFFDIA:
           IF(IROW1 .EQ. 1) THEN
              DO J=1,MIN0(NC,NUMEQ-JCOL1+1)
                 A(1,J) = OFFDIA
              END DO
           END IF
C
C          Set diagonal to DIA:
           IDIA1 = MAX0(IROW1,JCOL1)
           IDIA2 = MIN0(IROW1+NR-1,JCOL1+NC-1,NUMEQ)
           IF(IDIA1 .LE. IDIA2) THEN
              DO IDIA = IDIA1,IDIA2
                 A(IDIA-IROW1+1,IDIA-JCOL1+1) = DIA
              END DO
           END IF
C
C          Set A(1,1) to NUMEQ
           IF( (IROW1 .EQ. 1) .AND.
     1          (JCOL1 .EQ. 1) ) THEN
              A(1,1) = DFLOAT(NUMEQ)
           END IF
        ELSE
C
C          Fill the block with (row,column) numbers:
           SCALE = 10**ISCALE
           SCALE = ISIGN * SCALE
           IF(MOD .EQ. 1) THEN
C             Matrix is real symmetric:
              DO J=1,NC
                 JCOL = J + JCOL1 - 1
                 AJ = DFLOAT(JCOL)
                 DO I=1,NR
```

```
                    IROW = I + IROW1 - 1
                    AI = DFLOAT(IROW)
                    IF(JCOL .LE. IROW) THEN
                       A(I,J) = SCALE*AI + AJ
                    ELSE
                       A(I,J) = SCALE*AJ + AI
                    END IF
                 END DO
              END DO
           ELSE
C             Matrix is real nonsymmetric:
              DO J=1,NC
                 JCOL = J + JCOL1 - 1
                 AJ = DFLOAT(JCOL)
                 DO I=1,NR
                    IROW = I + IROW1 - 1
                    AI = SCALE*DFLOAT(IROW)
                    A(I,J) = AI + AJ
                 END DO
              END DO
           END IF
        END IF
        CALL FMSPOP(MYNAME)
        RETURN
        END
C=======================================================================
        SUBROUTINE CMFILL (A, NR, NC, IROW1, JCOL1, NUMEQ)
C=======================================================================
C       Fill a matrix block with COMPLEX*16 data:
        COMPLEX*16 A(NR,NC)
        INTEGER NR
        INTEGER NC
        INTEGER IROW1
        INTEGER JCOL1
        INTEGER NUMEQ
        INTEGER I, J, IROW, JCOL
        REAL*8  RN, SCALE, A_Re, A_Im
        COMPLEX*16 ZERO, DIA, OFFDIA, A11
```

```
      DATA       ZERO/( 0.0D0, 0.0D0)/
      CHARACTER*6 MYNAME
      PARAMETER (MYNAME='CMFILL')
      COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
      REAL*8 ERROR
C
      CALL FMSPSH(MYNAME)
C     Get MDATAU:
      CALL FMSIGT ('MDATAU', MDATAU)
C
      IF(MDATAU .EQ. 0) THEN
C        Fill the block with test data:
         DIA    = ( 1.0D0, 1.0D0)
         OFFDIA = (-1.0D0,-1.0D0)
         RN     = DFLOAT(NUMEQ)
         IF(MOD .EQ. 3) THEN
C           Hermitian test data:
            DIA    = ( 1.0D0, 0.0D0)
            OFFDIA = (-1.0D0, 0.0D0)
            A11    = DCMPLX(RN,0.0D0)
         ELSE
C           Complex symmetric or nonsymmetric test data:
            DIA    = ( 1.0D0, 1.0D0)
            OFFDIA = (-1.0D0,-1.0D0)
            A11    = DCMPLX(RN,RN)
         END IF
C
C        Zero out [A]
         DO I=1,NR
            DO J=1,NC
               A(I,J) = ZERO
            END DO
         END DO
C
C        Set first column to (-1,-1):
         IF(JCOL1 .EQ. 1) THEN
            DO I=1,MIN0(NR,NUMEQ-IROW1+1)
               A(I,1) = OFFDIA
            END DO
```

```
            END IF
C
C           Set first row to (-1,-1):
            IF(IROW1 .EQ. 1) THEN
               DO J=1,MIN0(NC,NUMEQ-JCOL1+1)
                  A(1,J) = OFFDIA
               END DO
            END IF
C
C           Set diagonal to (1,1):
            IDIA1 = MAX0(IROW1,JCOL1)
            IDIA2 = MIN0(IROW1+NR-1,JCOL1+NC-1,NUMEQ)
            IF(IDIA1 .LE. IDIA2) THEN
               DO IDIA = IDIA1,IDIA2
                  A(IDIA-IROW1+1,IDIA-JCOL1+1) = DIA
               END DO
            END IF
C
C           Set A(1,1) to (NUMEQ,NUMEQ)
            IF( (IROW1 .EQ. 1) .AND.
     1          (JCOL1 .EQ. 1) ) THEN
               A(1,1) = A11
            END IF
         ELSE
C
C           Fill the block with (row,column) numbers:
            IF(MOD .EQ. 3) THEN
C              Hermitian test data; Real diagonals.
               SCALE = 10**ISCALE
               DO J=1,NC
                  JCOL = J + JCOL1 - 1
                  A_Im = DFLOAT(ISIGN*JCOL)
                  DO I=1,NR
                     IROW = I + IROW1 - 1
                     A_Re = DFLOAT(ISIGN*IROW)
                     IF(JCOL .LT. IROW) THEN
C                       This term is in the lower triangle:
                        A(I,J) = DCMPLX(A_Re,A_Im)
                     ELSE IF(JCOL .EQ. IROW) THEN
```

```
C                     This term is on the diagonal:
                      RN = SCALE * A_Re + A_Im
                      A(I,J) = DCMPLX(RN,0.0D0)
                   ELSE
C                     This term is in the upper triangle:
                      A(I,J) = DCMPLX(A_Im,-A_Re)
                   END IF
                END DO
             END DO
          ELSE IF(MOD .EQ. 4) THEN
C            Complex Symmetric:
             DO J=1,NC
                JCOL = J + JCOL1 - 1
                A_Im = DFLOAT(ISIGN*JCOL)
                DO I=1,NR
                   IROW = I + IROW1 - 1
                   A_Re = DFLOAT(ISIGN*IROW)
                   IF(JCOL .LE. IROW) THEN
                      A(I,J) = DCMPLX(A_Re,A_Im)
                   ELSE
                      A(I,J) = DCMPLX(A_Im,A_Re)
                   END IF
                END DO
             END DO
          ELSE
C            Complex Nonsymmetric:
             DO J=1,NC
                JCOL = J + JCOL1 - 1
                A_Im = DFLOAT(ISIGN*JCOL)
                DO I=1,NR
                   IROW = I + IROW1 - 1
                   A_Re = DFLOAT(ISIGN*IROW)
                   A(I,J) = DCMPLX(A_Re,A_Im)
                END DO
             END DO
          END IF
       END IF
       CALL FMSPOP(MYNAME)
       RETURN
```

```
        END
C=======================================================================
        SUBROUTINE RCHECK (A, NR, NC, IROW1, JCOL1)
C=======================================================================
C       Check the matrix block:
        REAL*8  A(NR,NC)
        INTEGER NR
        INTEGER NC
        INTEGER IROW1
        INTEGER JCOL1
        INTEGER I, J, IROW, JCOL
        REAL*8  SCALE, ATEST
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='RCHECK')
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
        CALL FMSPSH(MYNAME)
        SCALE = 10**ISCALE
        SCALE = ISIGN * SCALE
        DO I=1,NR
           IROW = I + IROW1 - 1
           AI = DFLOAT(IROW)
           DO J=1,NC
              JCOL = J + JCOL1 - 1
              AJ = DFLOAT(JCOL)
              IF(JCOL .LE. IROW) THEN
                 ATEST = SCALE*AI + AJ
              ELSE
                 IF(MOD .EQ. 1) THEN
                    ATEST = SCALE*AJ + AI
                 ELSE
                    ATEST = SCALE*AI + AJ
                 END IF
              END IF
           IF(A(I,J) .NE. ATEST) CALL RERROR (IROW, JCOL, A(I,J), ATEST)
           END DO
        END DO
        CALL FMSPOP(MYNAME)
```

```
        RETURN
        END
C=======================================================================
        SUBROUTINE CCHECK (A, NR, NC, IROW1, JCOL1)
C=======================================================================
C       Fill a matrix block with COMPLEX*16 data:
        COMPLEX*16 A(NR,NC)
        INTEGER NR
        INTEGER NC
        INTEGER IROW1
        INTEGER JCOL1
        INTEGER I, J, IROW, JCOL
        REAL*8  RN, SCALE, A_Re, A_Im
        COMPLEX*16 ATEST
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='CCHECK')
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
        CALL FMSPSH(MYNAME)
        SCALE = 10**ISCALE
        DO I=1,NR
           IROW = I + IROW1 - 1
           A_Re = DFLOAT(ISIGN*IROW)
           DO J=1,NC
              JCOL = J + JCOL1 - 1
              A_Im = DFLOAT(ISIGN*JCOL)
              IF(JCOL .LT. IROW) THEN
                 ATEST = DCMPLX(A_Re,A_Im)
              ELSE IF(JCOL .EQ. IROW) THEN
                 IF(MOD .EQ. 3) THEN
                    RN    = SCALE * A_Re + A_Im
                    ATEST = DCMPLX(RN,0.0D0)
                 ELSE
                    ATEST = DCMPLX(A_Re,A_Im)
                 END IF
              ELSE
                 IF(MOD .EQ. 3) THEN
                    ATEST = DCMPLX(A Im,-A Re)
```

```
                ELSE IF(MOD .EQ. 4) THEN
                   ATEST = DCMPLX(A_Im,A_Re)
                ELSE
                   ATEST = DCMPLX(A_Re,A_Im)
                END IF
             END IF
          IF(A(I,J) .NE. ATEST) CALL CERROR (IROW,JCOL,A(I,J),ATEST)
          END DO
       END DO
       CALL FMSPOP(MYNAME)
       RETURN
       END
C=======================================================================
       LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
       CHARACTER* (*) QUESTION
       CHARACTER*1 IYN
       WRITE(6,2000) QUESTION
       READ (5,1000) IYN
       IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
          ASK = .TRUE.
       ELSE
          ASK = .FALSE.
       END IF
       RETURN
 1000  FORMAT (A)
 2000  FORMAT (1X,A,' (y,n)>')
       END
C=======================================================================
       INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
       CHARACTER* (*) STRING
       WRITE(6,2000) STRING
       READ (5,*) ASK_I
       RETURN
 2000  FORMAT (1X,A,'>')
       END
C=======================================================================
```

```
        INTEGER FUNCTION INTINC (I)
C-------------------------------------------------------------------------
C       This function increments a volatile shared variable.  It is
C       placed in a subroutine to prevent some compilers from storing
C       the value in a register and not updating it.
        INTEGER I
        I = I + 1
        INTINC = I
        RETURN
        END
C=========================================================================
        SUBROUTINE RERROR (IROW, JCOL, AIJ, AVALUE)
C-------------------------------------------------------------------------
C       This subroutine prints a real matrix term that is incorrect.
        INTEGER IROW, JCOL
        REAL*8  AIJ, AVALUE
        CHARACTER*55 FMT
        DATA FMT( 1:36)/'(''CPU___:A('',I_,'','',I_,'')='',F__.0,'';'/
        DATA FMT(37:55)/' Should be '',F__.0)'/
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
        INTEGER MYNODE
        INTEGER NERRORS
        DATA    NERRORS/0/
        CALL FMSIGT ('MYNODE', MYNODE)
        ISCALE2 = MAX0(2*ISCALE,10)
        WRITE(FMT( 6: 8), 2003) MYNODE
        WRITE(FMT(15:15), 2001) ISCALE
        WRITE(FMT(22:22), 2001) ISCALE
        WRITE(FMT(30:31), 2002) ISCALE2
        WRITE(FMT(51:52), 2002) ISCALE2
        IF(ERROR .EQ. 0.0D0) ERROR = AVALUE
        WRITE(6, FMT) IROW, JCOL, AIJ, AVALUE
        NERRORS = NERRORS + 1
        IF(NERRORS .GT. 100) STOP 'Fatal error checking Matrix Data'
 2001   FORMAT (I1)
 2002   FORMAT (I2)
 2003   FORMAT (I3.3)
        RETURN
```

```
        END
C=======================================================================
        SUBROUTINE CERROR (IROW, JCOL, AIJ, AVALUE)
C-----------------------------------------------------------------------
C       This subroutine prints a real matrix term that is incorrect.
        INTEGER IROW, JCOL
        COMPLEX*16  AIJ, AVALUE
        CHARACTER*82 FMT
        DATA FMT( 1:34)/'(''CPU___:A('',I_,'','',I_,'')=('',F__.0'/
        DATA FMT(35:48)/','','',F__.0,'');'/
        DATA FMT(49:82)/'_Should_be_('',F__.0,'','',F__.0,'')'')'/
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
        INTEGER MYNODE
        INTEGER NERRORS
        DATA    NERRORS/0/
        CALL FMSIGT ('MYNODE', MYNODE)
        ISCALE1 = ISCALE + 1
        WRITE(FMT( 6: 8), 2003) MYNODE
        WRITE(FMT(15:15), 2001) ISCALE
        WRITE(FMT(22:22), 2001) ISCALE
        WRITE(FMT(31:32), 2002) ISCALE1
        WRITE(FMT(41:42), 2002) ISCALE1
        WRITE(FMT(64:65), 2002) ISCALE1
        WRITE(FMT(74:75), 2002) ISCALE1
        IF(ERROR .EQ. 0.0D0) ERROR = AVALUE
        WRITE(6, FMT) IROW, JCOL, AIJ, AVALUE
        NERRORS = NERRORS + 1
        IF(NERRORS .GT. 100) STOP 'Fatal error checking Matrix Data'
 2001   FORMAT (I1)
 2002   FORMAT (I2)
 2003   FORMAT (I3.3)
        RETURN
        END
C       =======================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       =======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
```

```
      INTEGER    LOWEQ(*), LOCEQ(*)
      REAL*8     A(0:*), D(*), AIJ, ATEST, SCALE
      COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
      REAL*8 ERROR
C
C     Test the matrix data:
      SCALE = 10**ISCALE
      DO IROW = IROW1,IROW2
         JSTART = MAX0(LOWEQ(IROW), JCOL1)
         JEND   = MIN0(IROW,        JCOL2)
         LOCI   = LOCEQ(IROW)
         DO JCOL = JSTART, JEND
            IF(JCOL .EQ. IROW) THEN
C              This term is on the diagonal:
               AIJ   = D(IROW)
            ELSE
C              This term is on or within the profile:
               AIJ   = A(LOCI + IJSTEP*JCOL)
            END IF
            ATEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
            IF(AIJ .NE. ATEST) CALL RERROR (IROW, JCOL, AIJ, ATEST)
         END DO
      END DO
      RETURN
      END

C     ================================================================
      SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                   JCOL1, JCOL2, IJSTEP)
C     ================================================================
      INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
      INTEGER    LOWEQ(*), LOCEQ(*)
      REAL*8     A(0:*), D(*), AIJ, ATEST, SCALE
      COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
      REAL*8 ERROR
C
C     Test the matrix data:
      SCALE = 10**ISCALE
      DO JCOL = JCOL1,JCOL2
```

```
            ISTART = MAX0(LOWEQ(JCOL), IROW1)
            IEND   = MIN0(JCOL-1,      IROW2)
            LOCJ   = LOCEQ(JCOL)
            DO IROW = ISTART, IEND
               ATEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
               AIJ   = A(LOCJ + IJSTEP*IROW)
               IF(AIJ .NE. ATEST) CALL RERROR (IROW, JCOL, AIJ, ATEST)
            END DO
         END DO
         RETURN
         END

C        ================================================================
         SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     D(*), DII, DTEST, SCALE
         COMPLEX*16 A(0:*), AIJ, ATEST
         COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
         REAL*8 ERROR
C
C        Test the matrix data:
         SCALE = 10**ISCALE
         DO IROW = IROW1,IROW2
            JSTART = MAX0(LOWEQ(IROW), JCOL1)
            JEND   = MIN0(IROW,        JCOL2)
            LOCI   = LOCEQ(IROW)
            DO JCOL = JSTART, JEND
               IF(JCOL .EQ. IROW) THEN
C                 This term is on the diagonal:
                  DII   = D(IROW)
                  DTEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
                  IF(DII .NE. DTEST) CALL RERROR (IROW, JCOL, DII, DTEST)
               ELSE
C                 This term is on or within the profile:
                  AIJ   = A(LOCI + IJSTEP*JCOL)
                  IF(JCOL .LT. IROW) THEN
```

```
C                  This term is in the lower triangle:
                   ATEST = DCMPLX(IROW,JCOL)
                ELSE
C                  This term is in the upper triangle:
c                  Use complex conjugate:
                   ATEST = DCMPLX(JCOL,-IROW)
                END IF
                IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
             END IF
          END DO
       END DO
       RETURN
       END

C      ================================================================
       SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                    JCOL1, JCOL2, IJSTEP)
C      ================================================================
       INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
       INTEGER    LOWEQ(*), LOCEQ(*)
       COMPLEX*16 A(0:*), D(*), AIJ, ATEST
       COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
       REAL*8 ERROR
C
C      Test the matrix data:
       DO IROW = IROW1,IROW2
          JSTART = MAX0(LOWEQ(IROW), JCOL1)
          JEND   = MIN0(IROW,        JCOL2)
          LOCI   = LOCEQ(IROW)
          DO JCOL = JSTART, JEND
             IF(JCOL .EQ. IROW) THEN
C               This term is on the diagonal:
                AIJ   = D(IROW)
             ELSE
C               This term is on or within the profile:
                AIJ   = A(LOCI + IJSTEP*JCOL)
             END IF
             ATEST = DCMPLX(IROW,JCOL)
             IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
```

```
            END DO
         END DO
         RETURN
         END

C        ================================================================
         SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         COMPLEX*16 A(0:*), D(*), AIJ, ATEST
C
C        Test the matrix data:
         DO JCOL = JCOL1,JCOL2
            ISTART = MAX0(LOWEQ(JCOL), IROW1)
            IEND   = MIN0(JCOL-1,      IROW2)
            LOCJ   = LOCEQ(JCOL)
            DO IROW = ISTART, IEND
               AIJ   = A(LOCJ + IJSTEP*IROW)
               ATEST = DCMPLX(IROW,JCOL)
               IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
            END DO
         END DO
         RETURN
         END

C        ================================================================
         SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         REAL*8     A(0:*), AIJ, ATEST, SCALE
         COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
         REAL*8 ERROR
C
C        Test the matrix data:
         SCALE = 10**ISCALE
```

```
            DO JCOL = JEQN1,JEQN2
               DO IROW = 1,NUMEQ
                  AIJ   = A(LOCI(IROW) + LOCJ(1,LUFLAG(IROW)))
                  ATEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
                  IF(AIJ .NE. ATEST) CALL RERROR (IROW, JCOL, AIJ, ATEST)
               END DO
            END DO
            RETURN
            END

C           ================================================================
            SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C           ================================================================
            INTEGER    JEQN1, JEQN2, NUMEQ
            INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
            COMPLEX*16 A(0:*), AIJ, ATEST
C
C           Test the matrix data:
            DO JCOL = JEQN1,JEQN2
               DO IROW = 1,NUMEQ
                  AIJ    = A(LOCI(IROW) + LOCJ(1,LUFLAG(IROW)))
                  ATEST = DCMPLX(IROW,JCOL)
                  IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
               END DO
            END DO
            RETURN
            END
```

Copyright © Multipath Corporation

```
C       E X A M P L E  13
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_13')
C
C       Subroutines called in parallel:
        EXTERNAL WRITEM
        EXTERNAL WRITEV
C
C       FMS Module number (1=RS, 2=RN, 3=CH, 4=CS, 5=CN)
        INTEGER MOD
C
C       Number of equations:
        INTEGER N
C
C       Number of R.H.S. vectors:
        INTEGER NRHS
C
C       Size of block to fill matrix:
        INTEGER NROWS, NCOLS
C
C       Problem data:
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
C       Work queue index (shared):
        COMMON/SHARE/IWORK
        INTEGER IWORK
C
C       FMS matrix and vector file attributes:
        INTEGER     LUA(25), LUF(25), LUA0(25)
        DATA LUA0(1)/0/
        INTEGER     LUX(25)
        INTEGER     LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       FMS Parameters:
        INTEGER     MDATAU
C
C       Profile vector for a full matrix:
        INTEGER     LOWEQ(1)
        DATA        LOWEQ/-1/
C
C       Local variables:
        INTEGER     MAXCPU
        INTEGER     IDTYPE
        INTEGER     ISTYPE
        INTEGER     I, LX, LDISK, NV
        REAL*8      EI
        LOGICAL     ITEST
C
```

```
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       Constants:
        REAL*8     RALPHA(1)
        DATA RALPHA/1.0D0/
        COMPLEX*16 CALPHA(1)
        DATA CALPHA/(1.0D0,0.0D0)/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('IPRF'  , 1026)
    1   CONTINUE
        WRITE (6,*) 'Enter the FMS module number (1 to 5)'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        READ  (5,*) MOD
        IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
        N    = ASK_I('Enter the number of equations')
        ISCALE = 1
        DO I=1,15
           IF(10**ISCALE .GT. N) GO TO 11
           ISCALE = ISCALE + 1
        END DO
   11   CONTINUE
        NRHS = ASK_I('Enter the number of solution vectors')
        NROWS= ASK_I('Enter the number of rows per block (0=NEQBIO)')
        NCOLS= ASK_I('Enter the number of columns per block (0=NEQBIO)')
        IF(ASK('Do you want to check the matrix data?')) THEN
           MDATAU = 2
        ELSE
           MDATAU = 0
        END IF
        CALL FMSIST ('MDATAU', MDATAU)
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        CALL FMSIGT ('MAXCPU', MAXCPU)
        IF(MOD .LE. 2) THEN
           IDTYPE = 1
        ELSE
           IDTYPE = 2
        END IF
        ISTYPE = 1
        IF(MOD .EQ. 2) ISTYPE = 2
        IF(MOD .EQ. 3) ISTYPE = 3
        IF(MOD .EQ. 5) ISTYPE = 2
C
C (2)   Open FMS files:
C       CALL FMSIST ('IOKIDS', MAXCPU-1)
        IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
```

```
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C (3)    Write data to FMS files:
C
C        Fill the RHS vectors in parallel:
C        Initialize FMSPUT:
         CALL FMSPUT (LUX, 0, 0, 0, 0, RMD, 1)
C        Generate and write the vector data in parallel:
         IWORK  = 0
         DO 30 ICPU=2,MAXCPU
            CALL FMSPAR (1, WRITEV, LUX)
   30    CONTINUE
         IF(MAXCPU .GT. 1) CALL FMSRUN
         CALL WRITEV (LUX)
         IF(MAXCPU .GT. 1) CALL FMSYNC
C
C        End FMSPUT:
            CALL FMSPUT (LUX, 0, 0, N+1, 0, RMD, 1)
C
C        If testing the matrix fill, first fill with one processor:
         IF(MDATAU .NE. 0) THEN
            IF(ISTYPE .EQ. 2) THEN
C              This matrix is nonsymmetric:
               CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
            ELSE
C              This matrix is symmetric or Hermitian:
C              Initialize FMSPUT to define the matrix terms in the upper
C              triangle.
               CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
            END IF
            ISIGN  = -1
            IWORK  = 0
            CALL WRITEM (NROWS, NCOLS, LUA)
C           End FMSPUT:
            CALL FMSPUT (LUA, 0, 0, N+1, 0, RMD, 1)
         END IF
C
C        Fill the matrix in parallel:
C
C        Initialize FMSPUT:
         IF(ISTYPE .EQ. 2) THEN
C           This matrix is nonsymmetric:
            CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
         ELSE
C           This matrix is symmetric or Hermitian:
C           Initialize FMSPUT to define the matrix terms in the upper
C           triangle.
            CALL FMSPUT (LUA, 0, 0, 0, 0, RMD, 1)
         END IF
         ISIGN  = 1
         IWORK  = 0
         DO 35 ICPU=2,MAXCPU
            CALL FMSPAR (3, WRITEM, NROWS, NCOLS, LUA)
   35    CONTINUE
         IF(MAXCPU .GT. 1) CALL FMSRUN
         CALL WRITEM (NROWS, NCOLS, LUA)
         IF(MAXCPU .GT. 1) CALL FMSYNC
C
C        End FMSPUT:
         CALL FMSPUT (LUA, 0, 0, N+1, 0, RMD, 1)
C
C (4)    Perform matrix algebra:
C
C        This example offers two testing options:
C        MDATAU=0; Factor the matrix with valid test data;
```

```
C       MDATAU=2; Check the matrix data and quit.
        ERROR = 0.0D0
        IF(MDATAU .EQ. 0) THEN
           DO I=1,25
              LUF(I) = LUA(I)
           END DO
        ELSE
           LUF(1) = 0
        END IF
        IF(MOD.EQ.1) CALL RSDAF
     1   (LUA, RALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,  NUMRED)
        IF(MOD.EQ.2) CALL RNDAF
     1   (LUA, RALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,  NUMRED)
        IF(MOD.EQ.3) CALL CHDAF
     1   (LUA, RALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,  NUMRED)
        IF(MOD.EQ.4) CALL CSDAF
     1   (LUA, CALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,  NUMRED)
        IF(MOD.EQ.5) CALL CNDAF
     1   (LUA, CALPHA, 1, LUS0, 0, LUA0, LUF, LUX, LUX,  NUMRED)
C
        IF(MDATAU .EQ. 2) GO TO 70
        IF(MOD.EQ.1) CALL RSDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.2) CALL RNDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.3) CALL CHDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.4) CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
        IF(MOD.EQ.5) CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C (5)   Read data from FMS files:
C       Check the answer:
C
C       Allocate a vector for reading {X}:
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMG (RMD, LX, N)
        ELSE
           CALL FMSCMG (CMD, LX, N)
        END IF
        LDISK  = 1
        DO 60 NV = 1,NRHS
           IF(IDTYPE .EQ. 1) THEN
              CALL FMSRED (LUX(1), LDISK, RMD(LX), N)
              DO 50 I = 1,N
                 EI = ABS(RMD(LX-1+I) - 1.0D0)
                 IF(EI .GT. ERROR) ERROR = EI
   50         CONTINUE
           ELSE
              CALL FMSRED (LUX(1), LDISK, CMD(LX), 2*N)
              DO 51 I = 1,N
                 EI = ABS(CMD(LX-1+I) - 1.0D0)
                 IF(EI .GT. ERROR) ERROR = EI
   51         CONTINUE
           END IF
           LDISK  = LDISK  + LUX(4)
   60   CONTINUE
   70   CONTINUE
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)   Close FMS files:
        CALL FMSCM (LUA)
        CALL FMSCV (LUX)
        IF(MDATAU .EQ. 0) THEN
           IF(IDTYPE .EQ. 1) THEN
              CALL FMSRMR (RMD, LX, N)
           ELSE
              CALL FMSCMR (CMD, LX, N)
```

```
            END IF
         END IF
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP (MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE WRITEV (LUX)
C=======================================================================
C
C        DESCRIPTION:
C           This subroutine runs in parallel. For each vector
C           that it finds, it fills in the terms and calls FMSPUT to
C           write the vector to the vector file LUX.
C
C        FORMAL PARAMETERS:
C           (R ) LUX(25) = FMS vector file attributes
C-----------------------------------------------------------------------
C
C        Formal Parameters:
         INTEGER    LUX(25)
C
C        Local Variables:
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='WRITEV')
         INTEGER    MYNODE
         INTEGER    LUPR
         INTEGER    NUMEQ
         INTEGER    NUMVEC
         INTEGER    NROWS, NCOLS
         INTEGER    IEQN1, IEQN2, JEQN1, JEQN2
         INTEGER    MY_WORK
         INTEGER    MY_TOTAL
         INTEGER    LX
         REAL*8     RMD(0:1)
         COMPLEX*16 CMD(0:1)
         POINTER    (RMD_PTR,RMD)
         POINTER    (CMD_PTR,CMD)
         COMMON/SHARE/IWORK
         INTEGER    PARENT
         PARAMETER (PARENT = 0)
C
         CALL FMSPSH (MYNAME)
C        Get your CPU number (0 to MAXCPU-1)
         CALL FMSIGT ('MYNODE', MYNODE)
         CALL FMSIGT ('LUPR',   LUPR  )
         NUMEQ  = LUX( 3)
         LENVEC = LUX( 4)
         IDTYPE = LUX( 5)
         NUMVEC = LUX( 6)
         NROWS  = NUMEQ
         NCOLS  = 1
         IEQN1  = 1
         IEQN2  = NUMEQ
         LDX    = LENVEC/IDTYPE
C
C        Allocate a work array for storing the data you generate:
C        For this example, we will fill [B] vector by vector.
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSIGT ('MEMPTR', RMD_PTR)
            CALL FMSRMG (RMD, LX, NUMEQ)
         ELSE
            CALL FMSIGT ('MEMPTR', CMD_PTR)
            CALL FMSCMG (CMD, LX, NUMEQ)
         END IF
```

```
C
C       Find the next vector to fill:
        MY_TOTAL = 0
   10   CONTINUE
           CALL FMSONE
C             IWORK   = IWORK + 1
C             MY_WORK = IWORK
              MY_WORK = INTINC(IWORK)
C             Write message under protection of lock:
              IF(MY_WORK .LE. NUMVEC) WRITE(LUPR,2001) MYNODE, MY_WORK
           CALL FMSALL
           IF(MY_WORK .GT. NUMVEC) GO TO 100
           MY_TOTAL = MY_TOTAL + 1
           JEQN1  = MY_WORK
           JEQN2  = JEQN1
C
           IF(IDTYPE .EQ. 1) THEN
              CALL RVFILL (RMD(LX), IEQN1, IEQN2, JEQN1, JEQN2, NUMEQ)
              CALL FMSPUT (LUX, NROWS, NCOLS, IEQN1, JEQN1, RMD(LX),
     1                     LDX)
           ELSE
              CALL CVFILL (CMD(LX), IEQN1, IEQN2, JEQN1, JEQN2, NUMEQ)
              CALL FMSPUT (LUX, NROWS, NCOLS, IEQN1, JEQN1, CMD(LX),
     1                     LDX)
           END IF
        GO TO 10
  100   CONTINUE
C       Your part of the work is done:
C
C       Return work array:
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSRMR (RMD, LX, NUMEQ)
        ELSE
           CALL FMSCMR (CMD, LX, NUMEQ)
        END IF
C
C       Report your total work:
        CALL FMSONE
           WRITE(LUPR,2000) MYNODE, MY_TOTAL
        CALL FMSALL
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Process',I3,' computed',I5,' vectors.')
 2001   FORMAT (' Process',I3,' is computing vector ',I5)
        END
C=======================================================================
        SUBROUTINE WRITEM (NROWS, NCOLS, LUA)
C=======================================================================
C
C       DESCRIPTION:
C          This subroutine runs in parallel. For each block
C          A(NROWS, NCOLS) that it finds, it fills in the terms and
C          calls FMSPUT to write the block to the matrix file LUA.
C
C       FORMAL PARAMETERS:
C          (R ) NROWS   = Number of rows per block
C          (R ) NCOLS   = Number of columns per block
C          (R ) LUA(25) = FMS matrix file attributes
C-----------------------------------------------------------------------
C
C       Formal Parameters:
        INTEGER    NROWS
        INTEGER    NCOLS
        INTEGER    LUA(25)
C
```

```
C       Local Variables:
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='WRITEM')
        INTEGER    NEQROW, NEQCOL
        INTEGER    NBKROW, NBKCOL
        INTEGER    IEQN1, IEQN2, JEQN1, JEQN2
        INTEGER    IBLOCK, JBLOCK
        INTEGER    NUMBLK
        INTEGER    MYNODE
        INTEGER    MY_WORK
        INTEGER    MY_TOTAL
        INTEGER    LUPR
        INTEGER    NUMEQ
        INTEGER    NEQBIO
        INTEGER    NUMSEG
        INTEGER    IDTYPE
        INTEGER    LA
        REAL*8     RMD(0:1)
        COMPLEX*16 CMD(0:1)
        POINTER    (RMD_PTR,RMD)
        POINTER    (CMD_PTR,CMD)
        COMMON/SHARE/IWORK
        INTEGER    PARENT
        PARAMETER (PARENT = 0)
C
        CALL FMSPSH (MYNAME)
C       Get your CPU number (0 to MAXCPU-1)
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('LUPR',   LUPR  )
        NUMEQ  = LUA( 8)
        NUMSEG = LUA( 6)
        IDTYPE = LUA(11)
        ISTYPE = LUA(12)
        MFMAT  = LUA(13)
        NEQBIO = LUA(21)
        NEQROW = NROWS
        NEQCOL = NCOLS
        IF(NEQROW .EQ. 0) THEN
           IF(MFMAT .EQ. 1) THEN
              NEQROW = NUMEQ/5
           ELSE
              NEQROW = NEQBIO
           END IF
        END IF
        IF(NEQCOL .EQ. 0) THEN
           IF(MFMAT .EQ. 1) THEN
              NEQCOL = NUMEQ/5
           ELSE
              NEQCOL = NEQBIO
           END IF
        END IF
        NBKROW = (NUMEQ+NEQROW-1)/NEQROW
        NBKCOL = (NUMEQ+NEQCOL-1)/NEQCOL
        NUMBLK = NBKROW*NBKCOL
        LDA    = NEQROW
C
C       Allocate a work array for storing the data you generate:
        CALL FMSIGT ('MDATAU', MDATAU)
        IF(IDTYPE .EQ. 1) THEN
           CALL FMSIGT ('MEMPTR', RMD_PTR)
           CALL FMSRMG (RMD, LA, NEQROW*NEQCOL)
        ELSE
           CALL FMSIGT ('MEMPTR', CMD_PTR)
           CALL FMSCMG (CMD, LA, NEQROW*NEQCOL)
```

```
         END IF
C
C        Fill the matrix:
         MY_TOTAL = 0
   10    CONTINUE
            CALL FMSONE
C              IWORK   = IWORK + 1
C              MY_WORK = IWORK
               MY_WORK = INTINC(IWORK)
C              Write message under protection of lock:
               JBLOCK = (MY_WORK + NBKROW - 1)/NBKROW
               IBLOCK =  MY_WORK - NBKROW*(JBLOCK-1)
               IF(MY_WORK .LE. NUMBLK)
     1            WRITE(LUPR,2001) MYNODE, IBLOCK, JBLOCK
            CALL FMSALL
            IF(MY_WORK .GT. NUMBLK) GO TO 100
            MY_TOTAL = MY_TOTAL + 1
C
            IEQN1 = 1 + NEQROW*(IBLOCK-1)
            JEQN1 = 1 + NEQCOL*(JBLOCK-1)
            IEQN2 = IEQN1 + NEQROW - 1
            IEQN2 = MIN0(IEQN2,NUMEQ)
            JEQN2 = JEQN1 + NEQCOL - 1
            JEQN2 = MIN0(JEQN2,NUMEQ)
            NR    = IEQN2 - IEQN1 + 1
            NC    = JEQN2 - JEQN1 + 1
            IF(ISTYPE .NE. 2) THEN
C              This problem is symmetric.
C              Skip blocks entirely in [L]:
               IF(IEQN1 .GT. JEQN2) GO TO 10
            END IF
            IF(IDTYPE .EQ. 1) THEN
               CALL RMFILL (RMD(LA), NR, NC, IEQN1, JEQN1, NUMEQ)
               IF(MDATAU .GT. 0)
     1         CALL RCHECK (RMD(LA), NR, NC, IEQN1, JEQN1)
               CALL FMSPUT (LUA,  NR, NC, IEQN1, JEQN1, RMD(LA), NR)
            ELSE
               CALL CMFILL (CMD(LA), NR, NC, IEQN1, JEQN1, NUMEQ)
               IF(MDATAU .GT. 0)
     1         CALL CCHECK (CMD(LA), NR, NC, IEQN1, JEQN1)
               CALL FMSPUT (LUA,  NR, NC, IEQN1, JEQN1, CMD(LA), NR)
            END IF
         GO TO 10
  100    CONTINUE
C        Your part of the work is done:
C
C        Return work array:
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LA, NEQROW*NEQCOL)
         ELSE
            CALL FMSCMR (CMD, LA, NEQROW*NEQCOL)
         END IF
C
C        Report your total work:
         CALL FMSONE
            WRITE(LUPR,2000) MYNODE, MY_TOTAL
         CALL FMSALL
         CALL FMSPOP (MYNAME)
         RETURN
 2000    FORMAT (' Process',I3,' computed',I5,' blocks.')
 2001    FORMAT (' Process',I3,' is computing block (',I3,',',I3,')')
         END
C=======================================================================
         SUBROUTINE RVFILL (X, IROW1, IROW2, JCOL1, JCOL2, NUMEQ)
C=======================================================================
```

```
C       Fill a test vector with REAL*8 data:
        REAL*8  X(IROW1:IROW2,JCOL1:JCOL2)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='RVFILL')
        INTEGER IROW1
        INTEGER IROW2
        INTEGER JCOL1
        INTEGER JCOL2
        REAL*8  ZERO, X1
        DATA    ZERO/0.0D0/
        DATA    X1  /1.0D0/
C
        CALL FMSPSH (MYNAME)
C       Zero out [X]
        DO IROW=IROW1,IROW2
           DO JCOL=JCOL1,JCOL2
              X(IROW,JCOL) = ZERO
           END DO
        END DO
C
C       Set first row to X1:
        IF(IROW1 .EQ. 1) THEN
           DO JCOL=JCOL1,JCOL2
              X(1,JCOL) = X1
           END DO
        END IF
        CALL FMSPOP (MYNAME)
        RETURN
        END
C=======================================================================
        SUBROUTINE CVFILL (X, IROW1, IROW2, JCOL1, JCOL2, NUMEQ)
C=======================================================================
C       Fill a test vector with COMPLEX*16 data:
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='CVFILL')
        COMPLEX*16 X(IROW1:IROW2,JCOL1:JCOL2)
        INTEGER IROW1
        INTEGER IROW2
        INTEGER JCOL1
        INTEGER JCOL2
        COMPLEX*16 ZERO, X1
        DATA       ZERO/(0.0D0, 0.0D0)/
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
        CALL FMSPSH (MYNAME)
        IF(MOD .EQ. 3) THEN
           X1 = (1.0D0,0.0D0)
        ELSE
           X1 = (1.0D0,1.0D0)
        END IF
C
C       Zero out [X]
        DO IROW=IROW1,IROW2
           DO JCOL=JCOL1,JCOL2
              X(IROW,JCOL) = ZERO
           END DO
        END DO
C
C       Set first row to X1:
        IF(IROW1 .EQ. 1) THEN
           DO JCOL=JCOL1,JCOL2
              X(1,JCOL) = X1
           END DO
        END IF
```

```
        CALL FMSPOP (MYNAME)
        RETURN
        END
C=======================================================================
        SUBROUTINE RMFILL (A, NR, NC, IROW1, JCOL1, NUMEQ)
C=======================================================================
C       Fill a matrix block with REAL*8 data:
        REAL*8  A(NR,NC)
        INTEGER NR
        INTEGER NC
        INTEGER IROW1
        INTEGER JCOL1
        INTEGER I, J, IROW, JCOL
        REAL*8  OFFDIA, ZERO, DIA, AI, AJ, SCALE
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='RMFILL')
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
        DATA    OFFDIA/-1.0D0/
        DATA    ZERO  / 0.0D0/
        DATA    DIA   / 1.0D0/
C
        CALL FMSPSH (MYNAME)
C       Get MDATAU:
        CALL FMSIGT ('MDATAU', MDATAU)
C
        IF(MDATAU .EQ. 0) THEN
C          Fill the block with test data:
C          Zero out [A]
           DO I=1,NR
              DO J=1,NC
                 A(I,J) = ZERO
              END DO
           END DO
C
C          Set first column to OFFDIA:
           IF(JCOL1 .EQ. 1) THEN
              DO I=1,MIN0(NR,NUMEQ-IROW1+1)
                 A(I,1) = OFFDIA
              END DO
           END IF
C
C          Set first row to OFFDIA:
           IF(IROW1 .EQ. 1) THEN
              DO J=1,MIN0(NC,NUMEQ-JCOL1+1)
                 A(1,J) = OFFDIA
              END DO
           END IF
C
C          Set diagonal to DIA:
           IDIA1 = MAX0(IROW1,JCOL1)
           IDIA2 = MIN0(IROW1+NR-1,JCOL1+NC-1,NUMEQ)
           IF(IDIA1 .LE. IDIA2) THEN
              DO IDIA = IDIA1,IDIA2
                 A(IDIA-IROW1+1,IDIA-JCOL1+1) = DIA
              END DO
           END IF
C
C          Set A(1,1) to NUMEQ
           IF( (IROW1 .EQ. 1) .AND.
     1         (JCOL1 .EQ. 1) ) THEN
              A(1,1) = DFLOAT(NUMEQ)
           END IF
        ELSE
```

```
C
C           Fill the block with (row,column) numbers:
            SCALE = 10**ISCALE
            SCALE = ISIGN * SCALE
            IF(MOD .EQ. 1) THEN
C              Matrix is real symmetric:
               DO J=1,NC
                  JCOL = J + JCOL1 - 1
                  AJ = DFLOAT(JCOL)
                  DO I=1,NR
                     IROW = I + IROW1 - 1
                     AI = DFLOAT(IROW)
                     IF(JCOL .LE. IROW) THEN
                        A(I,J) = SCALE*AI + AJ
                     ELSE
                        A(I,J) = SCALE*AJ + AI
                     END IF
                  END DO
               END DO
            ELSE
C              Matrix is real nonsymmetric:
               DO J=1,NC
                  JCOL = J + JCOL1 - 1
                  AJ = DFLOAT(JCOL)
                  DO I=1,NR
                     IROW = I + IROW1 - 1
                     AI = SCALE*DFLOAT(IROW)
                     A(I,J) = AI + AJ
                  END DO
               END DO
            END IF
         END IF
         CALL FMSPOP (MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE CMFILL (A, NR, NC, IROW1, JCOL1, NUMEQ)
C=======================================================================
C        Fill a matrix block with COMPLEX*16 data:
         COMPLEX*16 A(NR,NC)
         INTEGER NR
         INTEGER NC
         INTEGER IROW1
         INTEGER JCOL1
         INTEGER NUMEQ
         INTEGER I, J, IROW, JCOL
         REAL*8  RN, SCALE, A_Re, A_Im
         COMPLEX*16 ZERO, DIA, OFFDIA, A11
         DATA       ZERO/( 0.0D0, 0.0D0)/
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='CMFILL')
         COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
         REAL*8 ERROR
C
         CALL FMSPSH (MYNAME)
C        Get MDATAU:
         CALL FMSIGT ('MDATAU', MDATAU)
C
         IF(MDATAU .EQ. 0) THEN
C           Fill the block with test data:
            DIA    = ( 1.0D0, 1.0D0)
            OFFDIA = (-1.0D0,-1.0D0)
            RN     = DFLOAT(NUMEQ)
            IF(MOD .EQ. 3) THEN
C              Hermitian test data:
```

```
               DIA    = ( 1.0D0, 0.0D0)
               OFFDIA = (-1.0D0, 0.0D0)
               A11    = DCMPLX(RN,0.0D0)
            ELSE
C              Complex symmetric or nonsymmetric test data:
               DIA    = ( 1.0D0, 1.0D0)
               OFFDIA = (-1.0D0,-1.0D0)
               A11    = DCMPLX(RN,RN)
            END IF
C
C           Zero out [A]
            DO I=1,NR
               DO J=1,NC
                  A(I,J) = ZERO
               END DO
            END DO
C
C           Set first column to (-1,-1):
            IF(JCOL1 .EQ. 1) THEN
               DO I=1,MIN0(NR,NUMEQ-IROW1+1)
                  A(I,1) = OFFDIA
               END DO
            END IF
C
C           Set first row to (-1,-1):
            IF(IROW1 .EQ. 1) THEN
               DO J=1,MIN0(NC,NUMEQ-JCOL1+1)
                  A(1,J) = OFFDIA
               END DO
            END IF
C
C           Set diagonal to (1,1):
            IDIA1 = MAX0(IROW1,JCOL1)
            IDIA2 = MIN0(IROW1+NR-1,JCOL1+NC-1,NUMEQ)
            IF(IDIA1 .LE. IDIA2) THEN
               DO IDIA = IDIA1,IDIA2
                  A(IDIA-IROW1+1,IDIA-JCOL1+1) = DIA
               END DO
            END IF
C
C           Set A(1,1) to (NUMEQ,NUMEQ)
            IF( (IROW1 .EQ. 1) .AND.
     1          (JCOL1 .EQ. 1) ) THEN
               A(1,1) = A11
            END IF
         ELSE
C
C           Fill the block with (row,column) numbers:
            IF(MOD .EQ. 3) THEN
C              Hermitian test data; Real diagonals.
               SCALE = 10**ISCALE
               DO J=1,NC
                  JCOL = J + JCOL1 - 1
                  A_Im = DFLOAT(ISIGN*JCOL)
                  DO I=1,NR
                     IROW = I + IROW1 - 1
                     A_Re = DFLOAT(ISIGN*IROW)
                     IF(JCOL .LT. IROW) THEN
C                       This term is in the lower triangle:
                        A(I,J) = DCMPLX(A_Re,A_Im)
                     ELSE IF(JCOL .EQ. IROW) THEN
C                       This term is on the diagonal:
                        RN = SCALE * A_Re + A_Im
                        A(I,J) = DCMPLX(RN,0.0D0)
                     ELSE
```

```
C                        This term is in the upper triangle:
                         A(I,J) = DCMPLX(A_Im,-A_Re)
                      END IF
                   END DO
                END DO
             ELSE IF(MOD .EQ. 4) THEN
C               Complex Symmetric:
                DO J=1,NC
                   JCOL = J + JCOL1 - 1
                   A_Im = DFLOAT(ISIGN*JCOL)
                   DO I=1,NR
                      IROW = I + IROW1 - 1
                      A_Re = DFLOAT(ISIGN*IROW)
                      IF(JCOL .LE. IROW) THEN
                         A(I,J) = DCMPLX(A_Re,A_Im)
                      ELSE
                         A(I,J) = DCMPLX(A_Im,A_Re)
                      END IF
                   END DO
                END DO
             ELSE
C               Complex Nonsymmetric:
                DO J=1,NC
                   JCOL = J + JCOL1 - 1
                   A_Im = DFLOAT(ISIGN*JCOL)
                   DO I=1,NR
                      IROW = I + IROW1 - 1
                      A_Re = DFLOAT(ISIGN*IROW)
                      A(I,J) = DCMPLX(A_Re,A_Im)
                   END DO
                END DO
             END IF
          END IF
          CALL FMSPOP (MYNAME)
          RETURN
          END
C=======================================================================
          SUBROUTINE RCHECK (A, NR, NC, IROW1, JCOL1)
C=======================================================================
C         Check the matrix block:
          REAL*8  A(NR,NC)
          INTEGER NR
          INTEGER NC
          INTEGER IROW1
          INTEGER JCOL1
          INTEGER I, J, IROW, JCOL
          REAL*8  SCALE, ATEST
          CHARACTER*6 MYNAME
          PARAMETER (MYNAME='RCHECK')
          COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
          REAL*8 ERROR
C
          CALL FMSPSH (MYNAME)
          SCALE = 10**ISCALE
          SCALE = ISIGN * SCALE
          DO I=1,NR
             IROW = I + IROW1 - 1
             AI = DFLOAT(IROW)
             DO J=1,NC
                JCOL = J + JCOL1 - 1
                AJ = DFLOAT(JCOL)
                IF(JCOL .LE. IROW) THEN
                   ATEST = SCALE*AI + AJ
                ELSE
```

```
                  IF(MOD .EQ. 1) THEN
                     ATEST = SCALE*AJ + AI
                  ELSE
                     ATEST = SCALE*AI + AJ
                  END IF
               END IF
            IF(A(I,J) .NE. ATEST) CALL RERROR (IROW, JCOL, A(I,J), ATEST)
            END DO
         END DO
         CALL FMSPOP (MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE CCHECK (A, NR, NC, IROW1, JCOL1)
C=======================================================================
C        Fill a matrix block with COMPLEX*16 data:
         COMPLEX*16 A(NR,NC)
         INTEGER NR
         INTEGER NC
         INTEGER IROW1
         INTEGER JCOL1
         INTEGER I, J, IROW, JCOL
         REAL*8  RN, SCALE, A_Re, A_Im
         COMPLEX*16 ATEST
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='CCHECK')
         COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
         REAL*8 ERROR
C
         CALL FMSPSH (MYNAME)
         SCALE = 10**ISCALE
         DO I=1,NR
            IROW = I + IROW1 - 1
            A_Re = DFLOAT(ISIGN*IROW)
            DO J=1,NC
               JCOL = J + JCOL1 - 1
               A_Im = DFLOAT(ISIGN*JCOL)
               IF(JCOL .LT. IROW) THEN
                  ATEST = DCMPLX(A_Re,A_Im)
               ELSE IF(JCOL .EQ. IROW) THEN
                  IF(MOD .EQ. 3) THEN
                     RN    = SCALE * A_Re + A_Im
                     ATEST = DCMPLX(RN,0.0D0)
                  ELSE
                     ATEST = DCMPLX(A_Re,A_Im)
                  END IF
               ELSE
                  IF(MOD .EQ. 3) THEN
                     ATEST = DCMPLX(A_Im,-A_Re)
                  ELSE IF(MOD .EQ. 4) THEN
                     ATEST = DCMPLX(A_Im,A_Re)
                  ELSE
                     ATEST = DCMPLX(A_Re,A_Im)
                  END IF
               END IF
            IF(A(I,J) .NE. ATEST) CALL CERROR (IROW,JCOL,A(I,J),ATEST)
            END DO
         END DO
         CALL FMSPOP (MYNAME)
         RETURN
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
```

```
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        INTEGER FUNCTION INTINC (I)
C-----------------------------------------------------------------------
C       This function increments a volatile shared variable.  It is
C       placed in a subroutine to prevent some compilers from storing
C       the value in a register and not updating it.
        INTEGER I
        I = I + 1
        INTINC = I
        RETURN
        END
C=======================================================================
        SUBROUTINE RERROR (IROW, JCOL, AIJ, AVALUE)
C-----------------------------------------------------------------------
C       This subroutine prints a real matrix term that is incorrect.
        INTEGER IROW, JCOL
        REAL*8  AIJ, AVALUE
        CHARACTER*55 FMT
        DATA FMT( 1:36)/'(''CPU___:A('',I_,'','',I_,'')='',F__.0,'';'/
        DATA FMT(37:55)/' Should be '',F__.0)'/
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
        INTEGER MYNODE
        INTEGER NERRORS
        DATA    NERRORS/0/
        CALL FMSIGT ('MYNODE', MYNODE)
        ISCALE2 = MAX0(2*ISCALE,10)
        WRITE(FMT( 6: 8), 2003) MYNODE
        WRITE(FMT(15:15), 2001) ISCALE
        WRITE(FMT(22:22), 2001) ISCALE
        WRITE(FMT(30:31), 2002) ISCALE2
        WRITE(FMT(51:52), 2002) ISCALE2
        IF(ERROR .EQ. 0.0D0) ERROR = AVALUE
        WRITE(6, FMT) IROW, JCOL, AIJ, AVALUE
        NERRORS = NERRORS + 1
        IF(NERRORS .GT. 100) STOP 'Fatal error checking Matrix Data'
 2001   FORMAT (I1)
 2002   FORMAT (I2)
 2003   FORMAT (I3.3)
        RETURN
        END
C=======================================================================
        SUBROUTINE CERROR (IROW, JCOL, AIJ, AVALUE)
C-----------------------------------------------------------------------
```

```
C       This subroutine prints a real matrix term that is incorrect.
        INTEGER IROW, JCOL
        COMPLEX*16  AIJ, AVALUE
        CHARACTER*82 FMT
        DATA FMT( 1:34)/''('CPU___:A('',I_,'','',I_,'')=('',F__.0'/
        DATA FMT(35:48)/',','','',F__.0,'');'/
        DATA FMT(49:82)/'_Should_be_('',F__.0,'','',F__.0,'')'')'/
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
        INTEGER MYNODE
        INTEGER NERRORS
        DATA    NERRORS/0/
        CALL FMSIGT ('MYNODE', MYNODE)
        ISCALE1 = ISCALE + 1
        WRITE(FMT( 6: 8), 2003) MYNODE
        WRITE(FMT(15:15), 2001) ISCALE
        WRITE(FMT(22:22), 2001) ISCALE
        WRITE(FMT(31:32), 2002) ISCALE1
        WRITE(FMT(41:42), 2002) ISCALE1
        WRITE(FMT(64:65), 2002) ISCALE1
        WRITE(FMT(74:75), 2002) ISCALE1
        IF(ERROR .EQ. 0.0D0) ERROR = AVALUE
        WRITE(6, FMT) IROW, JCOL, AIJ, AVALUE
        NERRORS = NERRORS + 1
        IF(NERRORS .GT. 100) STOP 'Fatal error checking Matrix Data'
 2001   FORMAT (I1)
 2002   FORMAT (I2)
 2003   FORMAT (I3.3)
        RETURN
        END
C       ================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), AIJ, ATEST, SCALE
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
C       Test the matrix data:
        SCALE = 10**ISCALE
        DO IROW = IROW1,IROW2
           JSTART = MAX0(LOWEQ(IROW), JCOL1)
           JEND   = MIN0(IROW,        JCOL2)
           LOCI   = LOCEQ(IROW)
           DO JCOL = JSTART, JEND
              IF(JCOL .EQ. IROW) THEN
C                This term is on the diagonal:
                 AIJ   = D(IROW)
              ELSE
C                This term is on or within the profile:
                 AIJ   = A(LOCI + IJSTEP*JCOL)
              END IF
              ATEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
              IF(AIJ .NE. ATEST) CALL RERROR (IROW, JCOL, AIJ, ATEST)
           END DO
        END DO
        RETURN
        END

C       ================================================================
        SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
```

```
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), AIJ, ATEST, SCALE
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
C       Test the matrix data:
        SCALE = 10**ISCALE
        DO JCOL = JCOL1,JCOL2
           ISTART = MAX0(LOWEQ(JCOL), IROW1)
           IEND   = MIN0(JCOL-1,      IROW2)
           LOCJ   = LOCEQ(JCOL)
           DO IROW = ISTART, IEND
              ATEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
              AIJ   = A(LOCJ + IJSTEP*IROW)
              IF(AIJ .NE. ATEST) CALL RERROR (IROW, JCOL, AIJ, ATEST)
           END DO
        END DO
        RETURN
        END

C       ================================================================
        SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     D(*), DII, DTEST, SCALE
        COMPLEX*16 A(0:*), AIJ, ATEST
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
C       Test the matrix data:
        SCALE = 10**ISCALE
        DO IROW = IROW1,IROW2
           JSTART = MAX0(LOWEQ(IROW), JCOL1)
           JEND   = MIN0(IROW,        JCOL2)
           LOCI   = LOCEQ(IROW)
           DO JCOL = JSTART, JEND
              IF(JCOL .EQ. IROW) THEN
C                This term is on the diagonal:
                 DII   = D(IROW)
                 DTEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
                 IF(DII .NE. DTEST) CALL RERROR (IROW, JCOL, DII, DTEST)
              ELSE
C                This term is on or within the profile:
                 AIJ   = A(LOCI + IJSTEP*JCOL)
                 IF(JCOL .LT. IROW) THEN
C                   This term is in the lower triangle:
                    ATEST = DCMPLX(IROW,JCOL)
                 ELSE
C                   This term is in the upper triangle:
c                   Use complex conjugate:
                    ATEST = DCMPLX(JCOL,-IROW)
                 END IF
                 IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
              END IF
           END DO
        END DO
        RETURN
        END

C       ================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
```

```
     1                    JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), AIJ, ATEST
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
C       Test the matrix data:
        DO IROW = IROW1,IROW2
           JSTART = MAX0(LOWEQ(IROW), JCOL1)
           JEND   = MIN0(IROW,        JCOL2)
           LOCI   = LOCEQ(IROW)
           DO JCOL = JSTART, JEND
              IF(JCOL .EQ. IROW) THEN
C                This term is on the diagonal:
                 AIJ   = D(IROW)
              ELSE
C                This term is on or within the profile:
                 AIJ   = A(LOCI + IJSTEP*JCOL)
              END IF
              ATEST = DCMPLX(IROW,JCOL)
              IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
           END DO
        END DO
        RETURN
        END

C       ================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                    JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), AIJ, ATEST
C
C       Test the matrix data:
        DO JCOL = JCOL1,JCOL2
           ISTART = MAX0(LOWEQ(JCOL), IROW1)
           IEND   = MIN0(JCOL-1,      IROW2)
           LOCJ   = LOCEQ(JCOL)
           DO IROW = ISTART, IEND
              AIJ   = A(LOCJ + IJSTEP*IROW)
              ATEST = DCMPLX(IROW,JCOL)
              IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
           END DO
        END DO
        RETURN
        END

C       ================================================================
        SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C       ================================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        REAL*8     A(0:*), AIJ, ATEST, SCALE
        COMMON /MYDATA/ERROR, MOD, N, NRHS, NROWS, NCOLS, ISCALE, ISIGN
        REAL*8 ERROR
C
C       Test the matrix data:
        SCALE = 10**ISCALE
        DO JCOL = JEQN1,JEQN2
           DO IROW = 1,NUMEQ
              AIJ   = A(LOCI(IROW) + LOCJ(1,LUFLAG(IROW)))
              ATEST = SCALE * DFLOAT(IROW) + DFLOAT(JCOL)
```

```
               IF(AIJ .NE. ATEST) CALL RERROR (IROW, JCOL, AIJ, ATEST)
            END DO
         END DO
         RETURN
         END

C        ================================================================
         SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         COMPLEX*16 A(0:*), AIJ, ATEST
C
C        Test the matrix data:
         DO JCOL = JEQN1,JEQN2
            DO IROW = 1,NUMEQ
               AIJ    = A(LOCI(IROW) + LOCJ(1,LUFLAG(IROW)))
               ATEST = DCMPLX(IROW,JCOL)
               IF(AIJ .NE. ATEST) CALL CERROR (IROW, JCOL, AIJ, ATEST)
            END DO
         END DO
         RETURN
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 14. CHANGING MATRIX ELEMENTS

This example shows how the FMS subroutines may be used to perform a sequence of solutions when some of the matrix terms are changing.

```
+-    +    -+ +-  -+   +-  -+
| A11 | A12 | | X1 |   | B1 |
+-----+-----+ +----+ = +----+
| A21 | A22 | | X2 |   | B2 |
+-    +    -+ +-  -+   +-  -+
```

The system of equations is partitioned as shown above. On the first solution, all terms are computed. On following solutions, only those terms which must be recomputed due to changes in the matrix elements are reevaluated. The main work is performed by subroutine RESOLV, which is designed to be cut and pasted into your application.
The following input is used to test subroutine RESOLV:

- Number of equations in [A11]
- Number of equations in [A22]
- Number of solution vectors
- Any FMS Parameter

After solving the system, you are asked the following:

- Do you want another solution?
- Do you want a new A11?
- Do you want a new A12?
- Do you want a new A21?
- Do you want a new A22?
- Do you want a new X1?
- Do you want a new X2?

If you answer yes to any of these questions, the problem is solved again, recomputing only those terms that are necessary.

```
C       T E S T     R E S O L V
C
C       This program tests subroutine RESOLV.
```

```
C       The test matrix is of the form
C
C              [A]        {X}  =  {B}
C       +-            -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-            -+ +   +   +   +
C
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_14')
C
C       FMS matrix and vector file attributes:
        INTEGER     LU_A11(25)
        INTEGER     LU_A12(25)
        INTEGER     LU_A21(25)
        INTEGER     LU_A22(25), LU_F22(25)
        INTEGER     LU_B1 (25),            LU_X1(25)
        INTEGER     LU_B2 (25), LU_S2(25), LU_X2(25)
C
C       FMS memory management requires the following arrays:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Data Type:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE=2)
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
```

```
C
C       LOCAL VARIABLES:
C       Number of equations in [A11]:
           INTEGER N1
C       Number of equations in [A22]:
           INTEGER N2
C       Maximum of N1 and N2:
           INTEGER NMAX
C       Number of solution vectors:
           INTEGER NRHS
C       Form a new [A11] matrix:
           LOGICAL NEWA11
C       Form a new [A12] matrix:
           LOGICAL NEWA12
C       Form a new [A21] matrix:
           LOGICAL NEWA21
C       Form a new [A22] matrix:
           LOGICAL NEWA22
C       Form a new {B1}  matrix:
           LOGICAL NEWB1
C       Form a new {B2}  matrix:
           LOGICAL NEWB2
C       Check the values of {X2}:
           LOGICAL ICK_X2
C       Error on current term:
           REAL*8  EI
C       Maximum overall error:
           REAL*8  E2MAX, ERROR
C       CMD Pointer to scratch vector:
           INTEGER L_TEMP
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
        DATA    LOWEQ/-1/
C
C       CONSTANTS:
           COMPLEX*16 C_ONE
           DATA C_ONE/(1.0D0,0.0D0)/
C
```

```
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
C
C       Use full column partial pivoting (SLAB format):
        CALL FMSIST ('MFMAT', 3)
C
C       Read in problem size parameters:
        N1   = ASK_I('Enter the number of equations in A11')
        N2   = ASK_I('Enter the number of equations in A22')
        NRHS = ASK_I('Enter the number of solution vectors')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Get a temporary vector for generating columns and RHS's:
        NMAX = MAX0(N1,N2)
        CALL FMSCMG (CMD, L_TEMP, NMAX)
C
C (2)   Open FMS files:
        CALL CNDI  (LOWEQ, N1,           'LU_A11', LU_A11)
        CALL FMSOV (N1, IDTYPE, N2  , 'LU_A12', LU_A12)
        CALL FMSOV (N1, IDTYPE, N2  , 'LU_A21', LU_A21)
        CALL CNDI  (LOWEQ, N2,           'LU_A22', LU_A22)
        CALL FMSOM  (LU_A22, 'LU_F22', LU_F22)
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_B1' , LU_B1 )
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_X1' , LU_X1 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_B2' , LU_B2 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_S2' , LU_S2 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_X2' , LU_X2 )
C
C (3)   Write data to FMS files:
        NEWA11 = .TRUE.
        NEWA12 = .TRUE.
        NEWA21 = .TRUE.
        NEWA22 = .TRUE.
```

```
        NEWB1  = .TRUE.
        NEWB2  = .TRUE.
        ICK_X2 = .TRUE.
C
C       Branch back to here for refinement solutions:
   30   CONTINUE
C
C       Generate [A11]:
        IF(NEWA11) CALL A11GEN (CMD(L_TEMP), N1, N2,   LU_A11)
C
C       Generate [A12]:
        IF(NEWA12) CALL A12GEN (CMD(L_TEMP), N1, N2,   LU_A12)
C
C       Generate [A21]:
        IF(NEWA21) CALL A21GEN (CMD(L_TEMP), N1, N2,   LU_A21)
C
C       Generate [A22]:
        IF(NEWA22) CALL A22GEN (CMD(L_TEMP), N2,       LU_A22)
C
C       Generate {B1}:
        IF(NEWB1 ) CALL B1GEN  (CMD(L_TEMP), N1, NRHS, LU_B1 )
C
C       Generate {B2}:
        IF(NEWB2 ) CALL B2GEN  (CMD(L_TEMP), N2, NRHS, LU_B2 )
C
C (4)   Perform matrix algebra:
        CALL RESOLV
     1  (LU_A11, LU_A11,
     2   LU_A12, LU_A12,
     3   LU_A21,
     4   LU_A22, LU_F22,
     5   LU_B1,  LU_B1, LU_X1,
     6   LU_B2,  LU_S2, LU_X2,
     7   NEWA11, NEWA12,
     8   NEWA21, NEWA22,
     9   NEWB1,  NEWB2,
     1   ICK_X2, E2MAX)
C
```

```
C (5)   Read data from FMS files:
C       Check the answer:
        ERROR  = E2MAX
C
C       Initialize FMSGET:
        CALL FMSGET (LU_X1, 0, 0, 0, 0, CMD, 0)
        DO 60 IRHS = 1,NRHS
C
C          Check {X1}:
           CALL FMSGET (LU_X1, N1, 1, 1, IRHS, CMD(L_TEMP), N1)
           DO 51 I = 0,(N1-1)
              EI = ABS(CMD(L_TEMP+I) - C_ONE)
              IF(EI .GT. ERROR) ERROR = EI
   51      CONTINUE
   60   CONTINUE
C
C       End FMSGET:
        CALL FMSGET (LU_X1, 0, 0, N1+1, 1, CMD, 0)
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C       Modify the matrix and solve it again:
        IF(.NOT. ASK('Do you want another solution?')) GO TO 100
        NEWA11 = ASK('Do you want a new A11?')
        NEWA12 = ASK('Do you want a new A12?')
        NEWA21 = ASK('Do you want a new A21?')
        NEWA22 = ASK('Do you want a new A22?')
        NEWB1  = ASK('Do you want a new B1?' )
        NEWB2  = ASK('Do you want a new B2?' )
C
C       We have chosen to overlay factored data on input matrices.
C       Therefore if new factors must be recomputed, the arrays must
C       be reset to the unfactored values.
C       Based on the options selected above, these conditions become:
        IF( NEWA11 ) THEN
            NEWA12 = .TRUE.
            NEWA22 = .TRUE.
            NEWB1  = .TRUE.
            NEWB2  = .TRUE.
        END IF
```

```
         IF( NEWA12 ) NEWA22 = .TRUE.
         IF( NEWB1  ) NEWB2  = .TRUE.
         IF( NEWA11 .OR.
     1       NEWA12 .OR.
     2       NEWA21 .OR.
     3       NEWA22 .OR.
     4       NEWB1  .OR.
     5       NEWB2) GO TO 30
C
C (6)    Close FMS files:
  100    CONTINUE
         CALL FMSCM (LU_A11)
         CALL FMSCV (LU_A12)
         CALL FMSCV (LU_A21)
         CALL FMSCV (LU_A22)
         CALL FMSCM (LU_F22)
         CALL FMSCV (LU_B1 )
         CALL FMSCV (LU_X1 )
         CALL FMSCV (LU_B2 )
         CALL FMSCV (LU_S2 )
         CALL FMSCV (LU_X2 )
         CALL FMSCMR (CMD, L_TEMP, NMAX)
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
         END IF
         RETURN
 1000    FORMAT (A)
```

```
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE A11GEN (ACOL, N1, N2, LU_A11)
C=======================================================================
        COMPLEX*16 ACOL(N1)
        INTEGER    N1, N2
        INTEGER    LU_A11(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A11GEN')
        REAL*8     R_ZERO, A11_Re
        DATA       R_ZERO/0.0D0/
        COMPLEX*16 C_MONE, C_ZERO, C_ONE
        DATA       C_MONE/(-1.0D0,0.0D0)/
        DATA       C_ZERO/( 0.0D0,0.0D0)/
        DATA       C_ONE /( 1.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
C       Initialize FMSCOL
        CALL FMSCOL (0, ACOL, LU_A11)
C
C       Write first column
        DO 10 I = 2,N1
           ACOL(I) = C_MONE
   10   CONTINUE
        A11_Re  = DFLOAT(N1+N2)
        ACOL(1) = DCMPLX(A11_Re,R_ZERO)
        CALL FMSCOL (1, ACOL, LU_A11)
```

```
C
C       Write remaining columns:
        DO 20 I = 2,N1
           ACOL(I) = C_ZERO
   20   CONTINUE
        ACOL(1) = C_MONE
        DO 30 I = 2,N1
           ACOL(I) = C_ONE
           CALL FMSCOL (I, ACOL, LU_A11)
           ACOL(I) = C_ZERO
   30   CONTINUE
C
C       End FMSCOL
        CALL FMSCOL (N1+1, ACOL, LU_A11)
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating [A11]:')
        END
C=======================================================================
        SUBROUTINE A12GEN (ACOL, N1, N2, LU_A12)
C=======================================================================
        COMPLEX*16 ACOL(N1)
        INTEGER    N1, N2
        INTEGER    LU_A12(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A12GEN')
        COMPLEX*16 C_MONE, C_ZERO
        DATA       C_MONE/(-1.0D0,0.0D0)/
        DATA       C_ZERO/( 0.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
           ACOL(I) = C_ZERO
   10   CONTINUE
        ACOL(1) = C_MONE
C
C       Initialize FMSPUT:
```

```
        CALL FMSPUT (LU_A12, 0, 0, 0, 0, ACOL, 0)
C
C       Write out columns of [A12]:
        DO 20 I = 1,N2
           CALL FMSPUT (LU_A12, N1, 1, 1, I, ACOL, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_A12, 0, 0, N1+1, 0, ACOL, 0)
C
C       Set status:
        LU_A12(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating [A12]:')
        END
C=======================================================================
        SUBROUTINE A21GEN (AROW, N1, N2, LU_A21)
C=======================================================================
        COMPLEX*16 AROW(N1)
        INTEGER    N1, N2
        INTEGER    LU_A21(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A21GEN')
        COMPLEX*16 C_MONE, C_ZERO
        DATA       C_MONE/(-1.0D0,0.0D0)/
        DATA       C_ZERO/( 0.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
           AROW(I) = C_ZERO
   10   CONTINUE
        AROW(1) = C_MONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_A21, 0, 0, 0, 0, AROW, 0)
C
```

```
C       Write out rows of [A21], which is stored transposed:
        DO 20 I = 1,N2
           CALL FMSPUT (LU_A21, N1, 1, 1, I, AROW, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_A21, 0, 0, N1+1, 0, AROW, 0)
C
C       Set status:
        LU_A21(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating [A21]:')
        END
C=======================================================================
        SUBROUTINE A22GEN (ACOL, N2, LU_A22)
C=======================================================================
        COMPLEX*16 ACOL(N2)
        INTEGER    N2
        INTEGER    LU_A22(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A22GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA       C_ZERO/( 0.0D0,0.0D0)/
        DATA       C_ONE /( 1.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
C       Initialize FMSCOL
        CALL FMSCOL (0, ACOL, LU_A22)
C
        DO 10 I = 1,N2
           ACOL(I) = C_ZERO
   10   CONTINUE
C
        DO 20 I = 1,N2
           ACOL(I) = C_ONE
```

```
            CALL FMSCOL (I, ACOL, LU_A22)
            ACOL(I) = C_ZERO
   20    CONTINUE
C
C       End FMSCOL
        CALL FMSCOL (N2+1, ACOL, LU_A22)
C
C       Set status:
        LU_A22(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating [A22]:')
        END
C=======================================================================
        SUBROUTINE B1GEN (RHS, N1, NRHS, LU_B1)
C=======================================================================
        COMPLEX*16 RHS(N1)
        INTEGER    N1, NRHS
        INTEGER    LU_B1(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B1GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA       C_ZERO/(0.0D0,0.0D0)/
        DATA       C_ONE /(1.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
          RHS(I) = C_ZERO
   10   CONTINUE
        RHS(1) = C_ONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B1}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B1, N1, 1, 1, I, RHS, N1)
```

```
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, N1+1, 0, RHS, 0)
C
C       Set status:
        LU_B1(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating {B1}:')
        END
C=======================================================================
        SUBROUTINE B2GEN (RHS, N2, NRHS, LU_B2)
C=======================================================================
        COMPLEX*16 RHS(N2)
        INTEGER    N2, NRHS
        INTEGER    LU_B2(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B2GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA       C_ZERO/(0.0D0,0.0D0)/
        DATA       C_ONE /(1.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
        DO 10 I=1,N2
          RHS(I) = C_ZERO
   10   CONTINUE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B2, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B2}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B2, N2, 1, 1, I, RHS, N2)
   20   CONTINUE
C
C       End FMSPUT:
```

```
        CALL FMSPUT (LU_B2, 0, 0, N2+1, 0, RHS, 0)
C
C       Set status:
        LU_B2(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating {B2}:')
        END
C=======================================================================
        SUBROUTINE RESOLV
     1  (LU_A11, LU_F11,
     2   LU_A12, LU_F12,
     2   LU_A21,
     4   LU_A22, LU_F22,
     3   LU_B1,  LU_S1, LU_X1,
     4   LU_B2,  LU_S2, LU_X2,
     5   NEWA11, NEWA12,
     6   NEWA21, NEWA22,
     7   NEWB1,  NEWB2,
     6   ICK_X2, E2MAX)
C=======================================================================
C
C       DESCRIPTION:
C          This subroutine solves the linear system
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | | X1 |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | | X2 |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C          when all or only part of the matrix terms change.
C
C
C          On input, the following matrix terms are specified:
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | |    |   | B1 |
C          +-----+-----+ +----+ = +----+
```

```
C          | A21 | A22 | |    |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          The following intermediate work arrays are used:
C
C          +-    +    -+ +-  -+   +-  -+
C          |     |     | | S1 |   |    |
C          +-----+-----+ +----+ = +----+
C          |     |     | | S2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          On output, the following terms are returned:
C
C          +-    +    -+ +-  -+   +-  -+
C          | F11 | F12 | | X1 |   |    |
C          +-----+-----+ +----+ = +----+
C          |     | F22 | | X2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
C
C       You may overlay any of the corresponding input, intermediate
C       or output terms to save storage by specifying the same actual
C       parameter on the subroutine call.
C
C       Arrays containing input data may share storage with
C       intermediate and output arrays if these conditions are met:
C       1) If any value changes in the input array, all values in that
C          input array are redefined.
C       2) In defining the new values for an array, the previous
C          values are not required. (They are no longer available
C          because they have been overwritten)
C       If these conditions are true, then the following arrays can
C       share the same storage:
C          [A11] & [F11],
C          [A12] & [F12],
C          [A22] & [F22],
C          {B1}  &  {S1},
C          {B2}  &  {S2}.
```

```
C
C       Ths storage for {S1} and {S2} can be shared with {X1} and {X2}.
C       No additional computation is required if the following
C       conditions are met:
C          {S1} & {X1} : if [A11]      or {B1}         change,
C          {S2} & {X2} : if [A11], [A12], {B1} or {B2} change.
C       If these conditions are not met and the storage is shared, then
C       additional computation will be performed.
C
C       You may also overlay {B1},{S1},{X1} or {B2},{S2},{X2} if
C       all the above conditions are met.
C
C       FORMAL PARAMETERS:
C          (R ) LU_A11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A11]
C
C          (R ) LU_F11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F11]
C                             May be the same as LU_A11.
C
C          (R ) LU_A12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A12]
C
C          (R ) LU_F12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [F12]
C                             May be the same as LU_A12.
C
C          (R ) LU_A21(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A21]
C                             Data on this file is not changed.
C
C          (R ) LU_A22(25)  = INTEGER ARRAY
C                             FMS mtarix file attributes for [A22]
C
C          (R ) LU_F22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F22]
C
C          (R ) LU_B1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B1}
```

```
C
C          (R ) LU_S1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {S1}.
C                             This is an intermediate work array.
C                             May be the same value as LU_B1.
C
C          (R ) LU_X1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X1}.
C                             May be the same as LU_S1.
C
C          (R ) LU_B2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B2}
C
C          (R ) LU_S2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {S2}.
C                             This is an intermediate work array.
C                             May be the same value as LU_B2.
C
C          (R ) LU_X2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X2}.
C                             May be the same as LU_B2.
C
C          (R ) NEWA11      = LOGICAL
C                             Indicates that [A11] has changed.
C
C          (R ) NEWA12      = LOGICAL
C                             Indicated that [A12] has changed.
C
C          (R ) NEWA21      = LOGICAL
C                             Indicated that [A21] has changed.
C
C          (R ) NEWA22      = LOGICAL
C                             Indicated that [A22] has changed.
C
C          (R ) NEWB1       = Logical
C                             Indicates that {B1} has changed.
C
C          (R ) NEWB2       = Logical
```

```
C                             Indicates that {B2} has changed.
C
C          (R ) ICK_X2      = Logical
C                             Check the values in {X2}.
C                             This requires access to [A22] and {B2}.
C                             Requires the following:
C                             LU_S2  or LU_X2 do not overlay LU_B2
C
C          (R ) E2MAX       = REAL*8
C                             Maximum error in {X2}.
C
C       FMSCOM PARAMETERS:
C          (RW) IACCOM      = Multiply accumulate flag
C          (R ) LUPR        = FORTRAN unit for printing
C
C       CALLED FROM:
C          EXAMPLE_14
C
C       SUBPROGRAMS USED:
C          CNDF
C          CNDS
C          FMSMM
C
C       ERROR CONDITIONS:
C          None.
C
C       HISTORY:
C
C       NOTES:
C
C       Expanding the partitioned system gives:
C
C       [A11]{X1} + [A12]{X2} = {B1}      (1)
C       [A21]{X1} + [A22]{X2} = {B2}      (2)
C
C       Premultiplying (1) by inv[A11] gives
C
C       {X1} = {S1} - [F12]{X2}           (3)
C
```

```
C       where
C          {S1}  = inv[A11]{B1}           (4)
C          [F12] = inv[A11][A12]          (5)
C
C       Substituting (3) into (2) gives:
C
C       [F22]{X2} = {S2}                  (6)
C
C       where
C          [F22] = [A22] - [A21][F12]     (7)
C          {S2}  = {B2}  - [A21]{S1}      (8)
C
C
C       The error in {X2} may be computed from:
C
C          E2MAX = | {B2} - [A21]{X1} - [A22]{X2} |
C
C       The algorithm consists of the following 8 steps:
C
C       Solve large system:
C       1. Factor   [A11] into [F11]=[L11][U11]
C       2. Solve    [F11][F12] = [A12]            (eq. 5)
C       3. Solve    [F11]{S1}  = {B1}             (eq. 4)
C
C       Project to small system:
C       4. Multiply [F22'] = [A22] - [A21][F12]   (eq. 7)
C       5. Factor   [F22'] into [F22]=[L22][U22]
C       6. Multiply {S2}   = {B2}  - [A21]{S1}    (eq. 8)
C
C       Solve small system:
C       7. Solve    [F22]{X2} = {S2}              (eq. 6)
C
C       Project back to large system:
C       8. Multiply {X1} = {S1} - [F12]*{X2}      (eq. 3)
C
C       Check {X2}: (optional)
C       9. Multiply {ERR} = {B2} - [A21]{X1} - [A22]{X2}
C          Then compute the maximum absolute value in {ERR}.
C
```

```
C       Summary of arrays used:
C
C       STEP  A11 F11 A12 F12 A21 A22 F22  B1  S1  X1  B2  S2  X2
C       ====  ======= ======= === ======= =========== ===========
C         1    R-->W
C         2        R   R-->W
C         3        R                       R-->W
C         4               R   R   R-->W
C         5                           RW
C         6                   R               R       R-->W
C         7                           R                   R-->W
C         8               R                   R-->W           R
C
C       Steps required:
C
C       STEP1 = (NEWA11                 )
C       STEP2 = (NEWA12 | STEP1         )
C       STEP3 = (NEWB1  | STEP1         )
C       STEP4 = (NEWA22 | NEWA21 | STEP2 )
C       STEP5 = (STEP4                  )
C       STEP6 = (NEWB2  | NEWA21 | STEP3 )
C       STEP7 = (STEP5  | STEP6         )
C       STEP8 = (STEP3  | STEP2  | STEP7 )
C
C       or, substituting,
C
C       STEP1 = (NEWA11                                         )
C       STEP2 = (NEWA11 | NEWA12                                )
C       STEP3 = (NEWA11 |                           NEWB1       )
C       STEP4 = (NEWA11 | NEWA12 | NEWA21 | NEWA22              )
C       STEP5 = (NEWA11 | NEWA12 | NEWA21 | NEWA22              )
C       STEP6 = (NEWA11 |          NEWA21 |          NEWB1 | NEWB2 )
C       STEP7 = (NEWA11 | NEWA12 | NEWA21 | NEWA22 | NEWB1 | NEWB2 )
C       STEP8 = (NEWA11 | NEWA12 | NEWA21 | NEWA22 | NEWB1 | NEWB2 )
C
C       Also if the intermediate vector storage is overlaid on the
C       storage for the solution vectors, it is assumed that it no
C       longer contains the intermediate arrays.  Therefore,
C       STEP3 = (STEP3  | (LU_S1 = LU_X1) )
```

```
C       STEP6 = (STEP6  | (LU_S2 = LU_X2) )
C-----------------------------------------------------------------------
C       FORMAL PARAMETERS
C-----------------------------------------------------------------------
        INTEGER    LU_A11(25)
        INTEGER    LU_F11(25)
C
        INTEGER    LU_A12(25)
        INTEGER    LU_F12(25)
C
        INTEGER    LU_A21(25)
C
        INTEGER    LU_A22(25)
C
        INTEGER    LU_F22(25)
C
        INTEGER    LU_B1 (25)
        INTEGER    LU_S1 (25)
        INTEGER    LU_X1 (25)
C
        INTEGER    LU_B2 (25)
        INTEGER    LU_S2 (25)
        INTEGER    LU_X2 (25)
C
        LOGICAL    NEWA11
        LOGICAL    NEWA12
        LOGICAL    NEWA21
        LOGICAL    NEWA22
        LOGICAL    NEWB1
        LOGICAL    NEWB2
        LOGICAL    ICK_X2
        REAL*8     E2MAX
C-----------------------------------------------------------------------
C       LOCAL VARIABLES
C-----------------------------------------------------------------------
C
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='RESOLV')
C
```

```
C       FMS Memory management:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
        COMMON /MYFMS/ L_A22, LDA22
C
C       Data type = complex:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE = 2)
C
C       Skip operations during solving (no):
        INTEGER    ISKIP
        PARAMETER (ISKIP=0)
C
C       Constants:
        COMPLEX*16 C_ONE(1)
        DATA       C_ONE  /(1.0D0,0.0D0)/
        REAL*8     R_ZERO
        DATA       R_ZERO / 0.0D0/
        REAL*8     R_ONE
        DATA       R_ONE  / 1.0D0/
        REAL*8     R_THREE
        DATA       R_THREE/ 3.0D0/
        REAL*8     R_EIGHT
        DATA       R_EIGHT/ 8.0D0/
C
C       FMS Parameters:
        INTEGER    LUPR
C
C       Matrix dimensions:
        INTEGER    N1
        INTEGER    N2
        INTEGER    NRHS
        REAL*8     RN1
        REAL*8     RN2
```

```
        REAL*8     RNRHS
C
C       Temporary file:
        INTEGER    LU_ERR(25)
        INTEGER     LUA0  (25)
        DATA        LUA0(1)/0/
        INTEGER     LUX0  (25)
        INTEGER     LUS0  (25)
C
C       CPU, Wall and I/O time:
        REAL*8     TCPU(10)
        REAL*8     TWALL(10)
        REAL*8     TIO(10)
        REAL*8     T1, T2, T3
C
C       Steps to perform:
        LOGICAL    STEP1
        LOGICAL    STEP2
        LOGICAL    STEP3
        LOGICAL    STEP4
        LOGICAL    STEP5
        LOGICAL    STEP6
        LOGICAL    STEP7
        LOGICAL    STEP8
        LOGICAL    STEP9
C
C       Floating point operations:
        REAL*8     OPS(9), TOPS
C-----------------------------------------------------------------------
C       FORTRAN PROCEDURE
C-----------------------------------------------------------------------
        CALL FMSPSH(MYNAME)
C
C       Get the FORTRAN unit for printng:
        CALL FMSIGT ('LUPR', LUPR)
C
C       Get the memory pointer for arrays IMD, RMD and CMD:
        CALL FMSIGT ('MEMPTR', IMD_PTR)
```

```
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C     Determine the size of the matrices:
      N1    = LU_A11(8)
      N2    = LU_A22(8)
      NRHS  = LU_B1 (6)
      NUMEQ = N1 + N2
      RN1   = DFLOAT(N1)
      RN2   = DFLOAT(N2)
      RNRHS = DFLOAT(NRHS)
      WRITE(LUPR,2000) NUMEQ, N1, N2, NRHS
C
C     Determine which steps to do:
      STEP1 = (NEWA11                        )
      STEP2 = (NEWA12 .OR. STEP1             )
      STEP3 = (NEWB1  .OR. STEP1             )
      STEP4 = (NEWA22 .OR. NEWA21 .OR. STEP2 )
      STEP5 = (STEP4                         )
      STEP6 = (NEWB2  .OR. NEWA21 .OR. STEP3 )
      STEP7 = (STEP5  .OR. STEP6             )
      STEP8 = (STEP2  .OR. STEP3  .OR. STEP7 )
      STEP9 = (ICK_X2                        )
C
C     Check overlaid storage:
      STEP3 = (STEP3  .OR. (LU_S1(1) .EQ. LU_X1(1)) )
      STEP6 = (STEP6  .OR. (LU_S2(2) .EQ. LU_X2(2)) )
      IF( (LU_B2(1)  .EQ. LU_S2(1) )   .OR.
     1     (LU_B2(1)  .EQ. LU_X2(1) ) ) STEP9 = .FALSE.
C
C     Initialize operation counts for each step:
      OPS(1) = R_ZERO
      OPS(2) = R_ZERO
      OPS(3) = R_ZERO
      OPS(4) = R_ZERO
      OPS(5) = R_ZERO
      OPS(6) = R_ZERO
      OPS(7) = R_ZERO
      OPS(8) = R_ZERO
```

```
         OPS(9) = R_ZERO
C
C (1)    Factor [A11] into [L11][U11]:
         CALL FMSTIM (TCPU(1), TWALL(1), TIO(1))
         IF(STEP1) THEN
            WRITE(LUPR,2001)
            OPS(1) = R_EIGHT*RN1*RN1*RN1/R_THREE
            CALL CNDAF (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11,
     1                  LUX0, LUX0, 0)
         END IF
C
C (2)    Solve [F11][F12] = [A12]:
         CALL FMSTIM (TCPU(2), TWALL(2), TIO(2))
         IF(STEP2) THEN
            WRITE(LUPR,2002)
            OPS(2) = R_EIGHT*RN1*RN1*RN2
            CALL CNDS (LU_F11, LU_A12, LU_F12, N2,   0)
         END IF
C
C (3)    Solve [F11]{S1}  = {B1}:
         CALL FMSTIM (TCPU(3), TWALL(3), TIO(3))
         IF(STEP3)  THEN
            WRITE(LUPR,2003)
            OPS(3) = R_EIGHT*RN1*RN1*RNRHS
            CALL CNDS (LU_F11, LU_B1,  LU_S1,  NRHS, 0)
         END IF
C
C (4)    Multiply [F22'] = [A22] - [A21][F12]
         CALL FMSTIM (TCPU(4), TWALL(4), TIO(4))
         IF(STEP4)  THEN
            WRITE(LUPR,2004)
            OPS(4) = R_EIGHT*RN1*RN2*RN2
            CALL FMSMM (LU_F22, LU_A22, -1, 'T', LU_A21, LU_F12)
         END IF
C
C (5)    Factor [F22'] into [F22]:
         CALL FMSTIM (TCPU(5), TWALL(5), TIO(5))
         IF(STEP5) THEN
            WRITE(LUPR,2005)
```

```
            OPS(5) = R_EIGHT*RN2*RN2*RN2/R_THREE
            CALL CNDAF (LU_F22, C_ONE, 1, LUS0, 0, LUA0, LU_F22,
     1                  LUX0, LUX0, 0)
         END IF
C
C (6)    Multiply {S2}  = {B2}  - [A21]{S1}
         CALL FMSTIM (TCPU(6), TWALL(6), TIO(6))
         IF(STEP6) THEN
            WRITE(LUPR,2006)
            OPS(6) = R_EIGHT*RN2*RN1*RNRHS
            CALL FMSMM (LU_S2, LU_B2, -1, 'T', LU_A21, LU_S1)
         END IF
C
C (7)    Solve [F22]{X2} = {S2}:
         CALL FMSTIM (TCPU(7), TWALL(7), TIO(7))
         IF(STEP7) THEN
            WRITE(LUPR,2007)
            OPS(7) = R_EIGHT*RN2*RN2*RNRHS
            CALL CNDS  (LU_F22, LU_S2, LU_X2, NRHS, 0)
         END IF
C
C (8)    Multiply {X1} = {S1} - [F12]{X2}
         CALL FMSTIM (TCPU(8), TWALL(8), TIO(8))
         IF(STEP8) THEN
            WRITE(LUPR,2008)
            OPS(8) = R_EIGHT*RN1*RN2*RNRHS
            CALL FMSMM ('N', -1, LU_F12, LU_X2, LU_S1, LU_X1)
         END IF
C
C (9)    Check {X2}: {ERR} = {B2} - [A21]{X1} - [A22]{X2}
         CALL FMSTIM (TCPU(9), TWALL(9), TIO(9))
         IF(STEP9) THEN
            WRITE(LUPR,2009)
            OPS(9) = R_EIGHT*(RN1+RN2)*RN2*RNRHS
C
C           Open a vector file to store the error array ERR(N2,NRHS):
         CALL FMSOV (N2, IDTYPE, NRHS, 'LU_ERR' , LU_ERR)
C
C           Multiply {ERR} = {B2} - [A21]{X1}
```

```
            CALL FMSMM (LU_ERR, LU_B2, -1, 'T', LU_A21, LU_X1)
C
C           Multiply {ERR} = {ERR} - [A22]{X2}:
            CALL FMSMM (LU_ERR, LU_ERR, -1, 'N', LU_A22, LU_X2)
C
C           Read LU_ERR and check the error:
C
C           Allocate storage for one vector:
            CALL FMSCMG (CMD, L_ERR, N2)
C
C           Initialize FMSGET:
            CALL FMSGET (LU_ERR, 0, 0, 0, 0, CMD, 0)
C
C           Compute the maximum error:
            E2MAX = R_ZERO
            DO 95 N = 1,NRHS
C              Read in next vector:
               CALL FMSGET (LU_ERR, N2, 1, 1, N, CMD(L_ERR), N2)
               L = L_ERR
               DO 90 I = 1,N2
                  T1 = ABS(CMD(L))
                  IF(T1 .GT. E2MAX) E2MAX = T1
                  L = L + 1
   90          CONTINUE
   95       CONTINUE
C
C           End FMSGET:
            CALL FMSGET (LU_ERR, 0, 0, N2+1, 1, CMD, 0)
C
C           Close the vector file LU_ERR:
            CALL FMSCV  (LU_ERR)
C
C           Delete the storage for {ERR}:
            CALL FMSCMR (CMD, L_ERR, N2)
         END IF
C
C        Print timings:
         WRITE(LUPR,2010)
         CALL FMSTIM (TCPU(10), TWALL(10), TIO(10))
```

```
         TOPS = R_ZERO
         DO 100 ISTEP = 1,9
            IF(OPS(ISTEP) .EQ. R_ZERO) THEN
C
C              Operation skipped:
               WRITE(LUPR,2012) ISTEP
            ELSE
               TOPS = TOPS + OPS(ISTEP)
C
C              CPU time and Mflops:
               T1 = TCPU(ISTEP+1)  - TCPU(ISTEP)
               IF(T1 .GT. R_ZERO) THEN
                  MF1   = ((1.0D-6)*OPS(ISTEP)/T1)
               ELSE
                  MF1   = 0
               END IF
C
C              Wall time, Mflops and ratio:
               T2 = TWALL(ISTEP+1) - TWALL(ISTEP)
               T3 = TIO(ISTEP+1)   - TIO(ISTEP)
               IF((T2 .GT. R_ZERO) .AND.
     1            (T2 .GT.  T3   ) ) THEN
                  MF2   = ((1.0D-6)*OPS(ISTEP)/T2)
                  RATIO = T1/(T2-T3)
               ELSE
                  MF2   = 0
                  RATIO = R_ONE
               END IF
               WRITE(LUPR,2011) ISTEP, T1, T2, T3, MF1, MF2, RATIO
            END IF
  100    CONTINUE
         T1 = TCPU(10)  - TCPU(1)
         T2 = TWALL(10) - TWALL(1)
         T3 = TIO(10)   - TIO(1)
         IF(T1 .GT. R_ZERO) THEN
            MF1   = ((1.0D-6)*TOPS/T1)
         ELSE
            MF1   = 0
         END IF
```

```
        IF(T2 .GT. R_ZERO) THEN
           MF2   = ((1.0D-6)*TOPS/T2)
           RATIO = T1/(T2-T3)
        ELSE
           MF2   = 0
           RATIO = R_ONE
        END IF
        WRITE(LUPR,2013) T1, T2, T3, MF1, MF2, RATIO
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (/
     1  ' Start of Subroutine RESOLV',/
     2  ' Rank of full matrix.....=',I6/
     3  ' Rank of submatrix #1....=',I6/
     4  ' Rank of submatrix #2....=',I6/
     5  ' # of right-hand-sides...=',I6/)
 2001   FORMAT (/
     1  ' STEP 1. Factor   [A11] into [F11]'/
     2  ' ================================')
 2002   FORMAT (/
     1  ' STEP 2. Solve    [F11][F12] = [A12]'/
     2  ' ===================================')
 2003   FORMAT (/
     1  ' STEP 3. Solve    [F11]{S1}  = {B1}'/
     2  ' ================================')
 2004   FORMAT (/
     1  ' STEP 4. Multiply [F22] = [A22] - [A21][F12]'/
     2  ' ============================================')
 2005   FORMAT (/
     1  ' STEP 5. Factor   [F22]'/
     2  ' ======================')
 2006   FORMAT (/
     1  ' STEP 6. Multiply {S2}  = {B2}  - [A21]{S1}'/
     2  ' ===========================================')
 2007   FORMAT (/
     1  ' STEP 7. Solve    [F22]{X2} = {S2}'/
     2  ' ================================')
 2008   FORMAT (/
```

```
     1  ' STEP 8. Multiply {X1} = {S1} - [A12]*{X2}'/
     2  ' ========================================')
  2009   FORMAT (/
     1  ' STEP 9. Checking values in {X2}'/
     2  ' ========================================')
  2010   FORMAT (/
     1  ' Summary of Subroutine RESOLV:'//
     2  '          <----------TIMES (Sec.)---------->'
     2  '    <--MFLOPS-->'/
     2  '  STEP          CPU         WALL          I/O     CPU    WALL',
     2  '    RATIO'/
     3  ' =====    =========    =========    =========    ====    ====',
     3  '    =====')
  2011   FORMAT (I6,3F13.3,2I8,F9.3)
  2012   FORMAT (I6,6X,'Skipped')
  2013   FORMAT (' TOTAL',3F13.3,2I8,F9.3)
         END
```

Copyright © Multipath Corporation

```
C       T E S T     R E S O L V
C
C       This program tests subroutine RESOLV.
C       The test matrix is of the form
C
C              [A]         {X}  =  {B}
C       +-             -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-             -+ +   +   +   +
C
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_14')
C
C       FMS matrix and vector file attributes:
        INTEGER     LU_A11(25)
        INTEGER     LU_A12(25)
        INTEGER     LU_A21(25)
        INTEGER     LU_A22(25), LU_F22(25)
        INTEGER     LU_B1 (25),            LU_X1(25)
        INTEGER     LU_B2 (25), LU_S2(25), LU_X2(25)
C
C       FMS memory management requires the following arrays:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Data Type:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE=2)
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       LOCAL VARIABLES:
C       Number of equations in [A11]:
           INTEGER N1
C       Number of equations in [A22]:
           INTEGER N2
C       Maximum of N1 and N2:
           INTEGER NMAX
C       Number of solution vectors:
           INTEGER NRHS
C       Form a new [A11] matrix:
           LOGICAL NEWA11
C       Form a new [A12] matrix:
           LOGICAL NEWA12
C       Form a new [A21] matrix:
           LOGICAL NEWA21
C       Form a new [A22] matrix:
           LOGICAL NEWA22
C       Form a new {B1}  matrix:
           LOGICAL NEWB1
C       Form a new {B2}  matrix:
```

```
            LOGICAL NEWB2
C       Check the values of {X2}:
            LOGICAL ICK_X2
C       Error on current term:
            REAL*8  EI
C       Maximum overall error:
            REAL*8  E2MAX, ERROR
C       CMD Pointer to scratch vector:
            INTEGER L_TEMP
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
        DATA    LOWEQ/-1/
C
C       CONSTANTS:
            COMPLEX*16 C_ONE
            DATA C_ONE/(1.0D0,0.0D0)/
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
C
C       Use full column partial pivoting (SLAB format):
        CALL FMSIST ('MFMAT', 3)
C
C       Read in problem size parameters:
        N1   = ASK_I('Enter the number of equations in A11')
        N2   = ASK_I('Enter the number of equations in A22')
        NRHS = ASK_I('Enter the number of solution vectors')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Get a temporary vector for generating columns and RHS's:
        NMAX = MAX0(N1,N2)
        CALL FMSCMG (CMD, L_TEMP, NMAX)
C
C (2)   Open FMS files:
        CALL CNDI  (LOWEQ, N1,           'LU_A11', LU_A11)
        CALL FMSOV (N1, IDTYPE, N2  , 'LU_A12', LU_A12)
        CALL FMSOV (N1, IDTYPE, N2  , 'LU_A21', LU_A21)
        CALL CNDI  (LOWEQ, N2,           'LU_A22', LU_A22)
        CALL FMSOM (LU_A22, 'LU_F22', LU_F22)
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_B1' , LU_B1 )
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_X1' , LU_X1 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_B2' , LU_B2 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_S2' , LU_S2 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_X2' , LU_X2 )
C
C (3)   Write data to FMS files:
        NEWA11 = .TRUE.
        NEWA12 = .TRUE.
        NEWA21 = .TRUE.
        NEWA22 = .TRUE.
        NEWB1  = .TRUE.
        NEWB2  = .TRUE.
        ICK_X2 = .TRUE.
C
C       Branch back to here for refinement solutions:
   30   CONTINUE
C
C       Generate [A11]:
```

```
      IF(NEWA11) CALL A11GEN (CMD(L_TEMP), N1, N2,   LU_A11)
C
C     Generate [A12]:
      IF(NEWA12) CALL A12GEN (CMD(L_TEMP), N1, N2,   LU_A12)
C
C     Generate [A21]:
      IF(NEWA21) CALL A21GEN (CMD(L_TEMP), N1, N2,   LU_A21)
C
C     Generate [A22]:
      IF(NEWA22) CALL A22GEN (CMD(L_TEMP), N2,       LU_A22)
C
C     Generate {B1}:
      IF(NEWB1 ) CALL B1GEN  (CMD(L_TEMP), N1, NRHS, LU_B1 )
C
C     Generate {B2}:
      IF(NEWB2 ) CALL B2GEN  (CMD(L_TEMP), N2, NRHS, LU_B2 )
C
C (4) Perform matrix algebra:
      CALL RESOLV
     1 (LU_A11, LU_A11,
     2  LU_A12, LU_A12,
     3  LU_A21,
     4  LU_A22, LU_F22,
     5  LU_B1,  LU_B1, LU_X1,
     6  LU_B2,  LU_S2, LU_X2,
     7  NEWA11, NEWA12,
     8  NEWA21, NEWA22,
     9  NEWB1,  NEWB2,
     1  ICK_X2, E2MAX)
C
C (5) Read data from FMS files:
C     Check the answer:
      ERROR  = E2MAX
C
C     Initialize FMSGET:
      CALL FMSGET (LU_X1, 0, 0, 0, 0, CMD, 0)
      DO 60 IRHS = 1,NRHS
C
C        Check {X1}:
         CALL FMSGET (LU_X1, N1, 1, 1, IRHS, CMD(L_TEMP), N1)
         DO 51 I = 0,(N1-1)
            EI = ABS(CMD(L_TEMP+I) - C_ONE)
            IF(EI .GT. ERROR) ERROR = EI
   51    CONTINUE
   60 CONTINUE
C
C     End FMSGET:
      CALL FMSGET (LU_X1, 0, 0, N1+1, 1, CMD, 0)
      WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C     Modify the matrix and solve it again:
      IF(.NOT. ASK('Do you want another solution?')) GO TO 100
      NEWA11 = ASK('Do you want a new A11?')
      NEWA12 = ASK('Do you want a new A12?')
      NEWA21 = ASK('Do you want a new A21?')
      NEWA22 = ASK('Do you want a new A22?')
      NEWB1  = ASK('Do you want a new B1?' )
      NEWB2  = ASK('Do you want a new B2?' )
C
C     We have chosen to overlay factored data on input matrices.
C     Therefore if new factors must be recomputed, the arrays must
C     be reset to the unfactored values.
C     Based on the options selected above, these conditions become:
      IF( NEWA11 ) THEN
          NEWA12 = .TRUE.
```

```
            NEWA22 = .TRUE.
            NEWB1  = .TRUE.
            NEWB2  = .TRUE.
         END IF
         IF( NEWA12 ) NEWA22 = .TRUE.
         IF( NEWB1  ) NEWB2  = .TRUE.
         IF( NEWA11 .OR.
     1       NEWA12 .OR.
     2       NEWA21 .OR.
     3       NEWA22 .OR.
     4       NEWB1  .OR.
     5       NEWB2) GO TO 30
C
C (6)   Close FMS files:
  100   CONTINUE
        CALL FMSCM (LU_A11)
        CALL FMSCV (LU_A12)
        CALL FMSCV (LU_A21)
        CALL FMSCM (LU_A22)
        CALL FMSCM (LU_F22)
        CALL FMSCV (LU_B1 )
        CALL FMSCV (LU_X1 )
        CALL FMSCV (LU_B2 )
        CALL FMSCV (LU_S2 )
        CALL FMSCV (LU_X2 )
        CALL FMSCMR (CMD, L_TEMP, NMAX)
        CALL FMSPOP (MYNAME)
        CALL FMSEND
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE A11GEN (ACOL, N1, N2, LU_A11)
C=======================================================================
        COMPLEX*16 ACOL(N1)
        INTEGER    N1, N2
        INTEGER    LU_A11(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A11GEN')
        REAL*8     R_ZERO, A11_Re
        DATA       R_ZERO/0.0D0/
```

```
      COMPLEX*16 C_MONE, C_ZERO, C_ONE
      DATA       C_MONE/(-1.0D0,0.0D0)/
      DATA       C_ZERO/( 0.0D0,0.0D0)/
      DATA       C_ONE /( 1.0D0,0.0D0)/
C
      CALL FMSPSH (MYNAME)
      WRITE(6,2000)
C
C     Initialize FMSCOL
      CALL FMSCOL (0, ACOL, LU_A11)
C
C     Write first column
      DO 10 I = 2,N1
         ACOL(I) = C_MONE
10    CONTINUE
      A11_Re  = DFLOAT(N1+N2)
      ACOL(1) = DCMPLX(A11_Re,R_ZERO)
      CALL FMSCOL (1, ACOL, LU_A11)
C
C     Write remaining columns:
      DO 20 I = 2,N1
         ACOL(I) = C_ZERO
20    CONTINUE
      ACOL(1) = C_MONE
      DO 30 I = 2,N1
         ACOL(I) = C_ONE
         CALL FMSCOL (I, ACOL, LU_A11)
         ACOL(I) = C_ZERO
30    CONTINUE
C
C     End FMSCOL
      CALL FMSCOL (N1+1, ACOL, LU_A11)
      CALL FMSPOP (MYNAME)
      RETURN
2000  FORMAT (' Populating [A11]:')
      END
C=======================================================================
      SUBROUTINE A12GEN (ACOL, N1, N2, LU_A12)
C=======================================================================
      COMPLEX*16 ACOL(N1)
      INTEGER    N1, N2
      INTEGER    LU_A12(25)
      CHARACTER*6 MYNAME
      PARAMETER (MYNAME='A12GEN')
      COMPLEX*16 C_MONE, C_ZERO
      DATA       C_MONE/(-1.0D0,0.0D0)/
      DATA       C_ZERO/( 0.0D0,0.0D0)/
C
      CALL FMSPSH (MYNAME)
      WRITE(6,2000)
C
      DO 10 I=2,N1
         ACOL(I) = C_ZERO
10    CONTINUE
      ACOL(1) = C_MONE
C
C     Initialize FMSPUT:
      CALL FMSPUT (LU_A12, 0, 0, 0, 0, ACOL, 0)
C
C     Write out columns of [A12]:
      DO 20 I = 1,N2
         CALL FMSPUT (LU_A12, N1, 1, 1, I, ACOL, N1)
20    CONTINUE
C
C     End FMSPUT:
```

```
        CALL FMSPUT (LU_A12, 0, 0, N1+1, 0, ACOL, 0)
C
C       Set status:
        LU_A12(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating [A12]:')
        END
C=======================================================================
        SUBROUTINE A21GEN (AROW, N1, N2, LU_A21)
C=======================================================================
        COMPLEX*16 AROW(N1)
        INTEGER    N1, N2
        INTEGER    LU_A21(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A21GEN')
        COMPLEX*16 C_MONE, C_ZERO
        DATA       C_MONE/(-1.0D0,0.0D0)/
        DATA       C_ZERO/( 0.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
           AROW(I) = C_ZERO
   10   CONTINUE
        AROW(1) = C_MONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_A21, 0, 0, 0, 0, AROW, 0)
C
C       Write out rows of [A21], which is stored transposed:
        DO 20 I = 1,N2
           CALL FMSPUT (LU_A21, N1, 1, 1, I, AROW, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_A21, 0, 0, N1+1, 0, AROW, 0)
C
C       Set status:
        LU_A21(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating [A21]:')
        END
C=======================================================================
        SUBROUTINE A22GEN (ACOL, N2, LU_A22)
C=======================================================================
        COMPLEX*16 ACOL(N2)
        INTEGER    N2
        INTEGER    LU_A22(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A22GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA       C_ZERO/( 0.0D0,0.0D0)/
        DATA       C_ONE /( 1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
C       Initialize FMSCOL
        CALL FMSCOL (0, ACOL, LU_A22)
C
        DO 10 I = 1,N2
           ACOL(I) = C ZERO
```

```
   10   CONTINUE
C
        DO 20 I = 1,N2
           ACOL(I) = C_ONE
           CALL FMSCOL (I, ACOL, LU_A22)
           ACOL(I) = C_ZERO
   20   CONTINUE
C
C       End FMSCOL
        CALL FMSCOL (N2+1, ACOL, LU_A22)
C
C       Set status:
        LU_A22(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating [A22]:')
        END
C=======================================================================
        SUBROUTINE B1GEN (RHS, N1, NRHS, LU_B1)
C=======================================================================
        COMPLEX*16 RHS(N1)
        INTEGER    N1, NRHS
        INTEGER    LU_B1(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B1GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA       C_ZERO/(0.0D0,0.0D0)/
        DATA       C_ONE /(1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
          RHS(I) = C_ZERO
   10   CONTINUE
        RHS(1) = C_ONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B1}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B1, N1, 1, 1, I, RHS, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, N1+1, 0, RHS, 0)
C
C       Set status:
        LU_B1(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating {B1}:')
        END
C=======================================================================
        SUBROUTINE B2GEN (RHS, N2, NRHS, LU_B2)
C=======================================================================
        COMPLEX*16 RHS(N2)
        INTEGER    N2, NRHS
        INTEGER    LU_B2(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B2GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA       C_ZERO/(0.0D0,0.0D0)/
```

```
        DATA       C_ONE /(1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
        DO 10 I=1,N2
          RHS(I) = C_ZERO
   10   CONTINUE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B2, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B2}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B2, N2, 1, 1, I, RHS, N2)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_B2, 0, 0, N2+1, 0, RHS, 0)
C
C       Set status:
        LU_B2(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating {B2}:')
        END
C=======================================================================
        SUBROUTINE RESOLV
     1  (LU_A11, LU_F11,
     2   LU_A12, LU_F12,
     2   LU_A21,
     4   LU_A22, LU_F22,
     3   LU_B1,  LU_S1, LU_X1,
     4   LU_B2,  LU_S2, LU_X2,
     5   NEWA11, NEWA12,
     6   NEWA21, NEWA22,
     7   NEWB1,  NEWB2,
     6   ICK_X2, E2MAX)
C=======================================================================
C
C       DESCRIPTION:
C          This subroutine solves the linear system
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | | X1 |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | | X2 |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C          when all or only part of the matrix terms change.
C
C
C          On input, the following matrix terms are specified:
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | |    |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | |    |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          The following intermediate work arrays are used:
C
C          +-    +    -+ +-  -+   +-  -+
C          |     |     | | S1 |   |    |
```

```
C          +-----+-----+ +----+ = +----+
C          |     |     | | S2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          On output, the following terms are returned:
C
C          +-    +    -+ +-  -+   +-  -+
C          | F11 | F12 | | X1 |   |    |
C          +-----+-----+ +----+ = +----+
C          |     | F22 | | X2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
C
C       You may overlay any of the corresponding input, intermediate
C       or output terms to save storage by specifying the same actual
C       parameter on the subroutine call.
C
C       Arrays containing input data may share storage with
C       intermediate and output arrays if these conditions are met:
C       1) If any value changes in the input array, all values in that
C          input array are redefined.
C       2) In defining the new values for an array, the previous
C          values are not required. (They are no longer available
C          because they have been overwritten)
C       If these conditions are true, then the following arrays can
C       share the same storage:
C          [A11] & [F11],
C          [A12] & [F12],
C          [A22] & [F22],
C          {B1}  &  {S1},
C          {B2}  &  {S2}.
C
C       Ths storage for {S1} and {S2} can be shared with {X1} and {X2}.
C       No additional computation is required if the following
C       conditions are met:
C          {S1} & {X1} : if [A11]      or {B1}         change,
C          {S2} & {X2} : if [A11], [A12], {B1} or {B2} change.
C       If these conditions are not met and the storage is shared, then
C       additional computation will be performed.
C
C       You may also overlay {B1},{S1},{X1} or {B2},{S2},{X2} if
C       all the above conditions are met.
C
C       FORMAL PARAMETERS:
C          (R ) LU_A11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A11]
C
C          (R ) LU_F11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F11]
C                             May be the same as LU_A11.
C
C          (R ) LU_A12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A12]
C
C          (R ) LU_F12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [F12]
C                             May be the same as LU_A12.
C
C          (R ) LU_A21(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A21]
C                             Data on this file is not changed.
C
C          (R ) LU_A22(25)  = INTEGER ARRAY
C                             FMS mtarix file attributes for [A22]
C
```

```
C          (R ) LU_F22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F22]
C
C          (R ) LU_B1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B1}
C
C          (R ) LU_S1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {S1}.
C                             This is an intermediate work array.
C                             May be the same value as LU_B1.
C
C          (R ) LU_X1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X1}.
C                             May be the same as LU_S1.
C
C          (R ) LU_B2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B2}
C
C          (R ) LU_S2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {S2}.
C                             This is an intermediate work array.
C                             May be the same value as LU_B2.
C
C          (R ) LU_X2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X2}.
C                             May be the same as LU_B2.
C
C          (R ) NEWA11      = LOGICAL
C                             Indicates that [A11] has changed.
C
C          (R ) NEWA12      = LOGICAL
C                             Indicated that [A12] has changed.
C
C          (R ) NEWA21      = LOGICAL
C                             Indicated that [A21] has changed.
C
C          (R ) NEWA22      = LOGICAL
C                             Indicated that [A22] has changed.
C
C          (R ) NEWB1       = Logical
C                             Indicates that {B1} has changed.
C
C          (R ) NEWB2       = Logical
C                             Indicates that {B2} has changed.
C
C          (R ) ICK_X2      = Logical
C                             Check the values in {X2}.
C                             This requires access to [A22] and {B2}.
C                             Requires the following:
C                             LU_S2  or LU_X2 do not overlay LU_B2
C
C          (R ) E2MAX       = REAL*8
C                             Maximum error in {X2}.
C
C       FMSCOM PARAMETERS:
C          (RW) IACCOM      = Multiply accumulate flag
C          (R ) LUPR        = FORTRAN unit for printing
C
C       CALLED FROM:
C          EXAMPLE_14
C
C       SUBPROGRAMS USED:
C          CNDF
C          CNDS
```

```
C          FMSMM
C
C       ERROR CONDITIONS:
C          None.
C
C       HISTORY:
C
C       NOTES:
C
C       Expanding the partitioned system gives:
C
C       [A11]{X1} + [A12]{X2} = {B1}      (1)
C       [A21]{X1} + [A22]{X2} = {B2}      (2)
C
C       Premultiplying (1) by inv[A11] gives
C
C       {X1} = {S1} - [F12]{X2}           (3)
C
C       where
C          {S1}  = inv[A11]{B1}           (4)
C          [F12] = inv[A11][A12]          (5)
C
C       Substituting (3) into (2) gives:
C
C       [F22]{X2} = {S2}                  (6)
C
C       where
C          [F22] = [A22] - [A21][F12]     (7)
C          {S2}  = {B2}  - [A21]{S1}      (8)
C
C
C       The error in {X2} may be computed from:
C
C          E2MAX = | {B2} - [A21]{X1} - [A22]{X2} |
C
C       The algorithm consists of the following 8 steps:
C
C       Solve large system:
C       1. Factor   [A11] into [F11]=[L11][U11]
C       2. Solve    [F11][F12] = [A12]            (eq. 5)
C       3. Solve    [F11]{S1}  = {B1}             (eq. 4)
C
C       Project to small system:
C       4. Multiply [F22'] = [A22] - [A21][F12]   (eq. 7)
C       5. Factor   [F22'] into [F22]=[L22][U22]
C       6. Multiply {S2}   = {B2}  - [A21]{S1}    (eq. 8)
C
C       Solve small system:
C       7. Solve    [F22]{X2} = {S2}              (eq. 6)
C
C       Project back to large system:
C       8. Multiply {X1} = {S1} - [F12]*{X2}      (eq. 3)
C
C       Check {X2}: (optional)
C       9. Multiply {ERR} = {B2} - [A21]{X1} - [A22]{X2}
C          Then compute the maximum absolute value in {ERR}.
C
C       Summary of arrays used:
C
C       STEP  A11 F11 A12 F12 A21 A22 F22  B1  S1  X1  B2  S2  X2
C       ====  ======= ======= === ======= =========== ===========
C         1    R-->W
C         2        R   R-->W
C         3        R                       R-->W
C         4               R   R   R-->W
```

```
C         5                           RW
C         6                   R               R       R-->W
C         7                           R                   R-->W
C         8               R                   R-->W           R
C
C       Steps required:
C
C       STEP1 = (NEWA11                 )
C       STEP2 = (NEWA12 | STEP1         )
C       STEP3 = (NEWB1  | STEP1         )
C       STEP4 = (NEWA22 | NEWA21 | STEP2 )
C       STEP5 = (STEP4                  )
C       STEP6 = (NEWB2  | NEWA21 | STEP3 )
C       STEP7 = (STEP5  | STEP6         )
C       STEP8 = (STEP3  | STEP2  | STEP7 )
C
C       or, substituting,
C
C       STEP1 = (NEWA11                                         )
C       STEP2 = (NEWA11 | NEWA12                                )
C       STEP3 = (NEWA11 |                           NEWB1       )
C       STEP4 = (NEWA11 | NEWA12 | NEWA21 | NEWA22              )
C       STEP5 = (NEWA11 | NEWA12 | NEWA21 | NEWA22              )
C       STEP6 = (NEWA11 |          NEWA21 |          NEWB1 | NEWB2 )
C       STEP7 = (NEWA11 | NEWA12 | NEWA21 | NEWA22 | NEWB1 | NEWB2 )
C       STEP8 = (NEWA11 | NEWA12 | NEWA21 | NEWA22 | NEWB1 | NEWB2 )
C
C       Also if the intermediate vector storage is overlaid on the
C       storage for the solution vectors, it is assumed that it no
C       longer contains the intermediate arrays.  Therefore,
C       STEP3 = (STEP3  | (LU_S1 = LU_X1) )
C       STEP6 = (STEP6  | (LU_S2 = LU_X2) )
C-----------------------------------------------------------------------
C       FORMAL PARAMETERS
C-----------------------------------------------------------------------
        INTEGER    LU_A11(25)
        INTEGER    LU_F11(25)
C
        INTEGER    LU_A12(25)
        INTEGER    LU_F12(25)
C
        INTEGER    LU_A21(25)
C
        INTEGER    LU_A22(25)
C
        INTEGER    LU_F22(25)
C
        INTEGER    LU_B1 (25)
        INTEGER    LU_S1 (25)
        INTEGER    LU_X1 (25)
C
        INTEGER    LU_B2 (25)
        INTEGER    LU_S2 (25)
        INTEGER    LU_X2 (25)
C
        LOGICAL    NEWA11
        LOGICAL    NEWA12
        LOGICAL    NEWA21
        LOGICAL    NEWA22
        LOGICAL    NEWB1
        LOGICAL    NEWB2
        LOGICAL    ICK_X2
        REAL*8     E2MAX
C-----------------------------------------------------------------------
C       LOCAL VARIABLES
```

```
C-----------------------------------------------------------------------
C
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='RESOLV')
C
C       FMS Memory management:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
        COMMON /MYFMS/ L_A22, LDA22
C
C       Data type = complex:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE = 2)
C
C       Skip operations during solving (no):
        INTEGER    ISKIP
        PARAMETER (ISKIP=0)
C
C       Constants:
        COMPLEX*16 C_ONE(1)
        DATA       C_ONE  /(1.0D0,0.0D0)/
        REAL*8     R_ZERO
        DATA       R_ZERO / 0.0D0/
        REAL*8     R_ONE
        DATA       R_ONE  / 1.0D0/
        REAL*8     R_THREE
        DATA       R_THREE/ 3.0D0/
        REAL*8     R_EIGHT
        DATA       R_EIGHT/ 8.0D0/
C
C       FMS Parameters:
        INTEGER    LUPR
C
C       Matrix dimensions:
        INTEGER    N1
        INTEGER    N2
        INTEGER    NRHS
        REAL*8     RN1
        REAL*8     RN2
        REAL*8     RNRHS
C
C       Temporary file:
        INTEGER    LU_ERR(25)
        INTEGER     LUA0  (25)
        DATA        LUA0(1)/0/
        INTEGER     LUX0  (25)
        INTEGER     LUS0  (25)
C
C       CPU, Wall and I/O time:
        REAL*8     TCPU(10)
        REAL*8     TWALL(10)
        REAL*8     TIO(10)
        REAL*8     T1, T2, T3
C
C       Steps to perform:
        LOGICAL    STEP1
        LOGICAL    STEP2
        LOGICAL    STEP3
        LOGICAL    STEP4
```

```
      LOGICAL    STEP5
      LOGICAL    STEP6
      LOGICAL    STEP7
      LOGICAL    STEP8
      LOGICAL    STEP9
C
C     Floating point operations:
      REAL*8     OPS(9), TOPS
C-----------------------------------------------------------------------
C     FORTRAN PROCEDURE
C-----------------------------------------------------------------------
      CALL FMSPSH (MYNAME)
C
C     Get the FORTRAN unit for printng:
      CALL FMSIGT ('LUPR', LUPR)
C
C     Get the memory pointer for arrays IMD, RMD and CMD:
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C     Determine the size of the matrices:
      N1    = LU_A11(8)
      N2    = LU_A22(8)
      NRHS  = LU_B1 (6)
      NUMEQ = N1 + N2
      RN1   = DFLOAT(N1)
      RN2   = DFLOAT(N2)
      RNRHS = DFLOAT(NRHS)
      WRITE(LUPR,2000) NUMEQ, N1, N2, NRHS
C
C     Determine which steps to do:
      STEP1 = (NEWA11                        )
      STEP2 = (NEWA12 .OR. STEP1             )
      STEP3 = (NEWB1  .OR. STEP1             )
      STEP4 = (NEWA22 .OR. NEWA21 .OR. STEP2 )
      STEP5 = (STEP4                         )
      STEP6 = (NEWB2  .OR. NEWA21 .OR. STEP3 )
      STEP7 = (STEP5  .OR. STEP6             )
      STEP8 = (STEP2  .OR. STEP3  .OR. STEP7 )
      STEP9 = (ICK_X2                        )
C
C     Check overlaid storage:
      STEP3 = (STEP3  .OR. (LU_S1(1) .EQ. LU_X1(1)) )
      STEP6 = (STEP6  .OR. (LU_S2(2) .EQ. LU_X2(2)) )
      IF( (LU_B2(1)  .EQ. LU_S2(1) )   .OR.
     1      (LU_B2(1)  .EQ. LU_X2(1) ) ) STEP9 = .FALSE.
C
C     Initialize operation counts for each step:
      OPS(1) = R_ZERO
      OPS(2) = R_ZERO
      OPS(3) = R_ZERO
      OPS(4) = R_ZERO
      OPS(5) = R_ZERO
      OPS(6) = R_ZERO
      OPS(7) = R_ZERO
      OPS(8) = R_ZERO
      OPS(9) = R_ZERO
C
C (1)   Factor [A11] into [L11][U11]:
      CALL FMSTIM (TCPU(1), TWALL(1), TIO(1))
      IF(STEP1) THEN
         WRITE(LUPR,2001)
         OPS(1) = R_EIGHT*RN1*RN1*RN1/R_THREE
         CALL CNDAF (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11,
```

```
     1                LUX0, LUX0, 0)
         END IF
C
C (2)   Solve [F11][F12] = [A12]:
         CALL FMSTIM (TCPU(2), TWALL(2), TIO(2))
         IF(STEP2) THEN
            WRITE(LUPR,2002)
            OPS(2) = R_EIGHT*RN1*RN1*RN2
            CALL CNDS  (LU_F11, LU_A12, LU_F12, N2,   0)
         END IF
C
C (3)   Solve [F11]{S1}  = {B1}:
         CALL FMSTIM (TCPU(3), TWALL(3), TIO(3))
         IF(STEP3)  THEN
            WRITE(LUPR,2003)
            OPS(3) = R_EIGHT*RN1*RN1*RNRHS
            CALL CNDS  (LU_F11, LU_B1,  LU_S1,  NRHS, 0)
         END IF
C
C (4)   Multiply [F22'] = [A22] - [A21][F12]
         CALL FMSTIM (TCPU(4), TWALL(4), TIO(4))
         IF(STEP4)  THEN
            WRITE(LUPR,2004)
            OPS(4) = R_EIGHT*RN1*RN2*RN2
            CALL FMSMM (LU_F22, LU_A22, -1, 'T', LU_A21, LU_F12)
         END IF
C
C (5)   Factor [F22'] into [F22]:
         CALL FMSTIM (TCPU(5), TWALL(5), TIO(5))
         IF(STEP5) THEN
            WRITE(LUPR,2005)
            OPS(5) = R_EIGHT*RN2*RN2*RN2/R_THREE
            CALL CNDAF (LU_F22, C_ONE, 1, LUS0, 0, LUA0, LU_F22,
     1                LUX0, LUX0, 0)
         END IF
C
C (6)   Multiply {S2}  = {B2}  - [A21]{S1}
         CALL FMSTIM (TCPU(6), TWALL(6), TIO(6))
         IF(STEP6) THEN
            WRITE(LUPR,2006)
            OPS(6) = R_EIGHT*RN2*RN1*RNRHS
            CALL FMSMM (LU_S2, LU_B2, -1, 'T', LU_A21, LU_S1)
         END IF
C
C (7)   Solve [F22]{X2} = {S2}:
         CALL FMSTIM (TCPU(7), TWALL(7), TIO(7))
         IF(STEP7) THEN
            WRITE(LUPR,2007)
            OPS(7) = R_EIGHT*RN2*RN2*RNRHS
            CALL CNDS  (LU_F22, LU_S2, LU_X2, NRHS, 0)
         END IF
C
C (8)   Multiply {X1} = {S1} - [F12]{X2}
         CALL FMSTIM (TCPU(8), TWALL(8), TIO(8))
         IF(STEP8) THEN
            WRITE(LUPR,2008)
            OPS(8) = R_EIGHT*RN1*RN2*RNRHS
            CALL FMSMM (LU_X1, LU_S1, -1, 'N', LU_F12, LU_X2)
         END IF
C
C (9)   Check {X2}: {ERR} = {B2} - [A21]{X1} - [A22]{X2}
         CALL FMSTIM (TCPU(9), TWALL(9), TIO(9))
         IF(STEP9) THEN
            WRITE(LUPR,2009)
            OPS(9) = R EIGHT*(RN1+RN2)*RN2*RNRHS
```

```
C
C          Open a vector file to store the error array ERR(N2,NRHS):
           CALL FMSOV (N2, IDTYPE, NRHS, 'LU_ERR' , LU_ERR)
C
C          Multiply {ERR} = {B2} - [A21]{X1}
           CALL FMSMM (LU_ERR, LU_B2, -1, 'T', LU_A21, LU_X1)
C
C          Multiply {ERR} = {ERR} - [A22]{X2}:
           CALL FMSMM (LU_ERR, LU_ERR, -1, 'N', LU_A22, LU_X2)
C
C          Read LU_ERR and check the error:
C
C          Allocate storage for one vector:
           CALL FMSCMG (CMD, L_ERR, N2)
C
C          Initialize FMSGET:
           CALL FMSGET (LU_ERR, 0, 0, 0, 0, CMD, 0)
C
C          Compute the maximum error:
           E2MAX = R_ZERO
           DO 95 N = 1,NRHS
C             Read in next vector:
              CALL FMSGET (LU_ERR, N2, 1, 1, N, CMD(L_ERR), N2)
              L = L_ERR
              DO 90 I = 1,N2
                 T1 = ABS(CMD(L))
                 IF(T1 .GT. E2MAX) E2MAX = T1
                 L = L + 1
   90         CONTINUE
   95      CONTINUE
C
C          End FMSGET:
           CALL FMSGET (LU_ERR, 0, 0, N2+1, 1, CMD, 0)
C
C          Close the vector file LU_ERR:
           CALL FMSCV(LU_ERR)
C
C          Delete the storage for {ERR}:
           CALL FMSCMR (CMD, L_ERR, N2)
        END IF
C
C       Print timings:
        WRITE(LUPR,2010)
        CALL FMSTIM (TCPU(10), TWALL(10), TIO(10))
        TOPS = R_ZERO
        DO 100 ISTEP = 1,9
           IF(OPS(ISTEP) .EQ. R_ZERO) THEN
C
C             Operation skipped:
              WRITE(LUPR,2012) ISTEP
           ELSE
              TOPS = TOPS + OPS(ISTEP)
C
C             CPU time and Mflops:
              T1 = TCPU(ISTEP+1)  - TCPU(ISTEP)
              IF(T1 .GT. R_ZERO) THEN
                 MF1   = ((1.0D-6)*OPS(ISTEP)/T1)
              ELSE
                 MF1   = 0
              END IF
C
C             Wall time, Mflops and ratio:
              T2 = TWALL(ISTEP+1) - TWALL(ISTEP)
              T3 = TIO(ISTEP+1)   - TIO(ISTEP)
```

```
               IF((T2 .GT. R_ZERO) .AND.
     1            (T2 .GT.  T3   ) ) THEN
                  MF2   = ((1.0D-6)*OPS(ISTEP)/T2)
                  RATIO = T1/(T2-T3)
               ELSE
                  MF2   = 0
                  RATIO = R_ONE
               END IF
               WRITE(LUPR,2011) ISTEP, T1, T2, T3, MF1, MF2, RATIO
            END IF
  100    CONTINUE
         T1 = TCPU(10)  - TCPU(1)
         T2 = TWALL(10) - TWALL(1)
         T3 = TIO(10)   - TIO(1)
         IF(T1 .GT. R_ZERO) THEN
            MF1   = ((1.0D-6)*TOPS/T1)
         ELSE
            MF1   = 0
         END IF
         IF(T2 .GT. R_ZERO) THEN
            MF2   = ((1.0D-6)*TOPS/T2)
            RATIO = T1/(T2-T3)
         ELSE
            MF2   = 0
            RATIO = R_ONE
         END IF
         WRITE(LUPR,2013) T1, T2, T3, MF1, MF2, RATIO
         CALL FMSPOP (MYNAME)
         RETURN
 2000    FORMAT (/
     1   ' Start of Subroutine RESOLV',/
     2   ' Rank of full matrix.....=',I6/
     3   ' Rank of submatrix #1....=',I6/
     4   ' Rank of submatrix #2....=',I6/
     5   ' # of right-hand-sides...=',I6/)
 2001    FORMAT (/
     1   ' STEP 1. Factor   [A11] into [F11]'/
     2   ' ================================')
 2002    FORMAT (/
     1   ' STEP 2. Solve    [F11][F12] = [A12]'/
     2   ' ================================')
 2003    FORMAT (/
     1   ' STEP 3. Solve    [F11]{S1}  = {B1}'/
     2   ' ================================')
 2004    FORMAT (/
     1   ' STEP 4. Multiply [F22] = [A22] - [A21][F12]'/
     2   ' ========================================')
 2005    FORMAT (/
     1   ' STEP 5. Factor   [F22]'/
     2   ' ======================')
 2006    FORMAT (/
     1   ' STEP 6. Multiply {S2}  = {B2}  - [A21]{S1}'/
     2   ' ========================================')
 2007    FORMAT (/
     1   ' STEP 7. Solve    [F22]{X2} = {S2}'/
     2   ' ================================')
 2008    FORMAT (/
     1   ' STEP 8. Multiply {X1} = {S1} - [A12]*{X2}'/
     2   ' ======================================')
 2009    FORMAT (/
     1   ' STEP 9. Checking values in {X2}'/
     2   ' ======================================')
 2010    FORMAT (/
     1   ' Summary of Subroutine RESOLV:'//
     2   '          <----------TIMES (Sec.)----------->'
```

```
     2  '    <--MFLOPS-->'/
     2  '  STEP          CPU          WALL          I/O     CPU    WALL',
     2  '    RATIO'/
     3  ' =====    =========    =========    =========    ====    ====',
     3  '    =====')
  2011   FORMAT (I6,3F13.3,2I8,F9.3)
  2012   FORMAT (I6,6X,'Skipped')
  2013   FORMAT (' TOTAL',3F13.3,2I8,F9.3)
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 15. USING FMSMM AND THE FMS PARAMETERS MMROW, MMCOL AND MMKA

```
C       The FMS Out-of-Core matrix multiply routine FMSMM forms the
C       product of two matrixes [A][B] and stores the result in matrix
C       [C].  Under some matrix partitioning conditions, the size of
C       the product [AB] = [A][B] will be different than the matrix [C].
C       Normally, FMS will store the product [AB] starting at the upper
C       left corner of [C], C(1,1).  However, it may be necessary to
C       shift the starting position of the product [AB] in [C] to some
C       other position.  The FMS Parameters MMROW and MMCOL specify
C       the number of rows and columns to shift the product [AB] in
C       [C], storing [AB] in [C] starting at C(1+MMROW,1+MMCOL).  The
C       default values of MMROW and MMCOL are 0.
C
C       Another shifting requirement occurs when the matrix [A] has
C       fewer columns than the matrix [B] has rows.  In this case, FMS
C       provides the shifting parameter MMKA, which has the effect of
C       adding MMKA columns of 0's to matrix [A].  The matrix product
C       then becomes
C
C       [AB] = [0|A][B]
C
C       The following example illustrates the use of FMSMM and these
C       FMS Parameters.
C
C       The system of linear equations is partitioned as follows:
C
C          --   +-    +     +    -+ +-  -+   +-  -+
C          N1   | A11 | A12 | A13 | | X1 |   | B1 |
C          --   +-----+-----+-----+ +----+   +----+
C          N2   | A21 |           | |    | = |    |
C          --   +-----+    A22    + + X2 +   + B2 +
```

```
C          N3   | A31 |           | |    |   |    |
C          --   +-    +-         -+ +-  -+   +-  -+
C
C               |< N1>|< N2>|< N3>| <NRHS>   <NRHS>
C                     |<---N22--->|
C
C       Because the matrices [A31] and [A13] do not lie on a boundary
C       of [A22], it is necessary to use the matrix multiply shifting
C       parameters during the partitioned solution of this system.
C       The following table lists the products computed by FMSMM and
C       the required values of the shifting parameters.
C
C       Matrix Product   Stored In   MMROW   MMCOL   MMKA
C       ==============   =========   =====   =====   ====
C       [A21][A12]       [A22]           0       0      0
C       [A21][A13]       [A22]           0      N2      0
C       [A21][X1]        [X2]            0       0      0
C       [A31][A12]       [A22]          N2       0      0
C       [A31][A13]       [A22]          N2      N2      0
C       [A31][X1]        [X2]           N2       0      0
C       [0|A13][X2]      [X1]            0       0     N2
C       [A12][X2]        [X1]            0       0      0
C
C       Normally FMS will overlay matrix factors and solution vectors
C       on the input matrix data.  However, for this example, we want
C       to save the original matrices to check the result.  Therefore
C       separate files will be used for the original matrix, factors
C       and the solution vectors.
C
C       The following FMS files will be used:
C
C       File     File
C       Name     Type     CONTENTS
C       ======   ======   ========
C       LU_A11   Matrix   [A11]
C       LU_F11   Matrix   [F11], factors of [A11]
C
C       LU_A12   Vector   [A12]
```

```
C       LU_F12   Vector   [F12], solution of [F11][F12]=[A12]
C
C       LU_A13   Vector   [A13]
C       LU_F13   Vector   [F13], solution of [F11][F13]=[A13]
C
C       LU_A21   Vector   [A21]T
C
C       LU_A31   Vector   [A31]T
C
C       LU_A22   Matrix   [A22]
C       LU_F22   Matrix   [F22], factors of [A22]-[A21|A31][F12|F13]
C
C       LU_B1    Vector   {B1}
C       LU_X1    Vector   {X1}
C
C       LU_B2    Vector   {B2}
C       LU_X2    Vector   {X2}
C
C       For symmetric problems, the files LU_A12 and LU_A13 are not
C       required, because they are the same as LU_A21 and LU_A31.
C
C       INPUT:
C       ======
C       The following input is used:
C          FMS Module (1=RS,2=RN,3=CH,4=CS,5=CN)
C          Number of equations in [A11], N1
C          Number of equations in [A21], N2
C          Number of equations in [A31], N2
C          Number of solution vectors, NRHS
C          Matrix test data (1=known, 2=random), MTEST
C          Any FMS Parameter
C
C
C       For known matrix test data (MTEST=1), the following pattern
C       is used:
C
C              [A]        {X}  =  {B}
C       +-            -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
```

```
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-            -+ +   +   +   +
C
C       where N=N1+N2+N3
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_15')
C
C       FMS matrix and vector file attributes:
        INTEGER     LU_A11(25), LU_F11(25)
        INTEGER     LU_A12(25), LU_F12(25)
        INTEGER     LU_A13(25), LU_F13(25)
        INTEGER     LU_A21(25)
        INTEGER     LU_A31(25)
        INTEGER     LU_A22(25), LU_F22(25)
        INTEGER     LU_B1 (25), LU_X1 (25)
        INTEGER     LU_B2 (25), LU_X2 (25)
        INTEGER     LUA0  (25)
        DATA        LUA0(1)/0/
        INTEGER     LUX0  (25)
        INTEGER     LUS0  (25)
C
C       FMS memory management requires the following arrays:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       LOCAL VARIABLES:
```

```
C       FMS module number (1 to 5)
           INTEGER MOD
C       Number of equations in [A11]:
           INTEGER N1
C       Number of equations in [A21]:
           INTEGER N2
C       Number of equations in [A31]:
           INTEGER N3
C       Number of equations in [A22]: (=N2+N3)
C          INTEGER N22
C       Number of solution vectors:
           INTEGER NRHS
C       FMS Data Type (1 or 2)
           INTEGER IDTYPE
C       FMS Matrix Symmetry (1, 2 or 3):
           INTEGER ISTYPE
C       Test matrix data type (1=fixed, 2=random):
           INTEGER MTEST
C       Error values:
           REAL*8  EI
           REAL*8  ERROR
C       Size of scratch work vector:
           INTEGER N_TEMP
C       RMD Pointer to scratch vector:
           INTEGER L_TEMP
C       Transpose of matrix A:
           CHARACTER*1 TRANSA
C
C       CONSTANTS:
           REAL*8     R_ZERO, R_ONE(1), R_BIG
           DATA R_ZERO/0.0D0/
           DATA R_ONE /1.0D0/
           DATA R_BIG /1.0D30/
           COMPLEX*16 C_ONE(1)
           DATA C_ONE /(1.0D0,0.0D0)/
C
C          Profile vector for a full matrix:
           INTEGER LOWEQ(1)
           DATA    LOWEQ/-1/
```

```
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
        CALL FMSIGT ('LUPR'  , LUPR)
C
C       Use block format matrix:
        CALL FMSIST ('MFMAT', 2)
C
C       Read in problem size parameters:
   10   CONTINUE
        WRITE (6,*) 'The FMS modules are numbered as follows:'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        MOD  = ASK_I('Enter the FMS module number (1 to 5)')
        IF( (MOD .LT. 1) .OR. (MOD. GT. 5) ) THEN
           PRINT *,'Illegal FMS Module entered.  Try again.'
           GO TO 10
        END IF
        N1   = ASK_I('Enter the number of equations in A11, N1')
        N2   = ASK_I('Enter the number of columns   in A12, N2')
        N3   = ASK_I('Enter the number of columns   in A13, N3')
        NRHS = ASK_I('Enter the number of solution vectors, NRHS')
        MTEST= ASK_I('Enter the test data (1=fixed,2=random)')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        N22   = N2 + N3
        NUMEQ = N1 + N22
        IF(MOD .LE. 2) THEN
C          Real*8 data:
           IDTYPE = 1
        ELSE
```

```
C          Complex*16 data:
           IDTYPE = 2
         END IF
         LENDIA = IDTYPE*NUMEQ
         TRANSA = 'T'
         IF     ((MOD .EQ. 1) .OR. (MOD .EQ. 4)) THEN
C          Symmetric matrix:
           ISTYPE = 1
         ELSE IF((MOD .EQ. 2) .OR. (MOD .EQ. 5)) THEN
C          Nonsymmetric matrix:
           ISTYPE = 2
         ELSE
C          Hermitian matrix:
           ISTYPE = 3
         END IF
C
C        Get a temporary vector for generating columns and RHS's:
         N_TEMP = IDTYPE*NUMEQ
         CALL FMSRMG (RMD, L_TEMP, N_TEMP)
C
C (2)    Open FMS files:
C
C        Matrix files:
C
C        [A11]:
         IF(N1 .GT. 0) THEN
            IF     (MOD.EQ.1) THEN
               CALL RSDI  (LOWEQ, N1 , 'LU_A11', LU_A11)
            ELSE IF(MOD.EQ.2) THEN
               CALL RNDI  (LOWEQ, N1 , 'LU_A11', LU_A11)
            ELSE IF(MOD.EQ.3) THEN
               CALL CHDI  (LOWEQ, N1 , 'LU_A11', LU_A11)
            ELSE IF(MOD.EQ.4) THEN
               CALL CSDI  (LOWEQ, N1 , 'LU_A11', LU_A11)
            ELSE
               CALL CNDI  (LOWEQ, N1 , 'LU_A11', LU_A11)
            END IF
            CALL FMSOM  (LU_A11, 'LU_F11', LU_F11)
         END IF
```

```
C
C       [A22]:
        IF(N22 .GT. 0) THEN
           IF     (MOD.EQ.1) THEN
              CALL RSDI  (LOWEQ, N22, 'LU_A22', LU_A22)
           ELSE IF(MOD.EQ.2) THEN
              CALL RNDI  (LOWEQ, N22, 'LU_A22', LU_A22)
           ELSE IF(MOD.EQ.3) THEN
              CALL CHDI  (LOWEQ, N22, 'LU_A22', LU_A22)
           ELSE IF(MOD.EQ.4) THEN
              CALL CSDI  (LOWEQ, N22, 'LU_A22', LU_A22)
           ELSE
              CALL CNDI  (LOWEQ, N22, 'LU_A22', LU_A22)
           END IF
           CALL FMSOM  (LU_A22, 'LU_F22', LU_F22)
        END IF
C
C       Vector and off-diagonal matrix files:
        IF(N1 .GT. 0) THEN
           IF(ISTYPE .NE. 1) THEN
              IF(N2 .GT. 0) THEN
                 CALL FMSOV (N1, IDTYPE, N2, 'LU_A12', LU_A12)
              END IF
              IF(N3 .GT. 0) THEN
                 CALL FMSOV (N1, IDTYPE, N3, 'LU_A13', LU_A13)
              END IF
           END IF
           IF(N2 .GT. 0) THEN
              CALL FMSOV (N1, IDTYPE, N2, 'LU_A21', LU_A21)
              CALL FMSOV (N1, IDTYPE, N2, 'LU_F12', LU_F12)
           END IF
           IF(N3 .GT. 0) THEN
              CALL FMSOV (N1, IDTYPE, N3, 'LU_A31', LU_A31)
              CALL FMSOV (N1, IDTYPE, N3, 'LU_F13', LU_F13)
           END IF
           CALL FMSOV (N1 , IDTYPE, NRHS, 'LU_B1' , LU_B1 )
           CALL FMSOV (N1 , IDTYPE, NRHS, 'LU_X1' , LU_X1 )
        END IF
```

```
         IF(N22 .GT. 0) THEN
            CALL FMSOV (N22, IDTYPE, NRHS, 'LU_B2' , LU_B2)
            CALL FMSOV (N22, IDTYPE, NRHS, 'LU_X2' , LU_X2)
         END IF
C
C (3)    Write data to FMS files:
C
C        Generate [A11]:
         IF(N1 .GT. 0) THEN
            CALL AGEN (RMD(L_TEMP), IDTYPE, 1, N1, 1, N1, NUMEQ, MTEST,
     1               LU_A11, ISTYPE)
C
            IF(ISTYPE .NE. 1) THEN
C
C              Generate [A12]:
               IF(N2 .GT. 0) THEN
                  CALL AGEN (RMD(L_TEMP), IDTYPE, 1, N1, N1+1, N1+N2,
     1                       NUMEQ, MTEST, LU_A12, ISTYPE)
               END IF
C
C              Generate [A13]:
               IF(N3 .GT. 0) THEN
               CALL AGEN (RMD(L_TEMP), IDTYPE, 1, N1, N1+N2+1, NUMEQ,
     1                    NUMEQ, MTEST, LU_A13, ISTYPE)
               END IF
            END IF
C
C           Generate [A21]:
            IF(N2 .GT. 0) THEN
               CALL AGEN (RMD(L_TEMP), IDTYPE, N1+1, N1+N2, 1, N1,
     1                    NUMEQ, MTEST, LU_A21, ISTYPE)
            END IF
C
C           Generate [A31]:
            IF(N3 .GT. 0) THEN
               CALL AGEN (RMD(L_TEMP), IDTYPE, N1+N2+1, NUMEQ, 1, N1,
     1                    NUMEQ, MTEST, LU_A31, ISTYPE)
            END IF
C
```

```
C          Generate {B1}:
           CALL BGEN  (RMD(L_TEMP), IDTYPE, 1, N1, NRHS, LU_B1)
         END IF
C
C        Generate [A22]:
         IF(N22 .GT. 0) THEN
           CALL AGEN (RMD(L_TEMP), IDTYPE, N1+1, NUMEQ, N1+1, NUMEQ,
     1               NUMEQ, MTEST, LU_A22, ISTYPE)
C
C          Generate {B2}:
           CALL BGEN  (RMD(L_TEMP), IDTYPE, N1+1, NUMEQ, NRHS, LU_B2)
         END IF
C
C (4)    Perform matrix algebra:
C
C (4.1) Factor [A11] into [F11]:
         IF(N1 .GT. 0) THEN
           WRITE(LUPR,2001)
           IF     (MOD .EQ. 1) THEN
              CALL RSDAF
     1        (LU_A11, R_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
           ELSE IF(MOD .EQ. 2) THEN
              CALL RNDAF
     1        (LU_A11, R_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
           ELSE IF(MOD .EQ. 3) THEN
              CALL CHDAF
     1        (LU_A11, R_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
           ELSE IF(MOD .EQ. 4) THEN
              CALL CSDAF
     1        (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
           ELSE
              CALL CNDAF
     1        (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
           END IF
         END IF
C
C (4.2) Solve [F11]{F12} = {A12}
         IF( (N1 .GT. 0) .AND.
     1       (N2 .GT. 0) ) THEN
```

```
            WRITE(LUPR,2002)
            IF     (MOD .EQ. 1) THEN
               CALL RSDS  (LU_F11, LU_A21, LU_F12, N2, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDS  (LU_F11, LU_A21, LU_F12, N2, 0)
            ELSE IF(MOD .EQ. 3) THEN
               CALL CHDS  (LU_F11, LU_A12, LU_F12, N2, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDS  (LU_F11, LU_A21, LU_F12, N2, 0)
            ELSE
               CALL CNDS  (LU_F11, LU_A12, LU_F12, N2, 0)
            END IF
         END IF
C
C (4.3) Solve [F11]{F13} = {A13}
         IF( (N1 .GT. 0) .AND.
     1       (N3 .GT. 0) ) THEN
            WRITE(LUPR,2003)
            IF     (MOD .EQ. 1) THEN
               CALL RSDS  (LU_F11, LU_A31, LU_F13, N3, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDS  (LU_F11, LU_A31, LU_F13, N3, 0)
            ELSE IF(MOD .EQ. 3) THEN
               CALL CHDS  (LU_F11, LU_A13, LU_F13, N3, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDS  (LU_F11, LU_A31, LU_F13, N3, 0)
            ELSE
               CALL CNDS  (LU_F11, LU_A13, LU_F13, N3, 0)
            END IF
         END IF
C
C (4.4) Solve [F11]{X1} = {B1}
         IF(N1 .GT. 0) THEN
            WRITE(LUPR,2004)
            IF     (MOD .EQ. 1) THEN
               CALL RSDS  (LU_F11, LU_B1, LU_X1, NRHS, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDS  (LU_F11, LU_B1, LU_X1, NRHS, 0)
            ELSE IF(MOD .EQ. 3) THEN
```

```
               CALL CHDS  (LU_F11, LU_B1, LU_X1, NRHS, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDS  (LU_F11, LU_B1, LU_X1, NRHS, 0)
            ELSE
               CALL CNDS  (LU_F11, LU_B1, LU_X1, NRHS, 0)
            END IF
         END IF
C
C (4.5) Multiply [F22] = [A22] - [A21]T[F12]
         IF(N22 .GT. 0) THEN
            IF( (N1  .GT. 0) .AND.
     1          (N2  .GT. 0) ) THEN
               WRITE(LUPR,2005)
               CALL FMSMM (LU_F22, LU_A22, -1, TRANSA, LU_A21, LU_F12)
            ELSE
               CALL FMSCPY (LU_A22, ' ', LU_F22)
            END IF
         END IF
C
C (4.6) Multiply [F22] = [F22] - [A21]T[F13]
         IF( (N1  .GT. 0) .AND.
     1       (N3  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2006)
            IF(ISTYPE .NE. 1) THEN
               CALL FMSIST ('MMCOL', N2)
               CALL FMSMM (LU_F22, LU_F22, -1, TRANSA, LU_A21, LU_F13)
               CALL FMSIST ('MMCOL',  0)
            END IF
         END IF
C
C (4.7) Multiply [F22] = [F22] - [A31]T[F12]
         IF( (N1  .GT. 0) .AND.
     1       (N2  .GT. 0) .AND.
     2       (N3  .GT. 0) .AND.
     3       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2007)
            CALL FMSIST ('MMROW', N2)
            CALL FMSMM (LU_F22, LU_F22, -1, TRANSA, LU_A31, LU_F12)
```

```
        END IF
C
C (4.8) Multiply [F22] = [F22] - [A31]T[F13]
        IF( (N1  .GT. 0) .AND.
     1      (N3  .GT. 0) .AND.
     2      (N22 .GT. 0) ) THEN
           WRITE(LUPR,2008)
           CALL FMSIST ('MMCOL', N2)
           CALL FMSMM (LU_F22, LU_F22, -1, 'T', LU_A31, LU_F13)
           CALL FMSIST ('MMCOL',  0)
           CALL FMSIST ('MMROW',  0)
        END IF
C
C (4.9) Multiply {X2}  = {B2}  - [A21]T{X1}
        IF(N22 .GT. 0) THEN
           IF( (N1  .GT. 0) .AND.
     1         (N2  .GT. 0) ) THEN
              WRITE(LUPR,2009)
              CALL FMSMM (LU_X2 , LU_B2 , -1, TRANSA, LU_A21, LU_X1 )
           ELSE
              CALL FMSCPY (LU_B2, ' ', LU_X2)
           END IF
        END IF
C
C (4.10)Multiply {X2}  = {X2}  - [A31]{X1}
        IF( (N1  .GT. 0) .AND.
     1      (N3  .GT. 0) .AND.
     2      (N22 .GT. 0) ) THEN
           WRITE(LUPR,2010)
           CALL FMSIST ('MMROW', N2)
           CALL FMSMM (LU_X2,  LU_X2,  -1, TRANSA, LU_A31, LU_X1 )
           CALL FMSIST ('MMROW',  0)
        END IF
C
C (4.11)Factor [A22] into [F22]
        IF(N22 .GT. 0) THEN
           WRITE(LUPR,2011)
           IF     (MOD .EQ. 1) THEN
              CALL RSDAF
```

```
     1        (LU_F22, R_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
           ELSE IF(MOD .EQ. 2) THEN
              CALL RNDAF
     1        (LU_F22, R_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
           ELSE IF(MOD .EQ. 3) THEN
              CALL CHDAF
     1        (LU_F22, R_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
           ELSE IF(MOD .EQ. 4) THEN
              CALL CSDAF
     1        (LU_F22, C_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
           ELSE IF(MOD .EQ. 5) THEN
              CALL CNDAF
     1        (LU_F22, C_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
           END IF
        END IF
C
C (4.12)Solve [F22]{X2} = {X2'}
        IF(N22 .GT. 0) THEN
           WRITE(LUPR,2012)
           IF     (MOD .EQ. 1) THEN
              CALL RSDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE IF(MOD .EQ. 2) THEN
              CALL RNDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE IF(MOD .EQ. 3) THEN
              CALL CHDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE IF(MOD .EQ. 4) THEN
              CALL CSDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE
              CALL CNDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           END IF
        END IF
C
C (4.13)Multiply {X1} = {X1} - [F13]{X2}
        IF( (N1  .GT. 0) .AND.
     1      (N3  .GT. 0) .AND.
     2      (N22 .GT. 0) ) THEN
           WRITE(LUPR,2013)
           CALL FMSIST ('MMKA', N2)
```

```
            CALL FMSMM (LU_X1, LU_X1, -1, 'N', LU_F13, LU_X2)
            CALL FMSIST ('MMKA',  0)
         END IF
C
C (4.14)Multiply {X1} = {X1} - [F12]{X2}
         IF( (N1  .GT. 0) .AND.
     1       (N2  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2014)
            CALL FMSMM (LU_X1, LU_X1, -1, 'N', LU_F12, LU_X2)
         END IF
C
C        Compute the error in {X1}:
C        {E1} = {B1} - [A11]{X1} - [A12|A13]{X2}:
C
C (4.15)Multiply {B1} = {B1} - [A11]{X1}
         IF(N1 .GT. 0) THEN
            WRITE(LUPR,2015)
            CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A11, LU_X1 )
         END IF
C
C (4.16)Multiply {B1} = {B1} - [A12]{X2}
         IF( (N1  .GT. 0) .AND.
     1       (N2  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2016)
            IF(ISTYPE .NE. 1) THEN
               CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A12, LU_X2 )
            ELSE
               CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A21, LU_X2 )
            END IF
         END IF
C
C (4.17)Multiply {B1} = {B1} - [A13]{X2}
         IF( (N1  .GT. 0) .AND.
     1       (N3  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2017)
            CALL FMSIST ('MMKA', N2)
```

```
            IF(ISTYPE .NE. 1) THEN
               CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A13, LU_X2 )
            ELSE
               CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A31, LU_X2 )
            END IF
            CALL FMSIST ('MMKA',  0)
         END IF
C
C        Compute the error in {X2}:
C        {E2} = {B2} - [A22]{X2} - [A21]{X1} - [A31]{X1}:
C
C (4.18)Multiply {B2} = {B2} - [A22]{X2}
         IF(N22 .GT. 0) THEN
            WRITE(LUPR,2018)
            CALL FMSMM (LU_B2, LU_B2, -1, 'N', LU_A22, LU_X2)
         END IF
C
C (4.19)Multiply {B2} = {B2} - [A21]T{X1}
         IF( (N1  .GT. 0) .AND.
     1       (N2  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2019)
            CALL FMSMM (LU_B2, LU_B2, -1, TRANSA, LU_A21, LU_X1)
         END IF
C
C (4.20)Multiply {B2} = {B2} - [A31]T{X1}
         IF( (N1  .GT. 0) .AND.
     1       (N3  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2020)
            CALL FMSIST ('MMROW', N2)
            CALL FMSMM (LU_B2, LU_B2, -1, TRANSA, LU_A31, LU_X1)
            CALL FMSIST ('MMROW',  0)
         END IF
C
C (5)    Read data from FMS files:
C
         IF(N1 .GT. 0) THEN
C           Check {X1}:
```

```
            ERROR  = R_ZERO
C
C           Initialize FMSGET:
            CALL FMSGET (LU_B1, 0, 0, 0, 0, RMD, 0)
            DO IRHS = 1,NRHS
               CALL FMSGET (LU_B1, N1, 1, 1, IRHS, RMD(L_TEMP), N1)
               L = L_TEMP
               DO I = 1,N1
                  IF(IDTYPE .EQ. 1) THEN
                     EI = ABS(RMD(L))
                     L  = L + 1
                  ELSE
                     EI = ABS(DCMPLX(RMD(L),RMD(L+1)))
                     L  = L + 2
                  END IF
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END DO
C
C           End FMSGET:
            CALL FMSGET (LU_B1, 0, 0, N1+1, 0, RMD, 0)
            WRITE(6,*) 'MAXIMUM ERROR IN {X1}     =', ERROR
         END IF
C
         IF(N22 .GT. 0) THEN
C           Check {X2}:
            ERROR  = R_ZERO
C
C           Initialize FMSGET:
            CALL FMSGET (LU_B2, 0, 0, 0, 0, RMD, 0)
            DO IRHS = 1,NRHS
               CALL FMSGET (LU_B2, N22, 1, 1, IRHS, RMD(L_TEMP), N22)
               L = L_TEMP
               DO I = 1,N22
                  IF(IDTYPE .EQ. 1) THEN
                     EI = ABS(RMD(L))
                     L  = L + 1
                  ELSE
                     EI = ABS(DCMPLX(RMD(L),RMD(L+1)))
```

```
                     L  = L + 2
                  END IF
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END DO
C
C           End FMSGET:
            CALL FMSGET (LU_B2, 0, 0, N22+1, 0, RMD, 0)
            WRITE(6,*) 'MAXIMUM ERROR IN {X2}     =', ERROR
         END IF
C
C (6)    Close FMS files:
         IF(N1 .GT. 0) THEN
            CALL FMSCM (LU_A11)
            CALL FMSCM (LU_F11)
            IF(ISTYPE .NE. 1) THEN
               IF(N2 .GT. 0) CALL FMSCV (LU_A12)
               IF(N3 .GT. 0) CALL FMSCV (LU_A13)
            END IF
            IF(N2 .GT. 0) THEN
               CALL FMSCV (LU_A21)
               CALL FMSCV (LU_F12)
            END IF
            IF(N3 .GT. 0) THEN
               CALL FMSCV (LU_A31)
               CALL FMSCV (LU_F13)
            END IF
            CALL FMSCV (LU_B1 )
            CALL FMSCV (LU_X1 )
         END IF
C
         IF(N22 .GT. 0) THEN
            CALL FMSCM (LU_A22)
            CALL FMSCM (LU_F22)
            CALL FMSCV (LU_B2 )
            CALL FMSCV (LU_X2 )
         END IF
         CALL FMSRMR (RMD, L_TEMP, N_TEMP)
         IF(ASK('Do you want another solution?')) GO TO 10
```

```
      CALL FMSPOP(MYNAME)
      CALL FMSEND
 2001 FORMAT (/
     1' ( 1) Factor   [A11] into [F11]'/
     2' ==============================')
 2002 FORMAT (/
     1' ( 2) Solve    [F11]{F12} = {A12}'/
     2' ==============================')
 2003 FORMAT (/
     1' ( 3) Solve    [F11]{F13} = {A13}'/
     2' ==============================')
 2004 FORMAT (/
     1' ( 4) Solve    [F11]{X1a} = {B1}'/
     2' ==============================')
 2005 FORMAT (/
     1' ( 5) Multiply [F22a] = [A22]  - [A21]T[F12]'/
     2' ==========================================')
 2006 FORMAT (/
     1' ( 6) Multiply [F22b] = [F22a] - [A21]T[F13]'/
     2' ==========================================')
 2007 FORMAT (/
     1' ( 7) Multiply [F22c] = [F22b] - [A31]T[F12]'/
     2' ==========================================')
 2008 FORMAT (/
     1' ( 8) Multiply [F22d] = [F22d] - [A31]T[F13]'/
     2' ==========================================')
 2009 FORMAT (/
     1' ( 9) Multiply {X2a}  = {B2}   - [A21]T{X1a}'/
     2' ==========================================')
 2010 FORMAT (/
     1' (10) Multiply {X2b}  = {X2a}  - [A31]T{X1a}'/
     2' ==========================================')
 2011 FORMAT (/
     1' (11) Factor   [F22d] into [F22]'/
     2' ==============================')
 2012 FORMAT (/
     1' (12) Solve    [F22]{X2} = {X2b}'/
     2' ==============================')
 2013 FORMAT (/
```

```
     1  ' (13) Multiply {X1b}  = {X1a}  - [F13]{X2}'/
     2  ' ========================================')
 2014   FORMAT (/
     1  ' (14) Multiply {X1}   = {X1b}  - [F12]{X2}'/
     2  ' ========================================')
 2015   FORMAT (/
     1  ' (15) Multiply {E1a}  = {B1}   - [A11]{X1}'/
     2  ' ========================================')
 2016   FORMAT (/
     1  ' (16) Multiply {E1b}  = {E1a}  - [A12]{X2}'/
     2  ' ========================================')
 2017   FORMAT (/
     1  ' (17) Multiply {E1}   = {E1b}  - [A13]{X2}'/
     2  ' ========================================')
 2018   FORMAT (/
     1  ' (18) Multiply {E2a}  = {B2}   - [A22]{X2}'/
     2  ' ========================================')
 2019   FORMAT (/
     1  ' (19) Multiply {E2b}  = {E2a}  - [A21]{X1}'/
     2  ' ========================================')
 2020   FORMAT (/
     1  ' (20) Multiply {E2}   = {E2b}  - [A31]{X1}'/
     2  ' ========================================')
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
```

```
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE AGEN (A, IDTYPE, IROW1, IROW2, JCOL1, JCOL2,
     1                   NUMEQ, MTEST, LUA, ISTYPE)
C=======================================================================
        REAL*8     A(IDTYPE,NUMEQ)
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, NUMEQ
        INTEGER    LUA(25)
        REAL*8     A_1, A_DIAG
        REAL*8     R_ZERO, R_ONE
        DATA       R_ZERO/0.0D0/
        DATA       R_ONE /1.0D0/
C
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2
        NROWS = IROW2 - IROW1 + 1
        NCOLS = JCOL2 - JCOL1 + 1
        IEDGE = 0
        IF(IROW1 .GT. JCOL1) THEN
C          This is a off-diagonal block in the lower triangle:
C          Write it out transposed:
           LDU  = 1
           I1   = JCOL1
           I2   = JCOL2
           J1   = IROW1
           J2   = IROW2
           IF(JCOL1 .EQ. 1) IEDGE = 1
        ELSE IF(IROW1 .EQ. JCOL1) THEN
C          This block is on the diagonal:
C          Write it out by columns:
```

```
            LDU  = 2
            I1   = IROW1
            I2   = IROW2
            J1   = JCOL1
            J2   = JCOL2
            IF(IROW1 .EQ. 1) IEDGE = 1
         ELSE
C           This block is in the upper triangle:
C           Write it out by columns:
            LDU  = 3
            I1   = IROW1
            I2   = IROW2
            J1   = JCOL1
            J2   = JCOL2
            IF(IROW1 .EQ. 1) IEDGE = 1
         END IF
         LENV = I2 - I1 + 1
C
         IF(MTEST .EQ. 1) THEN
C           Fixed data:
C           Initialize A to default values (generic column):
            DO I = 1,IDTYPE
               DO J=1,LENV
                  A(I,J) = R_ZERO
               END DO
            END DO
            A_DIAG = R_ONE
            IF(IEDGE .EQ. 1) THEN
               A_1 = -R_ONE
            ELSE
               A_1 = R_ZERO
            END IF
            A(1,1) = A_1
         END IF
C
C        Initialize FMSPUT:
         CALL FMSPUT (LUA, 0, 0, 0, 0, A, 0)
C
C        Loop over the columns:
```

```
         DO J = J1,J2
            JC1 = J - J1 + 1
            IF(MTEST .EQ. 1) THEN
C              Use fixed data:
               IF(LDU .EQ. 2) THEN
C                 Diagonal block:
                  IF(J .EQ. 1) THEN
C                    Special case.  This is the first global column:
                     DO I=1,NROWS
                        A(1,I) = -R_ONE
                     END DO
                     A_DIAG = DFLOAT(NUMEQ)
                  END IF
               A(1,JC1) = A_DIAG
               END IF
            ELSE
C              Use random data:
               IF(LDU .EQ. 1) THEN
C                 Lower triangle off-diagonal block; Generate transposed
                  CALL VRAN (A, IDTYPE, J, J, JCOL1, JCOL2,
     1                       NUMEQ, ISTYPE)
               ELSE
C                 Diagonal or upper triangle block:
                  CALL VRAN (A, IDTYPE, IROW1, IROW2, J, J,
     1                       NUMEQ, ISTYPE)
               END IF
            END IF
D5001    FORMAT (' CALL FMSPUT(LUA,',I5,', 1, 1,',I5,', A,',I5,')')
            CALL FMSPUT (LUA, LENV, 1, 1, JC1, A, LENV)
            IF(MTEST .EQ. 1) THEN
C              Restore A to default:
               IF(LDU .EQ. 2) THEN
C                 Diagonal block:
                  IF(J .EQ. 1) THEN
C                    Special case.  This is the first global column:
                     DO I=1,NROWS
                        A(1,I) = R_ZERO
                     END DO
                     A_DIAG = R_ONE
```

```
                  END IF
                  A(1,JC1) = R_ZERO
                  A(1,1) = A_1
               END IF
            END IF
         END DO
C
C        End FMSPUT:
         NEND = LENV+1
         CALL FMSPUT (LUA, 0, 0, NEND, 0, A, 0)
         RETURN
 2000    FORMAT (' Populating A(',I5,':',I5,',',I5,':',I5,')')
         END
C=======================================================================
         SUBROUTINE VRAN (A, IDTYPE, IROW1, IROW2, JCOL1, JCOL2, NUMEQ,
     1                    ISTYPE)
C=======================================================================
         REAL*8    A(IDTYPE, IROW1:IROW2,JCOL1:JCOL2)
         INTEGER   IROW1, IROW2, JCOL1, JCOL2, NUMEQ, ISTYPE
         REAL*8    RANAIJ, SUM
         INTEGER   ISEEDR, ISEEDC
         PARAMETER (ISEEDR=11111111)
         PARAMETER (ISEEDC=12345678)
         DO JCOL = JCOL1,JCOL2
            DO IROW = IROW1,IROW2
               IF(JCOL .LT. IROW) THEN
C                 This term is in the lower triangle:
                  IY     = IROW*ISEEDR + JCOL*ISEEDC
                  IY     = MOD(IY,65536)
                  IY     = IY * 11111 + 1
                  RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
                  RANAIJ = RANAIJ - 0.5D0
               ELSE IF(JCOL .EQ. IROW) THEN
C                 This term is on the diagonal:
C                 Set the value to (-) the sum of the off-diagonal terms:
                  SUM = 0.0D0
                  DO II = 1,NUMEQ
                     IF(II .NE. IROW) THEN
                        IF(II .LT. IROW) THEN
```

```
C                        This term is in the lower triangle:
                         IY     = IROW*ISEEDR + II*ISEEDC
                      ELSE
C                        This term is in the upper triangle:
                         IF(ISTYPE .EQ. 2) THEN
C                           This matrix is nonsymmetric:
                            IY     = IROW*ISEEDR + II*ISEEDC
                         ELSE
C                           This matrix is symmetric or Hermitian:
                            IY     =  II*ISEEDR + IROW*ISEEDC
                         END IF
                      END IF
                      IY     = MOD(IY,65536)
                      IY     = IY * 11111 + 1
                      RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
                      RANAIJ = RANAIJ - 0.5D0
                      SUM = SUM + RANAIJ
                   END IF
                END DO
                RANAIJ = -SUM
             ELSE
C               This term is in the upper triangle:
                IF(ISTYPE .EQ. 2) THEN
C                  This matrix is nonsymmetric:
                   IY     = IROW*ISEEDR + JCOL*ISEEDC
                ELSE
C                  This matrix is symmetric or Hermitian:
                   IY     = JCOL*ISEEDR + IROW*ISEEDC
                END IF
                IY     = MOD(IY,65536)
                IY     = IY * 11111 + 1
                RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
                RANAIJ = RANAIJ - 0.5D0
             END IF
             A(1,IROW,JCOL) = RANAIJ
             IF((IROW  .EQ. 1) .AND.
     1          (JCOL  .EQ. 1)) A(1,1,1) = RANAIJ + 1.0D0
             IF(IDTYPE .EQ. 2) THEN
                IF(ISTYPE .EQ. 3) THEN
```

```
C                  This matrix is Hermitian:
                   IF(JCOL .LT. IROW) THEN
                      A(2,IROW,JCOL) = RANAIJ
                   ELSE IF(JCOL .EQ. IROW) THEN
                      A(1,IROW,JCOL) = 2*RANAIJ
                      A(2,IROW,JCOL) = 0.0D0
                   ELSE
                      A(2,IROW,JCOL) = -RANAIJ
                   END IF
                ELSE
C                  This matrix is Complex Symmetric or Nonsymmetric:
                   A(2,IROW,JCOL) = RANAIJ
                END IF
             END IF
          END DO
       END DO
       RETURN
       END
C=======================================================================
       SUBROUTINE BGEN (RHS, IDTYPE, IROW1, IROW2, NRHS, LUB)
C=======================================================================
       REAL*8     RHS(IDTYPE, *)
       INTEGER    IDTYPE, IROW1, IROW2, NRHS
       INTEGER    LUB(25)
       REAL*8     R_ZERO, R_ONE
       DATA       R_ZERO/0.0D0/
       DATA       R_ONE /1.0D0/
C
       WRITE(6,2000) IROW1, IROW2, NRHS
C
       NROWS  = IROW2 - IROW1 + 1
       DO I=1,IDTYPE
          DO J=1,NROWS
            RHS(I,J) = R_ZERO
          END DO
       END DO
       IF(IROW1 .EQ. 1) RHS(1,1) = R_ONE
C
```

```
C       Initialize FMSPUT:
        CALL FMSPUT (LUB, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B}:
        DO IRHS = 1,NRHS
           CALL FMSPUT (LUB, NROWS, 1, 1, IRHS, RHS, NROWS)
        END DO
C
C       End FMSPUT:
        CALL FMSPUT (LUB, 0, 0, NROWS+1, 0, RHS, 0)
        RETURN
 2000   FORMAT (' Populating B(',I5,':',I5,',',I5,')')
        END
```

Copyright © Multipath Corporation

```
C       The FMS Out-of-Core matrix multiply routine FMSMM forms the
C       product of two matrixes [A][B] and stores the result in matrix
C       [C].  Under some matrix partitioning conditions, the size of
C       the product [AB] = [A][B] will be different than the matrix [C].
C       Normally, FMS will store the product [AB] starting at the upper
C       left corner of [C], C(1,1).  However, it may be necessary to
C       shift the starting position of the product [AB] in [C] to some
C       other position.  The FMS Parameters MMROW and MMCOL specify
C       the number of rows and columns to shift the product [AB] in
C       [C], storing [AB] in [C] starting at C(1+MMROW,1+MMCOL).  The
C       default values of MMROW and MMCOL are 0.
C
C       Another shifting requirement occurs when the matrix [A] has
C       fewer columns than the matrix [B] has rows.  In this case, FMS
C       provides the shifting parameter MMKA, which has the effect of
C       adding MMKA columns of 0's to matrix [A].  The matrix product
C       then becomes
C
C       [AB] = [0|A][B]
C
C       The following example illustrates the use of FMSMM and these
C       FMS Parameters.
C
C       The system of linear equations is partitioned as follows:
C
C          --   +-    +     +    -+ +-  -+   +-  -+
C          N1   | A11 | A12 | A13 | | X1 |   | B1 |
C          --   +-----+-----+-----+ +----+   +----+
C          N2   | A21 |           | |    | = |    |
C          --   +-----+    A22    + + X2 +   + B2 +
C          N3   | A31 |           | |    |   |    |
C          --   +-    +-         -+ +-  -+   +-  -+
C
C               |< N1>|< N2>|< N3>|
C                     |<---N22--->|
C
C       Because the matrices [A31] and [A13] do not lie on a boundary
C       of [A22], it is necessary to use the matrix multiply shifting
C       parameters during the partitioned solution of this system.
C       The following table lists the products computed by FMSMM and
C       the required values of the shifting parameters.
C
C       Matrix Product   Stored In   MMROW   MMCOL   MMKA
C       ==============   =========   =====   =====   ====
C       [A21][A12]       [A22]           0       0      0
C       [A21][A13]       [A22]           0      N2      0
C       [A21][X1]        [X2]            0       0      0
C       [A31][A12]       [A22]          N2       0      0
C       [A31][A13]       [A22]          N2      N2      0
C       [A31][X1]        [X2]           N2       0      0
C       [0|A13][X2]      [X1]            0       0     N2
C       [A12][X2]        [X1]            0       0      0
C
C       Normally FMS will overlay matrix factors and solution vectors
C       on the input matrix data.  However, for this example, we want
C       to save the original matrices to check the result.  Therefore
C       separate files will be used for the original matrix, factors
C       and the solution vectors.
C
C       The following FMS files will be used:
C
C       File     File
C       Name     Type     CONTENTS
```

```
C       ======   ======   ========
C       LU_A11   Matrix   [A11]
C       LU_F11   Matrix   [F11], factors of [A11]
C
C       LU_A12   Vector   [A12]
C       LU_F12   Vector   [F12], solution of [F11][F12]=[A12]
C
C       LU_A13   Vector   [A13]
C       LU_F13   Vector   [F13], solution of [F11][F13]=[A13]
C
C       LU_A21   Vector   [A21]T
C
C       LU_A31   Vector   [A31]T
C
C       LU_A22   Matrix   [A22]
C       LU_F22   Matrix   [F22], factors of [A22]-[A21|A31][F12|F13]
C
C       LU_B1    Vector   {B1}
C       LU_X1    Vector   {X1}
C
C       LU_B2    Vector   {B2}
C       LU_X2    Vector   {X2}
C
C       For symmetric problems, the files LU_A12 and LU_A13 are not
C       required, because they are the same as LU_A21 and LU_A31.
C
C       INPUT:
C       ======
C       The following input is used:
C          FMS Module (1=RS,2=RN,3=CH,4=CS,5=CN)
C          Number of equations in [A11], N1
C          Number of equations in [A21], N2
C          Number of equations in [A31], N2
C          Number of solution vectors, NRHS
C          Matrix test data (1=known, 2=random), MTEST
C          Any FMS Parameter
C
C
C       For known matrix test data (MTEST=1), the following pattern
C       is used:
C
C              [A]        {X}  =  {B}
C       +-            -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-            -+ +   +   +   +
C
C       where N=N1+N2+N3
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_15')
C
C       FMS matrix and vector file attributes:
        INTEGER     LU_A11(25), LU_F11(25)
        INTEGER     LU_A12(25), LU_F12(25)
        INTEGER     LU_A13(25), LU_F13(25)
        INTEGER     LU_A21(25)
        INTEGER     LU_A31(25)
        INTEGER     LU_A22(25), LU_F22(25)
        INTEGER     LU_B1 (25), LU_X1 (25)
        INTEGER     LU_B2 (25), LU_X2 (25)
```

```
      INTEGER     LUA0  (25)
      DATA        LUA0(1)/0/
      INTEGER     LUX0  (25)
      INTEGER     LUS0  (25)
C
C     FMS memory management requires the following arrays:
      POINTER   (CMD_PTR, CMD)
      POINTER   (RMD_PTR, RMD)
      POINTER   (IMD_PTR, IMD)
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C
C     Input Data:
      LOGICAL ASK
      INTEGER ASK_I
C
C     LOCAL VARIABLES:
C     FMS module number (1 to 5)
         INTEGER MOD
C     Number of equations in [A11]:
         INTEGER N1
C     Number of equations in [A21]:
         INTEGER N2
C     Number of equations in [A31]:
         INTEGER N3
C     Number of equations in [A22]: (=N2+N3)
C        INTEGER N22
C     Number of solution vectors:
         INTEGER NRHS
C     FMS Data Type (1 or 2)
         INTEGER IDTYPE
C     FMS Matrix Symmetry (1, 2 or 3):
         INTEGER ISTYPE
C     Test matrix data type (1=fixed, 2=random):
         INTEGER MTEST
C     Error values:
         REAL*8  EI
         REAL*8  ERROR
C     Size of scratch work vector:
         INTEGER N_TEMP
C     RMD Pointer to scratch vector:
         INTEGER L_TEMP
C     Transpose of matrix A:
         CHARACTER*1 TRANSA
C
C     CONSTANTS:
         REAL*8     R_ZERO, R_ONE(1), R_BIG
         DATA R_ZERO/0.0D0/
         DATA R_ONE /1.0D0/
         DATA R_BIG /1.0D30/
         COMPLEX*16 C_ONE(1)
         DATA C_ONE /(1.0D0,0.0D0)/
C
C        Profile vector for a full matrix:
         INTEGER LOWEQ(1)
         DATA    LOWEQ/-1/
C
C (1)  Initialize FMS:
      CALL FMSINI
      CALL FMSPSH (MYNAME)
      CALL FMSIGT ('LUPR'  , LUPR)
C
C     Use block format matrix:
      CALL FMSIST ('MFMAT', 2)
```

```
C
C       Read in problem size parameters:
   10   CONTINUE
        WRITE (6,*) 'The FMS modules are numbered as follows:'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        MOD  = ASK_I('Enter the FMS module number (1 to 5)')
        IF( (MOD .LT. 1) .OR. (MOD. GT. 5) ) THEN
           PRINT *,'Illegal FMS Module entered.  Try again.'
           GO TO 10
        END IF
        N1   = ASK_I('Enter the number of equations in A11, N1')
        N2   = ASK_I('Enter the number of columns   in A12, N2')
        N3   = ASK_I('Enter the number of columns   in A13, N3')
        NRHS = ASK_I('Enter the number of solution vectors, NRHS')
        MTEST= ASK_I('Enter the test data (1=fixed,2=random)')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        N22   = N2 + N3
        NUMEQ = N1 + N22
        IF(MOD .LE. 2) THEN
C          Real*8 data:
           IDTYPE = 1
        ELSE
C          Complex*16 data:
           IDTYPE = 2
        END IF
        LENDIA = IDTYPE*NUMEQ
        TRANSA = 'T'
        IF     ((MOD .EQ. 1) .OR. (MOD .EQ. 4)) THEN
C          Symmetric matrix:
           ISTYPE = 1
        ELSE IF((MOD .EQ. 2) .OR. (MOD .EQ. 5)) THEN
C          Nonsymmetric matrix:
           ISTYPE = 2
        ELSE
C          Hermitian matrix:
           ISTYPE = 3
        END IF
C
C       Get a temporary vector for generating columns and RHS's:
        N_TEMP = IDTYPE*NUMEQ
        CALL FMSRMG (RMD, L_TEMP, N_TEMP)
C
C (2)   Open FMS files:
C
C       Matrix files:
C
C       [A11]:
        IF(N1 .GT. 0) THEN
           IF     (MOD.EQ.1) THEN
              CALL RSDI(LOWEQ, N1 , 'LU_A11', LU_A11)
           ELSE IF(MOD.EQ.2) THEN
              CALL RNDI(LOWEQ, N1 , 'LU_A11', LU_A11)
           ELSE IF(MOD.EQ.3) THEN
              CALL CHDI(LOWEQ, N1 , 'LU_A11', LU_A11)
           ELSE IF(MOD.EQ.4) THEN
```

```
               CALL CSDI(LOWEQ, N1 , 'LU_A11', LU_A11)
            ELSE
               CALL CNDI(LOWEQ, N1 , 'LU_A11', LU_A11)
            END IF
            CALL FMSOM (LU_A11, 'LU_F11', LU_F11)
         END IF
C
C        [A22]:
         IF(N22 .GT. 0) THEN
            IF     (MOD.EQ.1) THEN
               CALL RSDI(LOWEQ, N22, 'LU_A22', LU_A22)
            ELSE IF(MOD.EQ.2) THEN
               CALL RNDI(LOWEQ, N22, 'LU_A22', LU_A22)
            ELSE IF(MOD.EQ.3) THEN
               CALL CHDI(LOWEQ, N22, 'LU_A22', LU_A22)
            ELSE IF(MOD.EQ.4) THEN
               CALL CSDI(LOWEQ, N22, 'LU_A22', LU_A22)
            ELSE
               CALL CNDI(LOWEQ, N22, 'LU_A22', LU_A22)
            END IF
            CALL FMSOM (LU_A22, 'LU_F22', LU_F22)
         END IF
C
C        Vector and off-diagonal matrix files:
         IF(N1 .GT. 0) THEN
            IF(ISTYPE .NE. 1) THEN
               IF(N2 .GT. 0) THEN
                  CALL FMSOV (N1, IDTYPE, N2, 'LU_A12', LU_A12)
               END IF
               IF(N3 .GT. 0) THEN
                  CALL FMSOV (N1, IDTYPE, N3, 'LU_A13', LU_A13)
               END IF
            END IF
            IF(N2 .GT. 0) THEN
               CALL FMSOV (N1, IDTYPE, N2, 'LU_A21', LU_A21)
               CALL FMSOV (N1, IDTYPE, N2, 'LU_F12', LU_F12)
            END IF
            IF(N3 .GT. 0) THEN
               CALL FMSOV (N1, IDTYPE, N3, 'LU_A31', LU_A31)
               CALL FMSOV (N1, IDTYPE, N3, 'LU_F13', LU_F13)
            END IF
            CALL FMSOV (N1 , IDTYPE, NRHS, 'LU_B1' , LU_B1 )
            CALL FMSOV (N1 , IDTYPE, NRHS, 'LU_X1' , LU_X1 )
         END IF
         IF(N22 .GT. 0) THEN
            CALL FMSOV (N22, IDTYPE, NRHS, 'LU_B2' , LU_B2)
            CALL FMSOV (N22, IDTYPE, NRHS, 'LU_X2' , LU_X2)
         END IF
C
C (3)    Write data to FMS files:
C
C        Generate [A11]:
         IF(N1 .GT. 0) THEN
            CALL AGEN (RMD(L_TEMP), IDTYPE, 1, N1, 1, N1, NUMEQ, MTEST,
     1                 LU_A11, ISTYPE)
D           CALL FMSPRM (LU_A11, 1)
C
            IF(ISTYPE .NE. 1) THEN
C
C              Generate [A12]:
               IF(N2 .GT. 0) THEN
                  CALL AGEN (RMD(L_TEMP), IDTYPE, 1, N1, N1+1, N1+N2,
     1                       NUMEQ, MTEST, LU_A12, ISTYPE)
D                 CALL FMSPRV (LU_A12, 1, N2, 10)
               END IF
```

```
C
C             Generate [A13]:
              IF(N3 .GT. 0) THEN
              CALL AGEN (RMD(L_TEMP), IDTYPE, 1, N1, N1+N2+1, NUMEQ,
     1                   NUMEQ, MTEST, LU_A13, ISTYPE)
D             CALL FMSPRV (LU_A13, 1, N3, 10)
              END IF
            END IF
C
C           Generate [A21]:
            IF(N2 .GT. 0) THEN
               CALL AGEN (RMD(L_TEMP), IDTYPE, N1+1, N1+N2, 1, N1,
     1                   NUMEQ, MTEST, LU_A21, ISTYPE)
D              CALL FMSPRV (LU_A21, 1, N2, 10)
            END IF
C
C           Generate [A31]:
            IF(N3 .GT. 0) THEN
               CALL AGEN (RMD(L_TEMP), IDTYPE, N1+N2+1, NUMEQ, 1, N1,
     1                   NUMEQ, MTEST, LU_A31, ISTYPE)
D              CALL FMSPRV (LU_A31, 1, N3, 10)
            END IF
C
C           Generate {B1}:
            CALL BGEN  (RMD(L_TEMP), IDTYPE, 1, N1, NRHS, LU_B1)
D           CALL FMSPRV (LU_B1, 1, NRHS, 10)
         END IF
C
C        Generate [A22]:
         IF(N22 .GT. 0) THEN
            CALL AGEN (RMD(L_TEMP), IDTYPE, N1+1, NUMEQ, N1+1, NUMEQ,
     1                NUMEQ, MTEST, LU_A22, ISTYPE)
D           CALL FMSPRM(LU_A22, 1)
C
C           Generate {B2}:
            CALL BGEN  (RMD(L_TEMP), IDTYPE, N1+1, NUMEQ, NRHS, LU_B2)
D           CALL FMSPRV (LU_B2, 1, NRHS, 10)
         END IF
C
C (4)    Perform matrix algebra:
C
C (4.1)  Factor [A11] into [F11]:
         IF(N1 .GT. 0) THEN
            WRITE(LUPR,2001)
            IF     (MOD .EQ. 1) THEN
               CALL RSDAF
     1         (LU_A11, R_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDAF
     1         (LU_A11, R_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
            ELSE IF(MOD .EQ. 3) THEN
               CALL CHDAF
     1         (LU_A11, R_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDAF
     1         (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
            ELSE
               CALL CNDAF
     1         (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11, LU_B1, LU_X1, 0)
            END IF
         END IF
C
C (4.2)  Solve [F11]{F12} = {A12}
         IF( (N1 .GT. 0) .AND.
     1       (N2 .GT. 0) ) THEN
```

```
            WRITE(LUPR,2002)
            IF     (MOD .EQ. 1) THEN
               CALL RSDS  (LU_F11, LU_A21, LU_F12, N2, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDS  (LU_F11, LU_A12, LU_F12, N2, 0)
            ELSE IF(MOD .EQ. 3) THEN
               CALL CHDS  (LU_F11, LU_A12, LU_F12, N2, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDS  (LU_F11, LU_A21, LU_F12, N2, 0)
            ELSE
               CALL CNDS  (LU_F11, LU_A12, LU_F12, N2, 0)
            END IF
         END IF
C
C (4.3) Solve [F11]{F13} = {A13}
         IF( (N1 .GT. 0) .AND.
     1       (N3 .GT. 0) ) THEN
            WRITE(LUPR,2003)
            IF     (MOD .EQ. 1) THEN
               CALL RSDS  (LU_F11, LU_A31, LU_F13, N3, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDS  (LU_F11, LU_A13, LU_F13, N3, 0)
            ELSE IF(MOD .EQ. 3) THEN
               CALL CHDS  (LU_F11, LU_A13, LU_F13, N3, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDS  (LU_F11, LU_A31, LU_F13, N3, 0)
            ELSE
               CALL CNDS  (LU_F11, LU_A13, LU_F13, N3, 0)
            END IF
         END IF
C
C (4.4) Solve [F11]{X1} = {B1}
         IF(N1 .GT. 0) THEN
            WRITE(LUPR,2004)
            IF     (MOD .EQ. 1) THEN
               CALL RSDS  (LU_F11, LU_B1,  LU_X1,  NRHS, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDS  (LU_F11, LU_B1,  LU_X1,  NRHS, 0)
            ELSE IF(MOD .EQ. 3) THEN
               CALL CHDS  (LU_F11, LU_B1,  LU_X1,  NRHS, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDS  (LU_F11, LU_B1,  LU_X1,  NRHS, 0)
            ELSE
               CALL CNDS  (LU_F11, LU_B1,  LU_X1,  NRHS, 0)
            END IF
         END IF
C
C (4.5) Multiply [F22] = [A22] - [A21]T[F12]
         IF(N22 .GT. 0) THEN
            IF( (N1  .GT. 0) .AND.
     1          (N2  .GT. 0) ) THEN
               WRITE(LUPR,2005)
               CALL FMSMM (LU_F22, LU_A22, -1, TRANSA, LU_A21, LU_F12)
            ELSE
               CALL FMSCPY (LU_A22, ' ', LU_F22)
            END IF
         END IF
C
C (4.6) Multiply [F22] = [F22] - [A21]T[F13]
         IF( (N1  .GT. 0) .AND.
     1       (N3  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2006)
            IF(ISTYPE .NE. 1) THEN
```

```
               CALL FMSIST('MMCOL', N2)
               CALL FMSMM (LU_F22, LU_F22, -1, TRANSA, LU_A21, LU_F13)
               CALL FMSIST('MMCOL',  0)
            END IF
         END IF
C
C (4.7) Multiply [F22] = [F22] - [A31]T[F12]
         IF( (N1  .GT. 0) .AND.
     1       (N2  .GT. 0) .AND.
     2       (N3  .GT. 0) .AND.
     3       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2007)
            CALL FMSIST('MMROW', N2)
            CALL FMSMM (LU_F22, LU_F22, -1, TRANSA, LU_A31, LU_F12)
         END IF
C
C (4.8) Multiply [F22] = [F22] - [A31]T[F13]
         IF( (N1  .GT. 0) .AND.
     1       (N3  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2008)
            CALL FMSIST('MMCOL', N2)
            CALL FMSMM (LU_F22, LU_F22, -1, TRANSA, LU_A31, LU_F13)
            CALL FMSIST('MMCOL',  0)
            CALL FMSIST('MMROW',  0)
         END IF
C
C (4.9) Multiply {X2}  = {B2}  - [A21]T{X1}
         IF(N22 .GT. 0) THEN
            IF( (N1  .GT. 0) .AND.
     1          (N2  .GT. 0) ) THEN
               WRITE(LUPR,2009)
               CALL FMSMM (LU_X2 , LU_B2 , -1, TRANSA, LU_A21, LU_X1 )
            ELSE
               CALL FMSCPY (LU_B2, ' ', LU_X2)
            END IF
         END IF
C
C (4.10)Multiply {X2}  = {X2}  - [A31]{X1}
         IF( (N1  .GT. 0) .AND.
     1       (N3  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2010)
            CALL FMSIST('MMROW', N2)
            CALL FMSMM (LU_X2,  LU_X2,  -1, TRANSA, LU_A31, LU_X1 )
            CALL FMSIST('MMROW',  0)
         END IF
C
C (4.11)Factor [A22] into [F22]
         IF(N22 .GT. 0) THEN
            WRITE(LUPR,2011)
            IF     (MOD .EQ. 1) THEN
               CALL RSDAF
     1         (LU_F22, R_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
            ELSE IF(MOD .EQ. 2) THEN
               CALL RNDAF
     1         (LU_F22, R_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
            ELSE IF(MOD .EQ. 3) THEN
               CALL CHDAF
     1         (LU_F22, R_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
            ELSE IF(MOD .EQ. 4) THEN
               CALL CSDAF
     1         (LU_F22, C_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
            ELSE IF(MOD .EQ. 5) THEN
               CALL CNDAF
```

```
     1        (LU_F22, C_ONE, 1, LUS0, 0, LUA0, LU_F22, LU_X2, LU_X2, 0)
           END IF
        END IF
C
C (4.12)Solve [F22]{X2} = {X2'}
        IF(N22 .GT. 0) THEN
           WRITE(LUPR,2012)
           IF     (MOD .EQ. 1) THEN
              CALL RSDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE IF(MOD .EQ. 2) THEN
              CALL RNDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE IF(MOD .EQ. 3) THEN
              CALL CHDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE IF(MOD .EQ. 4) THEN
              CALL CSDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           ELSE
              CALL CNDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
           END IF
        END IF
C
C (4.13)Multiply {X1} = {X1} - [F13]{X2}
        IF( (N1  .GT. 0) .AND.
     1      (N3  .GT. 0) .AND.
     2      (N22 .GT. 0) ) THEN
           WRITE(LUPR,2013)
           CALL FMSIST('MMKA', N2)
           CALL FMSMM (LU_X1, LU_X1, -1, 'N', LU_F13, LU_X2)
           CALL FMSIST('MMKA',  0)
        END IF
C
C (4.14)Multiply {X1} = {X1} - [F12]{X2}
        IF( (N1  .GT. 0) .AND.
     1      (N2  .GT. 0) .AND.
     2      (N22 .GT. 0) ) THEN
           WRITE(LUPR,2014)
           CALL FMSMM (LU_X1, LU_X1, -1, 'N', LU_F12, LU_X2)
        END IF
C
C       Compute the error in {X1}:
C       {E1} = {B1} - [A11]{X1} - [A12|A13]{X2}:
C
C (4.15)Multiply {B1} = {B1} - [A11]{X1}
        IF(N1 .GT. 0) THEN
           WRITE(LUPR,2015)
           CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A11, LU_X1 )
        END IF
C
C (4.16)Multiply {B1} = {B1} - [A12]{X2}
        IF( (N1  .GT. 0) .AND.
     1      (N2  .GT. 0) .AND.
     2      (N22 .GT. 0) ) THEN
           WRITE(LUPR,2016)
           IF(ISTYPE .NE. 1) THEN
              CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A12, LU_X2 )
           ELSE
              CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A21, LU_X2 )
           END IF
        END IF
C
C (4.17)Multiply {B1} = {B1} - [A13]{X2}
        IF( (N1  .GT. 0) .AND.
     1      (N3  .GT. 0) .AND.
     2      (N22 .GT. 0) ) THEN
           WRITE(LUPR,2017)
           CALL FMSIST ('MMKA', N2)
```

```
            IF(ISTYPE .NE. 1) THEN
               CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A13, LU_X2 )
            ELSE
               CALL FMSMM (LU_B1, LU_B1, -1, 'N', LU_A31, LU_X2 )
            END IF
            CALL FMSIST ('MMKA',  0)
         END IF
C
C        Compute the error in {X2}:
C        {E2} = {B2} - [A22]{X2} - [A21]{X1} - [A31]{X1}:
C
C (4.18)Multiply {B2} = {B2} - [A22]{X2}
         IF(N22 .GT. 0) THEN
            WRITE(LUPR,2018)
            CALL FMSMM (LU_B2, LU_B2, -1, 'N', LU_A22, LU_X2)
         END IF
C
C (4.19)Multiply {B2} = {B2} - [A21]T{X1}
         IF( (N1  .GT. 0) .AND.
     1       (N2  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2019)
            CALL FMSMM (LU_B2, LU_B2, -1, TRANSA, LU_A21, LU_X1)
         END IF
C
C (4.20)Multiply {B2} = {B2} - [A31]T{X1}
         IF( (N1  .GT. 0) .AND.
     1       (N3  .GT. 0) .AND.
     2       (N22 .GT. 0) ) THEN
            WRITE(LUPR,2020)
            CALL FMSIST('MMROW', N2)
            CALL FMSMM (LU_B2, LU_B2, -1, TRANSA, LU_A31, LU_X1)
            CALL FMSIST('MMROW',  0)
         END IF
C
C (5)   Read data from FMS files:
C
         IF(N1 .GT. 0) THEN
C           Check {X1}:
            ERROR  = R_ZERO
C
C           Initialize FMSGET:
            CALL FMSGET (LU_B1, 0, 0, 0, 0, RMD, 0)
            DO IRHS = 1,NRHS
               CALL FMSGET (LU_B1, N1, 1, 1, IRHS, RMD(L_TEMP), N1)
               L = L_TEMP
               DO I = 1,N1
                  IF(IDTYPE .EQ. 1) THEN
                     EI = ABS(RMD(L))
                     L  = L + 1
                  ELSE
                     EI = ABS(DCMPLX(RMD(L),RMD(L+1)))
                     L  = L + 2
                  END IF
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END DO
C
C           End FMSGET:
            CALL FMSGET (LU_B1, 0, 0, N1+1, 0, RMD, 0)
            WRITE(6,*) 'MAXIMUM ERROR IN {X1}     =', ERROR
         END IF
C
         IF(N22 .GT. 0) THEN
```

```
C          Check {X2}:
           ERROR  = R_ZERO
C
C          Initialize FMSGET:
           CALL FMSGET (LU_B2, 0, 0, 0, 0, RMD, 0)
           DO IRHS = 1,NRHS
              CALL FMSGET (LU_B2, N22, 1, 1, IRHS, RMD(L_TEMP), N22)
              L = L_TEMP
              DO I = 1,N22
                 IF(IDTYPE .EQ. 1) THEN
                    EI = ABS(RMD(L))
                    L  = L + 1
                 ELSE
                    EI = ABS(DCMPLX(RMD(L),RMD(L+1)))
                    L  = L + 2
                 END IF
                 IF(EI .GT. ERROR) ERROR = EI
              END DO
           END DO
C
C          End FMSGET:
           CALL FMSGET (LU_B2, 0, 0, N22+1, 0, RMD, 0)
           WRITE(6,*) 'MAXIMUM ERROR IN {X2}     =', ERROR
        END IF
C
C (6)   Close FMS files:
        IF(N1 .GT. 0) THEN
           CALL FMSCM (LU_A11)
           CALL FMSCM (LU_F11)
           IF(ISTYPE .NE. 1) THEN
              IF(N2 .GT. 0) CALL FMSCV (LU_A12)
              IF(N3 .GT. 0) CALL FMSCV (LU_A13)
           END IF
           IF(N2 .GT. 0) THEN
              CALL FMSCV (LU_A21)
              CALL FMSCV (LU_F12)
           END IF
           IF(N3 .GT. 0) THEN
              CALL FMSCV (LU_A31)
              CALL FMSCV (LU_F13)
           END IF
           CALL FMSCV (LU_B1 )
           CALL FMSCV (LU_X1 )
        END IF
C
        IF(N22 .GT. 0) THEN
           CALL FMSCM (LU_A22)
           CALL FMSCM (LU_F22)
           CALL FMSCV (LU_B2 )
           CALL FMSCV (LU_X2 )
        END IF
        CALL FMSRMR (RMD, L_TEMP, N_TEMP)
        IF(ASK('Do you want another solution?')) GO TO 10
        CALL FMSPOP (MYNAME)
        CALL FMSEND
 2001   FORMAT (/
     1  ' ( 1) Factor   [A11] into [F11]'/
     2  ' ==============================')
 2002   FORMAT (/
     1  ' ( 2) Solve    [F11]{F12} = {A12}'/
     2  ' ================================')
 2003   FORMAT (/
     1  ' ( 3) Solve    [F11]{F13} = {A13}'/
     2  ' ================================')
 2004   FORMAT (/
```

```
     1  ' ( 4) Solve    [F11]{X1a} = {B1}'/
     2  ' ================================')
 2005   FORMAT (/
     1  ' ( 5) Multiply [F22a] = [A22]  - [A21]T[F12]'/
     2  ' ========================================')
 2006   FORMAT (/
     1  ' ( 6) Multiply [F22b] = [F22a] - [A21]T[F13]'/
     2  ' ========================================')
 2007   FORMAT (/
     1  ' ( 7) Multiply [F22c] = [F22b] - [A31]T[F12]'/
     2  ' ========================================')
 2008   FORMAT (/
     1  ' ( 8) Multiply [F22d] = [F22d] - [A31]T[F13]'/
     2  ' ========================================')
 2009   FORMAT (/
     1  ' ( 9) Multiply {X2a}  = {B2}   - [A21]T{X1a}'/
     2  ' ========================================')
 2010   FORMAT (/
     1  ' (10) Multiply {X2b}  = {X2a}  - [A31]T{X1a}'/
     2  ' ========================================')
 2011   FORMAT (/
     1  ' (11) Factor   [F22d] into [F22]'/
     2  ' ================================')
 2012   FORMAT (/
     1  ' (12) Solve    [F22]{X2} = {X2b}'/
     2  ' ================================')
 2013   FORMAT (/
     1  ' (13) Multiply {X1b}  = {X1a}  - [F13]{X2}'/
     2  ' ======================================')
 2014   FORMAT (/
     1  ' (14) Multiply {X1}   = {X1b}  - [F12]{X2}'/
     2  ' ======================================')
 2015   FORMAT (/
     1  ' (15) Multiply {E1a}  = {B1}   - [A11]{X1}'/
     2  ' ======================================')
 2016   FORMAT (/
     1  ' (16) Multiply {E1b}  = {E1a}  - [A12]{X2}'/
     2  ' ======================================')
 2017   FORMAT (/
     1  ' (17) Multiply {E1}   = {E1b}  - [A13]{X2}'/
     2  ' ======================================')
 2018   FORMAT (/
     1  ' (18) Multiply {E2a}  = {B2}   - [A22]{X2}'/
     2  ' ======================================')
 2019   FORMAT (/
     1  ' (19) Multiply {E2b}  = {E2a}  - [A21]{X1}'/
     2  ' ======================================')
 2020   FORMAT (/
     1  ' (20) Multiply {E2}   = {E2b}  - [A31]{X1}'/
     2  ' ======================================')
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
```

```
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE AGEN (A, IDTYPE, IROW1, IROW2, JCOL1, JCOL2,
     1                   NUMEQ, MTEST, LUA, ISTYPE)
C=======================================================================
        REAL*8     A(IDTYPE,NUMEQ)
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, NUMEQ
        INTEGER    LUA(25)
        REAL*8     A_1, A_DIAG
        REAL*8     R_ZERO, R_ONE
        DATA       R_ZERO/0.0D0/
        DATA       R_ONE /1.0D0/
C
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2
        NROWS = IROW2 - IROW1 + 1
        NCOLS = JCOL2 - JCOL1 + 1
        IEDGE = 0
        IF(IROW1 .GT. JCOL1) THEN
C          This is a off-diagonal block in the lower triangle:
C          Write it out transposed:
           LDU  = 1
           I1   = JCOL1
           I2   = JCOL2
           J1   = IROW1
           J2   = IROW2
           IF(JCOL1 .EQ. 1) IEDGE = 1
        ELSE IF(IROW1 .EQ. JCOL1) THEN
C          This block is on the diagonal:
C          Write it out by columns:
           LDU  = 2
           I1   = IROW1
           I2   = IROW2
           J1   = JCOL1
           J2   = JCOL2
           IF(IROW1 .EQ. 1) IEDGE = 1
        ELSE
C          This block is in the upper triangle:
C          Write it out by columns:
           LDU  = 3
           I1   = IROW1
           I2   = IROW2
           J1   = JCOL1
           J2   = JCOL2
           IF(IROW1 .EQ. 1) IEDGE = 1
        END IF
        LENV = I2 - I1 + 1
C
        IF(MTEST .EQ. 1) THEN
C          Fixed data:
C          Initialize A to default values (generic column):
           DO I = 1,IDTYPE
              DO J=1,LENV
                 A(I,J) = R_ZERO
              END DO
```

```
            END DO
            A_DIAG = R_ONE
            IF(IEDGE .EQ. 1) THEN
               A_1 = -R_ONE
            ELSE
               A_1 = R_ZERO
            END IF
            A(1,1) = A_1
         END IF
C
C        Initialize FMSPUT:
         CALL FMSPUT (LUA, 0, 0, 0, 0, A, 0)
C
C        Loop over the columns:
         DO J = J1,J2
            JC1 = J - J1 + 1
            IF(MTEST .EQ. 1) THEN
C              Use fixed data:
               IF(LDU .EQ. 2) THEN
C                 Diagonal block:
                  IF(J .EQ. 1) THEN
C                    Special case.  This is the first global column:
                     DO I=1,NROWS
                        A(1,I) = -R_ONE
                     END DO
                     A_DIAG = DFLOAT(NUMEQ)
                  END IF
               A(1,JC1) = A_DIAG
               END IF
            ELSE
C              Use random data:
               IF(LDU .EQ. 1) THEN
C                 Lower triangle off-diagonal block; Generate transposed
                  CALL VRAN (A, IDTYPE, J, J, JCOL1, JCOL2,
     1                       NUMEQ, ISTYPE)
               ELSE
C                 Diagonal or upper triangle block:
                  CALL VRAN (A, IDTYPE, IROW1, IROW2, J, J,
     1                       NUMEQ, ISTYPE)
               END IF
            END IF
D           WRITE(6,5001) LENV, JC1, LENV
D5001    FORMAT (' CALL FMSPUT(LUA,',I5,', 1, 1,',I5,', A,',I5,')')
            CALL FMSPUT (LUA, LENV, 1, 1, JC1, A, LENV)
            IF(MTEST .EQ. 1) THEN
C              Restore A to default:
               IF(LDU .EQ. 2) THEN
C                 Diagonal block:
                  IF(J .EQ. 1) THEN
D                    PRINT *,'Restoring first column'
C                    Special case.  This is the first global column:
                     DO I=1,NROWS
                        A(1,I) = R_ZERO
                     END DO
                     A_DIAG = R_ONE
                  END IF
                  A(1,JC1) = R_ZERO
                  A(1,1) = A_1
               END IF
            END IF
         END DO
C
C        End FMSPUT:
         NEND = LENV+1
         CALL FMSPUT (LUA, 0, 0, NEND, 0, A, 0)
```

```
      RETURN
 2000 FORMAT (' Populating A(',I5,':',I5,',',I5,':',I5,')')
      END
C=======================================================================
      SUBROUTINE VRAN (A, IDTYPE, IROW1, IROW2, JCOL1, JCOL2, NUMEQ,
     1                 ISTYPE)
C=======================================================================
      REAL*8    A(IDTYPE, IROW1:IROW2,JCOL1:JCOL2)
      INTEGER   IROW1, IROW2, JCOL1, JCOL2, NUMEQ, ISTYPE
      REAL*8    RANAIJ, SUM
      INTEGER   ISEEDR, ISEEDC
      PARAMETER (ISEEDR=11111111)
      PARAMETER (ISEEDC=12345678)
      DO JCOL = JCOL1,JCOL2
         DO IROW = IROW1,IROW2
            IF(JCOL .LT. IROW) THEN
C              This term is in the lower triangle:
               IY     = IROW*ISEEDR + JCOL*ISEEDC
               IY     = MOD(IY,65536)
               IY     = IY * 11111 + 1
               RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
               RANAIJ = RANAIJ - 0.5D0
            ELSE IF(JCOL .EQ. IROW) THEN
C              This term is on the diagonal:
C              Set the value to (-) the sum of the off-diagonal terms:
               SUM = 0.0D0
               DO II = 1,NUMEQ
                  IF(II .NE. IROW) THEN
                     IF(II .LT. IROW) THEN
C                       This term is in the lower triangle:
                        IY     = IROW*ISEEDR + II*ISEEDC
                     ELSE
C                       This term is in the upper triangle:
                        IF(ISTYPE .EQ. 2) THEN
C                          This matrix is nonsymmetric:
                           IY     = IROW*ISEEDR + II*ISEEDC
                        ELSE
C                          This matrix is symmetric or Hermitian:
                           IY     =  II*ISEEDR + IROW*ISEEDC
                        END IF
                     END IF
                     IY     = MOD(IY,65536)
                     IY     = IY * 11111 + 1
                     RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
                     RANAIJ = RANAIJ - 0.5D0
                     SUM = SUM + RANAIJ
                  END IF
               END DO
D              WRITE(6,2001) IROW, SUM
               RANAIJ = -SUM
            ELSE
C              This term is in the upper triangle:
               IF(ISTYPE .EQ. 2) THEN
C                 This matrix is nonsymmetric:
                  IY     = IROW*ISEEDR + JCOL*ISEEDC
               ELSE
C                 This matrix is symmetric or Hermitian:
                  IY     = JCOL*ISEEDR + IROW*ISEEDC
               END IF
               IY     = MOD(IY,65536)
               IY     = IY * 11111 + 1
               RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
               RANAIJ = RANAIJ - 0.5D0
            END IF
D           WRITE(6,2000) IROW, JCOL, RANAIJ
```

```
               A(1,IROW,JCOL) = RANAIJ
               IF((IROW  .EQ. 1) .AND.
     1            (JCOL  .EQ. 1)) A(1,1,1) = RANAIJ + 1.0D0
               IF(IDTYPE .EQ. 2) THEN
                  IF(ISTYPE .EQ. 3) THEN
C                    This matrix is Hermitian:
                     IF(JCOL .LT. IROW) THEN
                        A(2,IROW,JCOL) = RANAIJ
                     ELSE IF(JCOL .EQ. IROW) THEN
                        A(1,IROW,JCOL) = 2*RANAIJ
                        A(2,IROW,JCOL) = 0.0D0
                     ELSE
                        A(2,IROW,JCOL) = -RANAIJ
                     END IF
                  ELSE
C                    This matrix is Complex Symmetric or Nonsymmetric:
                     A(2,IROW,JCOL) = RANAIJ
                  END IF
               END IF
            END DO
         END DO
         RETURN
D2000    FORMAT (' VRAN: A(',I5,',',I5,')....=',F10.5)
D2001    FORMAT (' VRAN: SUM OF ROW',I5, '...=',F10.5)
         END
C=======================================================================
         SUBROUTINE BGEN (RHS, IDTYPE, IROW1, IROW2, NRHS, LUB)
C=======================================================================
         REAL*8     RHS(IDTYPE, *)
         INTEGER    IDTYPE, IROW1, IROW2, NRHS
         INTEGER    LUB(25)
         REAL*8     R_ZERO, R_ONE
         DATA       R_ZERO/0.0D0/
         DATA       R_ONE /1.0D0/
C
         WRITE(6,2000) IROW1, IROW2, NRHS
C
         NROWS  = IROW2 - IROW1 + 1
         DO I=1,IDTYPE
            DO J=1,NROWS
              RHS(I,J) = R_ZERO
            END DO
         END DO
         IF(IROW1 .EQ. 1) RHS(1,1) = R_ONE
C
C        Initialize FMSPUT:
         CALL FMSPUT (LUB, 0, 0, 0, 0, RHS, 0)
C
C        Write out columns of {B}:
         DO IRHS = 1,NRHS
            CALL FMSPUT (LUB, NROWS, 1, 1, IRHS, RHS, NROWS)
         END DO
C
C        End FMSPUT:
         CALL FMSPUT (LUB, 0, 0, NROWS+1, 0, RHS, 0)
         RETURN
 2000    FORMAT (' Populating B(',I5,':',I5,',',I5,')')
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 16. RESTART DURING FACTORING

This example shows how to use the FMS Parameters IREST and LOWASM to restart the matrix factoring process when it is interrupted at an arbitrary point.
During the first run, which builds and starts factoring the matrix, the input parameters are:

- Is this job being restarted?
  Answer n.
- Enter the***FMS*** Module number (1 to 5)
  1 = Real Symmetric
  2 = Real Nonsymmetric
  3 = Complex Hermitian
  4 = Complex Symmetric
  5 = Complex Nonsymmetric
- Enter the number of equations
- Enter the number of solution vectors
- FMSSET
  You may modify any FMS Parameter.
  When finished, type the letters RETURN

FMS will then form and factor the matrix. As the factoring process proceeds, status information is printed indicating the highest factored equation. When this information prints, all data has been written to the factored matrix file up to that point and you may restart at that equation. You may either let this process complete or interrupt it (Control-C) during factoring.
This example is than run again. During this second run, you will be asked the following questions:

- Is this job being restarted?
  Answer y.
- Enter the highest factored equation
  Enter the highest equation printed by FMS in the previous run.
- Should all files be deleted after this run (y,n)
  Enter y to delete the files.

FMS will then restart the factoring, starting with the segment that contains the equation after the highest factored equation you indicated.

```
C       E X A M P L E   16
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_16')
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
C       Unfactored matrix:
           INTEGER     LUA(25)
C       Factored matrix:
           INTEGER     LUF(25)
C       R.H.S. Vectors:
           INTEGER     LUB(25)
C       Solution Vectors:
           INTEGER     LUX(25)
           INTEGER     LUA0(25)
           DATA        LUA0(1)/0/
           INTEGER     LUX0(25)
           INTEGER     LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Local variables:
C       Data type:
           INTEGER     IDTYPE
C       Loop counter:
           INTEGER     I
```

```
C       Array subscript:
           INTEGER     LX
C       Array length:
           INTEGER     LENX
C       Disk location:
           INTEGER     LDISK
C       Vector number:
           INTEGER     NV
C       First nonfactored equation:
           INTEGER     LOWASM
C       Current error:
           REAL*8      EI
C       Maximum overall error:
           REAL*8      ERROR
C       File delete status (Keep or Delete)
           CHARACTER*6 K_OR_D
C       Restart flag:
           LOGICAL     RESTART
C       Delete files:
           LOGICAL     DELETE
C       Input functions:
           LOGICAL     ASK
           INTEGER     ASK_I
C       Dummy argument:
           INTEGER     IDUM
C       FORTRAN unit number for restart data:
           INTEGER     LUREST
           DATA LUREST/10/
C       Values of ALPHA for xxDAF:
           REAL*8      RZERO(1)
           DATA        RZERO/0.0D0/
           COMPLEX*16  CZERO(1)
           DATA        CZERO/(0.0D0,0.0D0)/
C       Profile vector for a full matrix:
           INTEGER     LOWEQ(1)
           DATA        LOWEQ/-1/
C
C       Common block to communicate with fill routines:
           COMMON /MYDATA/MOD, N, NRHS
```

```
C
C (1)   Initialize FMS:
  100   CONTINUE
        CALL FMSINI
        CALL FMSPSH(MYNAME)
        IDUM   = 0
        CALL FMSIGT ('LUPR', CMD_PTR)
        CALL FMSIST ('IREST', 1)
        RESTART = ASK('Is this job being restarted?')
        IF(.NOT.RESTART) THEN
C          This is a new job:
    1      CONTINUE
           WRITE (6,*) 'The FMS modules are numbered as follows:'
           WRITE (6,*) '   1 = Real Symmetric'
           WRITE (6,*) '   2 = Real Nonsymmetric'
           WRITE (6,*) '   3 = Complex Hermitian'
           WRITE (6,*) '   4 = Complex Symmetric'
           WRITE (6,*) '   5 = Complex Nonsymmetric'
           MOD    = ASK_I('Enter the FMS module number (1 to 5)')
           IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
           N      = ASK_I('Enter the number of equations')
           NRHS   = ASK_I('Enter the number of solution vectors')
           WRITE (6,*) 'You may now alter any FMS parameter.'
           WRITE (6,*) 'When you are finished, type the letters RETURN'
           CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
           IF(MOD .LE. 2) THEN
              IDTYPE = 1
           ELSE
              IDTYPE = 2
           END IF
           NUMEQ  = N
           LOWASM = 1
C          Start at the beginning of the matrix:
           CALL FMSIST ('LOWASM', 1)
C          Create new files:
```

```
            CALL FMSIST ('IEXIST', 0)
C           Keep all files at the end of this run.
            DELETE = .FALSE.
            CALL FMSIST ('IFKEEP', 1)
         ELSE
C           This job is being restarted.
            LOWASM = ASK_I('Enter the highest factored equation')
C           Start at the first unfactored equation:
C           NOTE: FMS will automatically round this down to the first
C           unfactored segment.
            LOWASM = LOWASM + 1
            CALL FMSIST ('LOWASM', LOWASM)
            WRITE(LUPR,2000) MOD, N, NRHS, LOWASM
            CALL FMSIST ('IEXIST', 1)
            DELETE = ASK('Should all files be deleted after this run')
            IF(DELETE) THEN
               CALL FMSIST ('IFKEEP', 0)
               K_OR_D = 'DELETE'
            ELSE
               CALL FMSIST ('IFKEEP', 1)
               K_OR_D = 'KEEP'
            END IF
C
C           Open the restart file:
            OPEN (
     1       UNIT      = LUREST,
     2       FILE      = 'FMS_RESTART',
     3       STATUS    = 'OLD',
     4       FORM      = 'FORMATTED',
     5       IOSTAT    = ISTAT)
            PRINT *,'Restart file ',LUREST,' was reopened'
            IF(ISTAT .NE. 0) THEN
               PRINT *,'Error opening restart file FMS_RESTART'
               PRINT *,'Status of opening file=',ISTAT
               PRINT *,'Delete FMS_RESTART if it already exists'
               PRINT *,'Processing terminated.'
               GO TO 900
            END IF
C           Read in the restart data:
```

```
C          LUA(25):
           DO I=1,25
              READ(LUREST,1000) LUA(I)
           END DO
C          LUF(25):
           DO I=1,25
              READ(LUREST,1000) LUF(I)
           END DO
C          LUB(25):
           DO I=1,25
              READ(LUREST,1000) LUB(I)
           END DO
C          LUX(25):
           DO I=1,25
              READ(LUREST,1000) LUX(I)
           END DO
           CLOSE(UNIT=LUREST,STATUS=K_OR_D,IOSTAT=ISTAT)
           PRINT *,'Restart file ',LUREST,' was closed with status ',
     1             K_OR_D
           IF(ISTAT .NE. 0) THEN
              PRINT *,'Error closing restart file FMS_RESTART'
              PRINT *,'Status of closing file=',ISTAT
              PRINT *,'Processing terminated.'
              GO TO 900
           END IF
C          Print the information read from the restart file:
C          WRITE(LUPR,2005)
C          DO I=1,10
C             WRITE(LUPR,2006) I, LUA(I), LUF(I), LUB(I), LUX(I)
C          END DO
C          DO I=11,15
C             WRITE(LUPR,2006) I, LUA(I), LUF(I)
C          END DO
           N      = LUA( 8)
           IDTYPE = LUA(11)
           ISTYPE = LUA(12)
           NRHS   = LUB( 6)
           NUMEQ  = N
           IF( (IDTYPE.EQ.1) .AND. (ISTYPE.EQ.1) ) MOD = 1
```

```
            IF( (IDTYPE.EQ.1) .AND. (ISTYPE.EQ.2) ) MOD = 2
            IF( (IDTYPE.EQ.2) .AND. (ISTYPE.EQ.3) ) MOD = 3
            IF( (IDTYPE.EQ.2) .AND. (ISTYPE.EQ.1) ) MOD = 4
            IF( (IDTYPE.EQ.2) .AND. (ISTYPE.EQ.2) ) MOD = 5
         END IF
C
C (2)    Open FMS files:
         DO I=11,25
            LUB(I) = 0
            LUX(I) = 0
         END DO
         CALL FMSOV (N, IDTYPE, NRHS, 'LUB', LUB)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C        Allocate a vector of storage:
         IF(IDTYPE .EQ. 1) THEN
            LENX = LUX(4)
            CALL FMSRMG (RMD, LX, LENX)
         ELSE
            LENX = LUX(4)/2
            CALL FMSCMG (CMD, LX, LENX)
         END IF
C        NOTE: When these files are being reopened (due to restart),
C        the file attribute lists must be defined.  In addition,
C        the FMS Parameter IEXIST must be 1.
         IF(LOWASM .LE. N) THEN
            IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
            CALL FMSOM (LUA, 'LUF', LUF)
            IF(RESTART) GO TO 400
         ELSE
C           Start with the solving:
            IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUF', LUF)
            IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
            IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUF', LUF)
            IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUF', LUF)
```

```
            IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
            GO TO 450
         END IF
C
C (3)    Write data to FMS files:
C
C        Matrix File:
         CALL FMSIST ('MDATAU', 1)
         IF(MOD.EQ.1) CALL RSDF
     1     (LUA0, RZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.2) CALL RNDAF
     1     (LUA0, RZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.3) CALL CHDAF
     1     (LUA0, RZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.4) CALL CSDAF
     1     (LUA0, CZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.5) CALL CNDAF
     1     (LUA0, CZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
C
C        R.H.S. Vector File:
C
C        Populate test vector:
         IF(IDTYPE .EQ. 1) THEN
            DO 10 I = 1,LENX
               RMD(LX-1+I) = 0.0D0
   10       CONTINUE
            RMD(LX) = 1.0D0
         ELSE
            DO 11 I = 1,LENX
               CMD(LX-1+I) = (0.0D0,0.0D0)
   11       CONTINUE
            IF(MOD.EQ.3) THEN
               CMD(LX) = (1.0D0,0.0D0)
            ELSE
               CMD(LX) = (1.0D0,1.0D0)
            END IF
         END IF
C
         LDISK  = 1
```

```
         DO 30 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSWRT (LUB(1), LDISK, RMD(LX), LUB(4))
            ELSE
               CALL FMSWRT (LUB(1), LDISK, CMD(LX), LUB(4))
            END IF
            LDISK = LDISK + LUB(4)
   30    CONTINUE
C
C        Open the restart file:
         OPEN (
     1    UNIT      = LUREST,
     2    FILE      = 'FMS_RESTART',
     3    STATUS    = 'NEW',
     4    FORM      = 'FORMATTED',
     5    IOSTAT    = ISTAT)
         PRINT *,'Restart file ',LUREST,' was created and opened'
         IF(ISTAT .NE. 0) THEN
            PRINT *,'Error opening restart file FMS_RESTART'
            PRINT *,'Status of OPEN=',ISTAT
            PRINT *,'Processing terminated.'
            GO TO 900
         END IF
C
C        Save file attribute arrays for restart:
C        LUA(25):
         DO I=1,25
            WRITE(LUREST,2001) I, LUA(I)
         END DO
C        LUF(25):
         DO I=1,25
            WRITE(LUREST,2002) I, LUF(I)
         END DO
C        LUB(25):
         DO I=1,25
            WRITE(LUREST,2003) I, LUB(I)
         END DO
C        LUX(25):
         DO I=1,25
```

```
            WRITE(LUREST,2004) I, LUX(I)
         END DO
C        Print the information written to the restart file:
C        WRITE(LUPR,2005)
C        DO I=1,10
C           WRITE(LUPR,2006) I, LUA(I), LUF(I), LUB(I), LUX(I)
C        END DO
C        DO I=11,25
C           WRITE(LUPR,2006) I, LUA(I), LUF(I)
C        END DO
         CLOSE(UNIT=LUREST,STATUS='KEEP',IOSTAT=ISTAT)
         PRINT *,'Restart file ',LUREST,' was closed with status KEEP'
         IF(ISTAT .NE. 0) THEN
            PRINT *,'Error closing restart file FMS_RESTART'
            PRINT *,'Status of closing file=',ISTAT
            PRINT *,'Processing terminated.'
            GO TO 900
         END IF
C
C (4)    Perform matrix algebra:
  400    CONTINUE
         IF(MOD.EQ.1) CALL RSDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.2) CALL RNDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.3) CALL CHDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.4) CALL CSDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.5) CALL CNDF (LUA, LUF, LUB, LUX,  NUMRED)
C
C        Update the restart file:
         IF(.NOT.DELETE) THEN
            OPEN (
     1       UNIT      = LUREST,
     2       FILE      = 'FMS_RESTART',
     3       STATUS    = 'OLD',
     4       FORM      = 'FORMATTED',
     5       IOSTAT    = ISTAT)
            IF(ISTAT .NE. 0) THEN
               PRINT *,'Error opening restart file FMS_RESTART'
               PRINT *,'Status of OPEN=',ISTAT
```

```
               PRINT *,'Processing terminated.'
               GO TO 900
            END IF
C           LUA(25):
            DO I=1,25
               WRITE(LUREST,2001) I, LUA(I)
            END DO
C           LUF(25):
            DO I=1,25
               WRITE(LUREST,2002) I, LUF(I)
            END DO
C           LUB(25):
            DO I=1,25
               WRITE(LUREST,2003) I, LUB(I)
            END DO
C           LUX(25):
            DO I=1,25
               WRITE(LUREST,2004) I, LUX(I)
            END DO
            CLOSE(UNIT=LUREST,STATUS='KEEP',IOSTAT=ISTAT)
            PRINT *,'Restart file ',LUREST,' was updated'
            IF(ISTAT .NE. 0) THEN
               PRINT *,'Error closing restart file FMS_RESTART'
               PRINT *,'Status of closing file=',ISTAT
               PRINT *,'Processing terminated.'
               GO TO 900
            END IF
         END IF
  450    CONTINUE
         IF(MOD.EQ.1) CALL RSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.3) CALL CHDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.4) CALL CSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUB, LUX, NRHS, ISKIP)
C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
         LDISK  = 1
```

```
         DO 60 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRED (LUX(1), LDISK, RMD(LX), N)
               DO 50 I = 1,N
                  EI = ABS(RMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   50          CONTINUE
            ELSE
               CALL FMSRED (LUX(1), LDISK, CMD(LX), 2*N)
               DO 51 I = 1,N
                  EI = ABS(CMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   51          CONTINUE
            END IF
            LDISK  = LDISK  + LUX(4)
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         IF(LOWASM .LE. N) CALL FMSCM (LUA)
         CALL FMSCM (LUF)
         CALL FMSCV (LUB)
         CALL FMSCV (LUX)
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LX, LENX)
         ELSE
            CALL FMSCMR (CMD, LX, LENX)
         END IF
         IF(ASK('Do you want another solution?')) GO TO 100
         CALL FMSPOP(MYNAME)
         CALL FMSEND
   900   CONTINUE
C        ---------------------------------------------------------------
C        FORMAT STATEMENTS
C        ---------------------------------------------------------------
 1000    FORMAT (9X,I15)
 2000    FORMAT (/
     1   ' Problem is being restarted:'/
     2   ' FMS Module number.........=',I10/
```

```
     3  ' Number of equations........=',I10/
     4  ' Number of R.H.S. vectors...=',I10/
     5  ' Restart equation...........=',I10/)
 2001   FORMAT (' LUA(',I2,')=',I15)
 2002   FORMAT (' LUF(',I2,')=',I15)
 2003   FORMAT (' LUB(',I2,')=',I15)
 2004   FORMAT (' LUX(',I2,')=',I15)
C2005   FORMAT (/
C    1  'Contents of restart file:'/
C    2  '           LUA       LUF       LUB       LUX'/
C    3  '       =======   =======   =======   =======')
C2006   FORMAT ('(',I2,')',4I10)
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        WRITE(6,2001) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
 2001   FORMAT (4X,'You entered ',A)
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        WRITE(6,2001) ASK_I
        RETURN
```

```
 2000   FORMAT (1X,A,'>')
 2001   FORMAT (4X,'You entered',I10)
        END
C=======================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), DIA, OFFDIA
        PARAMETER  (DIA   = 1.0D0)
        PARAMETER  (OFFDIA=-1.0D0)
        COMMON /MYDATA/MOD, N, NRHS
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20   CONTINUE
        RETURN
        END

C=======================================================================
        SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
```

```
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), OFFDIA
        PARAMETER  (OFFDIA=-1.0D0)
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 20 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
   20   CONTINUE
        RETURN
        END

C=======================================================================
        SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     D(*), DIA
        COMPLEX*16 A(0:*), OFFDIA
        PARAMETER  (DIA   =  1.0D0        )
        PARAMETER  (OFFDIA=(-1.0D0,0.0D0))
        COMMON /MYDATA/MOD, N, NRHS
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
```

```
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20   CONTINUE
        RETURN
        END

C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), DIA, OFFDIA
        PARAMETER  (DIA   =( 1.0D0, 1.0D0))
        PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
        COMMON /MYDATA/MOD, N, NRHS
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20   CONTINUE
        RETURN
        END
```

```
C=======================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), OFFDIA
        PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 20 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
   20   CONTINUE
        RETURN
        END

C=======================================================================
        SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C=======================================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        REAL*8     A(0:*), DIA, OFFDIA
        PARAMETER  (DIA   = 1.0D0)
        PARAMETER  (OFFDIA=-1.0D0)
C
C       Fill first row and diagonal:
        LU1    = LUFLAG(1)
        LI1    = LOCI(1)
        DO 10 J = JEQN1,JEQN2
           LIJ = LI1 + LOCJ(J,LU1)
           A(LIJ) = OFFDIA
           LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
           A(LD ) = DIA
   10   CONTINUE
```

```
C
C       Fill column 1 if this is the first slab:
        IF(JEQN1 .EQ. 1) THEN
           DO 20 I = 1,NUMEQ
              LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
              A(LIJ) = OFFDIA
   20      CONTINUE
           LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
           A(LD) = DFLOAT(NUMEQ)
        END IF
        RETURN
        END

C=======================================================================
        SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C=======================================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        COMPLEX*16 A(0:*), DIA, OFFDIA
        PARAMETER  (DIA   =( 1.0D0, 1.0D0))
        PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C       Fill first row and diagonal:
        LU1    = LUFLAG(1)
        LI1    = LOCI(1)
        DO 10 J = JEQN1,JEQN2
           LIJ = LI1 + LOCJ(J,LU1)
           A(LIJ) = OFFDIA
           LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
           A(LD ) = DIA
   10   CONTINUE
C
C       Fill column 1 if this is the first slab:
        IF(JEQN1 .EQ. 1) THEN
           DO 20 I = 1,NUMEQ
              LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
              A(LIJ) = OFFDIA
   20      CONTINUE
           LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
```

```
            A(LD) = DCMPLX(NUMEQ,NUMEQ)
        END IF
        RETURN
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   16
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_16')
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
C       Unfactored matrix:
           INTEGER     LUA(25)
C       Factored matrix:
           INTEGER     LUF(25)
C       R.H.S. Vectors:
           INTEGER     LUB(25)
C       Solution Vectors:
           INTEGER     LUX(25)
           INTEGER     LUA0(25)
           DATA        LUA0(1)/0/
           INTEGER     LUX0(25)
           INTEGER     LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Local variables:
C       Data type:
           INTEGER     IDTYPE
C       Loop counter:
           INTEGER     I
C       Array subscript:
           INTEGER     LX
C       Array length:
           INTEGER     LENX
C       Disk location:
           INTEGER     LDISK
C       Vector number:
           INTEGER     NV
C       First nonfactored equation:
           INTEGER     LOWASM
C       Current error:
           REAL*8      EI
C       Maximum overall error:
           REAL*8      ERROR
C       File delete status (Keep or Delete)
           CHARACTER*6 K_OR_D
C       Restart flag:
           LOGICAL     RESTART
C       Delete files:
           LOGICAL     DELETE
C       Input functions:
           LOGICAL     ASK
           INTEGER     ASK_I
C       Dummy argument:
```

```
            INTEGER     IDUM
C        FORTRAN unit number for restart data:
            INTEGER     LUREST
            DATA LUREST/10/
C        Values of ALPHA for xxDAF:
            REAL*8      RZERO(1)
            DATA        RZERO/0.0D0/
            COMPLEX*16  CZERO(1)
            DATA        CZERO/(0.0D0,0.0D0)/
C        Profile vector for a full matrix:
            INTEGER     LOWEQ(1)
            DATA        LOWEQ/-1/
C
C        Common block to communicate with fill routines:
         COMMON /MYDATA/MOD, N, NRHS
C
C (1)    Initialize FMS:
  100    CONTINUE
         CALL FMSINI
         CALL FMSPSH (MYNAME)
         IDUM   = 0
         CALL FMSIGT ('LUPR',   LUPR)
         CALL FMSIST ('IREST', 1)
         RESTART = ASK('Is this job being restarted?')
         IF(.NOT.RESTART) THEN
C           This is a new job:
    1       CONTINUE
            WRITE (6,*) 'The FMS modules are numbered as follows:'
            WRITE (6,*) '   1 = Real Symmetric'
            WRITE (6,*) '   2 = Real Nonsymmetric'
            WRITE (6,*) '   3 = Complex Hermitian'
            WRITE (6,*) '   4 = Complex Symmetric'
            WRITE (6,*) '   5 = Complex Nonsymmetric'
            MOD    = ASK_I('Enter the FMS module number (1 to 5)')
            IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
            N      = ASK_I('Enter the number of equations')
            NRHS   = ASK_I('Enter the number of solution vectors')
            WRITE (6,*) 'You may now alter any FMS parameter.'
            WRITE (6,*) 'When you are finished, type the letters RETURN'
            CALL FMSSET
         CALL FMSIGT ('MEMPTR', IMD_PTR)
         CALL FMSIGT ('MEMPTR', RMD_PTR)
         CALL FMSIGT ('MEMPTR', CMD_PTR)
            IF(MOD .LE. 2) THEN
               IDTYPE = 1
            ELSE
               IDTYPE = 2
            END IF
            NUMEQ  = N
            LOWASM = 1
C           Start at the beginning of the matrix:
            CALL FMSIST ('LOWASM', 1)
C           Create new files:
            CALL FMSIST ('IEXIST', 0)
C           Keep all files at the end of this run.
            DELETE = .FALSE.
            CALL FMSIST ('IFKEEP', 1)
         ELSE
C           This job is being restarted.
            LOWASM = ASK_I('Enter the highest factored equation')
C           Start at the first unfactored equation:
C           NOTE: FMS will automatically round this down to the first
C           unfactored segment.
            LOWASM = LOWASM + 1
            CALL FMSIST ('LOWASM', LOWASM)
```

```
            WRITE(LUPR,2000) MOD, N, NRHS, LOWASM
            CALL FMSIST ('IEXIST', 1)
            DELETE = ASK('Should all files be deleted after this run')
            IF(DELETE) THEN
               CALL FMSIST ('IFKEEP', 0)
               K_OR_D = 'DELETE'
            ELSE
               CALL FMSIST ('IFKEEP', 1)
               K_OR_D = 'KEEP'
            END IF
C
C           Open the restart file:
            OPEN (
     1       UNIT      = LUREST,
     2       FILE      = 'FMS_RESTART',
     3       STATUS    = 'OLD',
     4       FORM      = 'FORMATTED',
     5       IOSTAT    = ISTAT)
            PRINT *,'Restart file ',LUREST,' was reopened'
            IF(ISTAT .NE. 0) THEN
               PRINT *,'Error opening restart file FMS_RESTART'
               PRINT *,'Status of opening file=',ISTAT
               PRINT *,'Delete FMS_RESTART if it already exists'
               PRINT *,'Processing terminated.'
               GO TO 900
            END IF
C           Read in the restart data:
C           LUA(25):
            DO I=1,25
               READ(LUREST,1000) LUA(I)
            END DO
C           LUF(25):
            DO I=1,25
               READ(LUREST,1000) LUF(I)
            END DO
C           LUB(25):
            DO I=1,25
               READ(LUREST,1000) LUB(I)
            END DO
C           LUX(25):
            DO I=1,25
               READ(LUREST,1000) LUX(I)
            END DO
            CLOSE(UNIT=LUREST,STATUS=K_OR_D,IOSTAT=ISTAT)
            PRINT *,'Restart file ',LUREST,' was closed with status ',
     1              K_OR_D
            IF(ISTAT .NE. 0) THEN
               PRINT *,'Error closing restart file FMS_RESTART'
               PRINT *,'Status of closing file=',ISTAT
               PRINT *,'Processing terminated.'
               GO TO 900
            END IF
C           Print the information read from the restart file:
C           WRITE(LUPR,2005)
C           DO I=1,10
C              WRITE(LUPR,2006) I, LUA(I), LUF(I), LUB(I), LUX(I)
C           END DO
C           DO I=11,15
C              WRITE(LUPR,2006) I, LUA(I), LUF(I)
C           END DO
            N      = LUA( 8)
            IDTYPE = LUA(11)
            ISTYPE = LUA(12)
            NRHS   = LUB( 6)
            NUMEQ  = N
```

```
            IF( (IDTYPE.EQ.1) .AND. (ISTYPE.EQ.1) ) MOD = 1
            IF( (IDTYPE.EQ.1) .AND. (ISTYPE.EQ.2) ) MOD = 2
            IF( (IDTYPE.EQ.2) .AND. (ISTYPE.EQ.3) ) MOD = 3
            IF( (IDTYPE.EQ.2) .AND. (ISTYPE.EQ.1) ) MOD = 4
            IF( (IDTYPE.EQ.2) .AND. (ISTYPE.EQ.2) ) MOD = 5
         END IF
C
C (2)    Open FMS files:
         DO I=11,25
            LUB(I) = 0
            LUX(I) = 0
         END DO
         CALL FMSOV (N, IDTYPE, NRHS, 'LUB', LUB)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C        Allocate a vector of storage:
         IF(IDTYPE .EQ. 1) THEN
            LENX = LUX(4)
            CALL FMSRMG (RMD, LX, LENX)
         ELSE
            LENX = LUX(4)/2
            CALL FMSCMG (CMD, LX, LENX)
         END IF
C        NOTE: When these files are being reopened (due to restart),
C        the file attribute lists must be defined.  In addition,
C        the FMS Parameter IEXIST must be 1.
         IF(LOWASM .LE. N) THEN
            IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
            IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
            CALL FMSOM (LUA, 'LUF', LUF)
            IF(RESTART) GO TO 400
         ELSE
C           Start with the solving:
            IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUF', LUF)
            IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
            IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUF', LUF)
            IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUF', LUF)
            IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
            GO TO 450
         END IF
C
C (3)    Write data to FMS files:
C
C        Matrix File:
         CALL FMSIST ('MDATAU', 1)
         IF(MOD.EQ.1) CALL RSDAF
     1     (LUA0, RZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.2) CALL RNDAF
     1     (LUA0, RZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.3) CALL CHDAF
     1     (LUA0, RZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.4) CALL CSDAF
     1     (LUA0, CZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
         IF(MOD.EQ.5) CALL CNDAF
     1     (LUA0, CZERO, 0, LUS0, 0, LUA, LUA0, LUX0, LUX0, 0)
C
C        R.H.S. Vector File:
C
C        Populate test vector:
         IF(IDTYPE .EQ. 1) THEN
            DO 10 I = 1,LENX
```

```
               RMD(LX-1+I) = 0.0D0
   10       CONTINUE
            RMD(LX) = 1.0D0
         ELSE
            DO 11 I = 1,LENX
               CMD(LX-1+I) = (0.0D0,0.0D0)
   11       CONTINUE
            IF(MOD.EQ.3) THEN
               CMD(LX) = (1.0D0,0.0D0)
            ELSE
               CMD(LX) = (1.0D0,1.0D0)
            END IF
         END IF
C
         LDISK  = 1
         DO 30 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSWRT (LUB(1), LDISK, RMD(LX), LUB(4))
            ELSE
               CALL FMSWRT (LUB(1), LDISK, CMD(LX), LUB(4))
            END IF
            LDISK = LDISK + LUB(4)
   30    CONTINUE
C
C        Open the restart file:
         OPEN (
     1    UNIT      = LUREST,
     2    FILE      = 'FMS_RESTART',
     3    STATUS    = 'NEW',
     4    FORM      = 'FORMATTED',
     5    IOSTAT    = ISTAT)
         PRINT *,'Restart file ',LUREST,' was created and opened'
         IF(ISTAT .NE. 0) THEN
            PRINT *,'Error opening restart file FMS_RESTART'
            PRINT *,'Status of OPEN=',ISTAT
            PRINT *,'Processing terminated.'
            GO TO 900
         END IF
C
C        Save file attribute arrays for restart:
C        LUA(25):
         DO I=1,25
            WRITE(LUREST,2001) I, LUA(I)
         END DO
C        LUF(25):
         DO I=1,25
            WRITE(LUREST,2002) I, LUF(I)
         END DO
C        LUB(25):
         DO I=1,25
            WRITE(LUREST,2003) I, LUB(I)
         END DO
C        LUX(25):
         DO I=1,25
            WRITE(LUREST,2004) I, LUX(I)
         END DO
C        Print the information written to the restart file:
C        WRITE(LUPR,2005)
C        DO I=1,10
C           WRITE(LUPR,2006) I, LUA(I), LUF(I), LUB(I), LUX(I)
C        END DO
C        DO I=11,25
C           WRITE(LUPR,2006) I, LUA(I), LUF(I)
C        END DO
         CLOSE(UNIT=LUREST,STATUS='KEEP',IOSTAT=ISTAT)
```

```
         PRINT *,'Restart file ',LUREST,' was closed with status KEEP'
         IF(ISTAT .NE. 0) THEN
            PRINT *,'Error closing restart file FMS_RESTART'
            PRINT *,'Status of closing file=',ISTAT
            PRINT *,'Processing terminated.'
            GO TO 900
         END IF
C
C (4)    Perform matrix algebra:
  400    CONTINUE
         IF(MOD.EQ.1) CALL RSDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.2) CALL RNDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.3) CALL CHDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.4) CALL CSDF (LUA, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.5) CALL CNDF (LUA, LUF, LUB, LUX,  NUMRED)
C
C        Update the restart file:
         IF(.NOT.DELETE) THEN
            OPEN (
     1       UNIT      = LUREST,
     2       FILE      = 'FMS_RESTART',
     3       STATUS    = 'OLD',
     4       FORM      = 'FORMATTED',
     5       IOSTAT    = ISTAT)
            IF(ISTAT .NE. 0) THEN
               PRINT *,'Error opening restart file FMS_RESTART'
               PRINT *,'Status of OPEN=',ISTAT
               PRINT *,'Processing terminated.'
               GO TO 900
            END IF
C           LUA(25):
            DO I=1,25
               WRITE(LUREST,2001) I, LUA(I)
            END DO
C           LUF(25):
            DO I=1,25
               WRITE(LUREST,2002) I, LUF(I)
            END DO
C           LUB(25):
            DO I=1,25
               WRITE(LUREST,2003) I, LUB(I)
            END DO
C           LUX(25):
            DO I=1,25
               WRITE(LUREST,2004) I, LUX(I)
            END DO
            CLOSE(UNIT=LUREST,STATUS='KEEP',IOSTAT=ISTAT)
            PRINT *,'Restart file ',LUREST,' was updated'
            IF(ISTAT .NE. 0) THEN
               PRINT *,'Error closing restart file FMS_RESTART'
               PRINT *,'Status of closing file=',ISTAT
               PRINT *,'Processing terminated.'
               GO TO 900
            END IF
         END IF
  450    CONTINUE
         IF(MOD.EQ.1) CALL RSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.3) CALL CHDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.4) CALL CSDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUB, LUX, NRHS, ISKIP)
C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
```

```
         LDISK  = 1
         DO 60 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRED (LUX(1), LDISK, RMD(LX), N)
               DO 50 I = 1,N
                  EI = ABS(RMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   50          CONTINUE
            ELSE
               CALL FMSRED (LUX(1), LDISK, CMD(LX), 2*N)
               DO 51 I = 1,N
                  EI = ABS(CMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   51          CONTINUE
            END IF
            LDISK  = LDISK  + LUX(4)
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         IF(LOWASM .LE. N) CALL FMSCM (LUA)
         CALL FMSCM (LUF)
         CALL FMSCV (LUB)
         CALL FMSCV (LUX)
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LX, LENX)
         ELSE
            CALL FMSCMR (CMD, LX, LENX)
         END IF
         IF(ASK('Do you want another solution?')) GO TO 100
         CALL FMSPOP (MYNAME)
         CALL FMSEND
  900    CONTINUE
C        ---------------------------------------------------------------
C        FORMAT STATEMENTS
C        ---------------------------------------------------------------
 1000    FORMAT (9X,I15)
 2000    FORMAT (/
     1   ' Problem is being restarted:'/
     2   ' FMS Module number..........=',I10/
     3   ' Number of equations........=',I10/
     4   ' Number of R.H.S. vectors...=',I10/
     5   ' Restart equation...........=',I10/)
 2001    FORMAT (' LUA(',I2,')=',I15)
 2002    FORMAT (' LUF(',I2,')=',I15)
 2003    FORMAT (' LUB(',I2,')=',I15)
 2004    FORMAT (' LUX(',I2,')=',I15)
C2005    FORMAT (/
C    1   'Contents of restart file:'/
C    2   '           LUA       LUF       LUB       LUX'/
C    3   '       =======   =======   =======   ======='/)
C2006    FORMAT ('(',I2,')',4I10)
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         WRITE(6,2001) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
```

```
            ASK = .FALSE.
         END IF
         RETURN
 1000    FORMAT (A)
 2000    FORMAT (1X,A,' (y,n)>')
 2001    FORMAT (4X,'You entered ',A)
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
         CHARACTER* (*) STRING
         WRITE(6,2000) STRING
         READ (5,*) ASK_I
         WRITE(6,2001) ASK_I
         RETURN
 2000    FORMAT (1X,A,'>')
 2001    FORMAT (4X,'You entered',I10)
         END
C=======================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*), DIA, OFFDIA
         PARAMETER  (DIA   = 1.0D0)
         PARAMETER  (OFFDIA=-1.0D0)
         COMMON /MYDATA/MOD, N, NRHS
C
C        Populate the diagonal with test data:
         IF(IROW2 .EQ. JCOL2) THEN
C           This is a diagonal block:
            DO 10 I = IROW1,IROW2
               D(I) = DIA
   10       CONTINUE
            IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
         END IF
C
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
         DO 20 I = IROW1,IROW2
            J = LOWEQ(I)
            IF( (J .GE. JCOL1) .AND.
     1          (J .LE. JCOL2) .AND.
     2          (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20    CONTINUE
         RETURN
         END

C=======================================================================
         SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*), OFFDIA
         PARAMETER  (OFFDIA=-1.0D0)
C
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
         DO 20 J = JCOL1,JCOL2
            I = LOWEQ(J)
            IF( (I .GE. IROW1) .AND.
     1          (I .LE. IROW2) .AND.
     2          (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
```

```
   20   CONTINUE
        RETURN
        END

C=======================================================================
        SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     D(*), DIA
        COMPLEX*16 A(0:*), OFFDIA
        PARAMETER  (DIA   =  1.0D0        )
        PARAMETER  (OFFDIA=(-1.0D0,0.0D0))
        COMMON /MYDATA/MOD, N, NRHS
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DFLOAT(N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20   CONTINUE
        RETURN
        END

C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), DIA, OFFDIA
        PARAMETER  (DIA   =( 1.0D0, 1.0D0))
        PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
        COMMON /MYDATA/MOD, N, NRHS
C
C       Populate the diagonal with test data:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO 10 I = IROW1,IROW2
              D(I) = DIA
   10      CONTINUE
           IF(IROW1 .EQ. 1) D(1) = DCMPLX(N,N)
        END IF
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
        DO 20 I = IROW1,IROW2
           J = LOWEQ(I)
           IF( (J .GE. JCOL1) .AND.
     1         (J .LE. JCOL2) .AND.
     2         (J .LT.     I) ) A(LOCEQ(I) + IJSTEP*J) = OFFDIA
   20   CONTINUE
        RETURN
```

```
        END

C=======================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*), OFFDIA
        PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C       Populate profile of [AL] with test data:
C       The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
        DO 20 J = JCOL1,JCOL2
           I = LOWEQ(J)
           IF( (I .GE. IROW1) .AND.
     1         (I .LE. IROW2) .AND.
     2         (I .LT.     J) ) A(LOCEQ(J) + IJSTEP*I) = OFFDIA
   20   CONTINUE
        RETURN
        END

C=======================================================================
        SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C=======================================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        REAL*8     A(0:*), DIA, OFFDIA
        PARAMETER  (DIA   = 1.0D0)
        PARAMETER  (OFFDIA=-1.0D0)
C
C       Fill first row and diagonal:
        LU1    = LUFLAG(1)
        LI1    = LOCI(1)
        DO 10 J = JEQN1,JEQN2
           LIJ = LI1 + LOCJ(J,LU1)
           A(LIJ) = OFFDIA
           LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
           A(LD ) = DIA
   10   CONTINUE
C
C       Fill column 1 if this is the first slab:
        IF(JEQN1 .EQ. 1) THEN
           DO 20 I = 1,NUMEQ
              LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
              A(LIJ) = OFFDIA
   20      CONTINUE
           LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
           A(LD) = DFLOAT(NUMEQ)
        END IF
        RETURN
        END

C=======================================================================
        SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C=======================================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        COMPLEX*16 A(0:*), DIA, OFFDIA
        PARAMETER  (DIA   =( 1.0D0, 1.0D0))
        PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
C
C       Fill first row and diagonal:
        LU1    = LUFLAG(1)
```

```
            LI1    = LOCI(1)
            DO 10 J = JEQN1,JEQN2
               LIJ = LI1 + LOCJ(J,LU1)
               A(LIJ) = OFFDIA
               LD  = LOCI(J) + LOCJ(J,LUFLAG(J))
               A(LD ) = DIA
      10    CONTINUE
      C
      C        Fill column 1 if this is the first slab:
            IF(JEQN1 .EQ. 1) THEN
               DO 20 I = 1,NUMEQ
                  LIJ = LOCI(I) + LOCJ(1,LUFLAG(I))
                  A(LIJ) = OFFDIA
      20        CONTINUE
               LD  = LOCI(1) + LOCJ(1,LUFLAG(1))
               A(LD) = DCMPLX(NUMEQ,NUMEQ)
            END IF
            RETURN
            END
```

Copyright © Multipath Corporation

FMS EXAMPLE 17. BLOCK ITERATIVE SOLUTIONS

This example shows how the FMS subroutines may be used to perform a sequence of iterative solutions.

```
+-    +    -+ +-  -+   +-  -+
| A11 | A12 | | X1 |   | B1 |
+-----+-----+ +----+ = +----+
| A21 | A22 | | X2 |   | B2 |
+-    +    -+ +-  -+   +-  -+
```

The system of equations is partitioned as shown above. The diagonal blocks [A11] and [A22] are factored but the off-diagonal blocks [A12] and [A21] are not. A sequence of iterative solutions is performed using Jacobi or Gauss-Seidel iteration to obtain approximate solutions {X1} and {X2}.
This approach can save computer time over the exact solution under the following conditions:

- The matrix values in the diagonal blocks [A11] and [A22] are large compared to the values in the off-diagonal blocks [A12] and [A21].
- An approximate solution to {X1} and {X2} is acceptable.
- The number of vectors, times the number of iterations, is less than the dimension of [A22].

In the Jacobi solution, the "old" values of {X1} and {X2} are used to compute the new values of {X1} and {X2}. In the Gauss-Seidel solution, the old value of {X2} is first used to compute a new value of {X1}. This updated value of {X1} is then used to compute the new value of {X2}. Generally the Gauss-Seidel method will converge faster than the Jacobi method.
A variation of the Jacobi method is also presented where the solution values are the change in {DX1} and {DX2}. This approach is numerically equivalent to the standard Jacobi method that solves directly for the solution {X1} and {X2}.

Subroutine J_GS provides both Jacobi and Gauss-Seidel solutions. Subroutine Jacobi2 uses the incremental Jacobi approach.

The following input is used to test these subroutines:

- Number of equations in [A11]
- Number of equations in [A22]
- Number of solution vectors
- Maximum number of iterations to perform

- Maximum solution error to terminate iterations
- Solution method
  - 1=Jacobi
  - 2=Gauss-Seidel
  - 3=Incremental Jacobi
- Vector norm used to test for convergence
  - 0=Infinity norm (maximum absolute value)
  - 1=Sum of the absolute values
  - 2=Square root of the sum of the values squared
- If FMS output is required during the iterations.
- Any FMS Parameter

After solving the system, you are asked the following:

- Do you want another solution?
- Do you want a new A11?
- Do you want a new A22?

If you answer yes to any of these questions, the problem is solved again, recomputing only those terms that are necessary.

When all solutions of the given problem are complete, you are asked if you want to solve another problem, in which any of the input may change.

```
C       T E S T     J A C O B I   A N D   G A U S S - S E I D E L
C
C       This program tests subroutines J_GS and Jacobi2, which
C       illustrate how the FMS subroutines may be used for a block
C       iterative solution.
C
C       The test matrix is of the form
C
C               [A]         {X}  =  {B}
C       +-             -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
```

```
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-            -+ +   +   +   +
C
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_17')
C
C       FMS matrix and vector file attributes:
        INTEGER     LU_A11(25)
        INTEGER     LU_A12(25)
        INTEGER     LU_A21(25)
        INTEGER     LU_A22(25)
        INTEGER     LU_B1 (25), LU_X1(25)
        INTEGER     LU_B2 (25), LU_X2(25)
C
C
FMS memory management requires the following arrays:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Data Type:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE=2)
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
        REAL*8  ASK_R
C
C       Solution methods:
        INTEGER    METHOD_JACOBI
        PARAMETER (METHOD_JACOBI          =1)
        INTEGER    METHOD_GAUSS_SEIDEL
        PARAMETER (METHOD_GAUSS_SEIDEL    =2)
```

```
      INTEGER    METHOD_JACOBI_INCREMENT
      PARAMETER (METHOD_JACOBI_INCREMENT =3)
C
C     LOCAL VARIABLES:
C     Number of equations in [A11]:
         INTEGER N1
C     Number of equations in [A22]:
         INTEGER N2
C     Maximum of N1 and N2:
         INTEGER NMAX
C     Number of solution vectors:
         INTEGER NRHS
C     Form a new [A11] matrix:
         LOGICAL NEWA11
C     Form a new [A22] matrix:
         LOGICAL NEWA22
C     Type of solution method:
         INTEGER METHOD
C     Print FMS output during interation:
         LOGICAL NOFMSP
C     Maximum number of iterations:
         INTEGER MAXITR
C     Maximum iteration error:
         REAL*8  ERRMAX
C     Error on current term:
         REAL*8  EI
C     Maximum overall error:
         REAL*8  ERROR
C     CMD Pointer to scratch vector:
         INTEGER L_TEMP
C     Profile vector for a full matrix:
      INTEGER    LOWEQ(1)
      DATA       LOWEQ/-1/
C
C     CONSTANTS:
         REAL*8     R_ZERO
         DATA R_ZERO/0.0D0/
         COMPLEX*16 C_ONE
         DATA C_ONE/(1.0D0,0.0D0)/
```

```
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
C
C       Use full column partial pivoting (SLAB format):
        CALL FMSIST ('MFMAT, 3)
C
C       Read in problem size parameters:
   10   CONTINUE
        N1     = ASK_I('Enter the number of equations in A11.....')
        N2     = ASK_I('Enter the number of equations in A22.....')
        NRHS   = ASK_I('Enter the number of solution vectors.....')
        MAXITR = ASK_I('Enter the maximum number of iterations...')
        ERRMAX = ASK_R('Enter the maximum error..................')
        WRITE(6,*) 'The following solution methods are available:'
        WRITE(6,*) '   1 = Jacobi,'
        WRITE(6,*) '   2 = Gauss-Seidel,'
        WRITE(6,*) '   3 = Incremental Jacobi.'
        METHOD = ASK_I('Enter solution method (1 | 2 | 3 ).......')
        WRITE(6,*) 'The following vector norms are available:'
        WRITE(6,*) '   0 = Infinity norm (MAX(ABS(X(i)))'
        WRITE(6,*) '   1 =  SUM( ABS( X(i) ) )'
        WRITE(6,*) '   2 =( SUM( ABS( X(i) )**2) )**(1/2)'
        INORM  = ASK_I('Enter the norm number (0 | 1 | 2).........')
        NOFMSP = ASK  ('Print FMS output during iterations?')
        NOFMSP = (.NOT.NOFMSP)
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSIGT ('MEMPTR, IMD_PTR)
        CALL FMSIGT ('MEMPTR, RMD_PTR)
        CALL FMSIGT ('MEMPTR, CMD_PTR)
C
C       Get a temporary vector for generating columns and RHS's:
        NMAX = MAX0(N1,N2)
        CALL FMSCMG (CMD, L_TEMP, NMAX)
C
C (2)   Open FMS files:
        CALL CNDI  (LOWEQ, N1,           'LU_A11', LU_A11)
```

```
         CALL FMSOV  (N1, IDTYPE, N2  , 'LU_A12', LU_A12)
         CALL FMSOV  (N1, IDTYPE, N2  , 'LU_A21', LU_A21)
         CALL CNDI  (LOWEQ, N2,           'LU_A22', LU_A22)
         CALL FMSOV  (N1, IDTYPE, NRHS, 'LU_B1' , LU_B1 )
         CALL FMSOV  (N1, IDTYPE, NRHS, 'LU_X1' , LU_X1 )
         CALL FMSOV  (N2, IDTYPE, NRHS, 'LU_B2' , LU_B2 )
         CALL FMSOV  (N2, IDTYPE, NRHS, 'LU_X2' , LU_X2 )
C
C (3)    Write data to FMS files:
         NEWA11 = .TRUE.
         NEWA22 = .TRUE.
C
C        Branch back to here for refinement solutions:
   30    CONTINUE
C
C        Generate [A11]:
         IF(NEWA11) CALL A11GEN (CMD(L_TEMP), N1, N2,   LU_A11)
C
C        Generate [A12]:
         CALL A12GEN (CMD(L_TEMP), N1, N2,   LU_A12)
C
C        Generate [A21]:
         CALL A21GEN (CMD(L_TEMP), N1, N2,   LU_A21)
C
C        Generate [A22]:
         IF(NEWA22) CALL A22GEN (CMD(L_TEMP), N2,       LU_A22)
C
C        Generate {B1}:
         CALL B1GEN  (CMD(L_TEMP), N1, NRHS, LU_B1 )
C
C        Generate {B2}:
         CALL B2GEN  (CMD(L_TEMP), N2, NRHS, LU_B2 )
C
C (4)    Perform matrix algebra:
         IF( (METHOD .EQ. METHOD_JACOBI) .OR.
     1       (METHOD .EQ. METHOD_GAUSS_SEIDEL) ) THEN
            CALL J_GS
     1     (LU_A11, LU_A11,
     2      LU_A12,
```

```
     3      LU_A21,
     4      LU_A22, LU_A22,
     5      LU_B1,  LU_X1,
     6      LU_B2,  LU_X2,
     7      NEWA11, NEWA22,
     8      INORM, MAXITR, ERRMAX, METHOD, NOFMSP)
C
        ELSE IF(METHOD .EQ. METHOD_JACOBI_INCREMENT) THEN
           CALL JACOBI2
     1     (LU_A11, LU_A11,
     2      LU_A12,
     3      LU_A21,
     4      LU_A22, LU_A22,
     5      LU_B1,  LU_X1,
     6      LU_B2,  LU_X2,
     7      NEWA11, NEWA22,
     8      INORM, MAXITR, ERRMAX, NOFMSP)
        END IF
C
C (5)   Read data from FMS files:
C       Check the answer:
        ERROR  = R_ZERO
C
C       Check {X1}:
        CALL FMSGET (LU_X1, 0, 0, 0, 0, CMD, 0)
        DO IRHS = 1,NRHS
           CALL FMSGET (LU_X1, N1, 1, 1, IRHS, CMD(L_TEMP), N1)
           DO I = 0,(N1-1)
              EI = ABS(CMD(L_TEMP+I) - C_ONE)
              IF(EI .GT. ERROR) ERROR = EI
           END DO
        END DO
           CALL FMSGET (LU_X1, 0, 0, N1+1, 1, CMD, 0)
C
C       Check {X2}:
           CALL FMSGET (LU_X2, 0, 0, 0, 0, CMD, 0)
        DO IRHS = 1,NRHS
           CALL FMSGET (LU_X2, N2, 1, 1, IRHS, CMD(L_TEMP), N2)
           DO I = 0,(N2-1)
```

```
               EI = ABS(CMD(L_TEMP+I) - C_ONE)
               IF(EI .GT. ERROR) ERROR = EI
            END DO
         END DO
            CALL FMSGET (LU_X2, 0, 0, N2+1, 1, CMD, 0)
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C        Modify the matrix and solve it again:
         IF(.NOT. ASK('Do you want another solution with this matrix?'))
     1           GO TO 100
         NEWA11 = ASK('Do you want a new A11?')
         NEWA22 = ASK('Do you want a new A22?')
C
         IF( NEWA11 .OR.
     1      NEWA22) GO TO 30
C
C (6)    Close FMS files:
  100    CONTINUE
            CALL FMSCM (LU_A11)
            CALL FMSCV (LU_A12)
            CALL FMSCV (LU_A21)
            CALL FMSCM (LU_A22)
            CALL FMSCV (LU_B1 )
            CALL FMSCV (LU_X1 )
            CALL FMSCV (LU_B2 )
            CALL FMSCV (LU_X2 )
            CALL FMSCMR (CMD, L_TEMP, NMAX)
         IF(ASK('Do you want to solve another problem?')) GO TO 10
         CALL FMSPOP(MYNAME)
            CALL FMSEND
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
```

```
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        REAL*8 FUNCTION ASK_R(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_R
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE A11GEN (ACOL, N1, N2, LU_A11)
C=======================================================================
        COMPLEX*16 ACOL(N1)
        INTEGER    N1, N2
        INTEGER    LU_A11(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A11GEN')
        REAL*8     A11_Re, R_ZERO
        COMPLEX*16 C_ZERO, C_ONE, C_MONE
        DATA R_ZERO/0.0D0/
        DATA C_ZERO/( 0.0D0,0.0D0)/
```

```
      DATA C_MONE/(-1.0D0,0.0D0)/
      DATA C_ONE /( 1.0D0,0.0D0)/
C
      CALL FMSPSH(MYNAME)
      WRITE(6,2000)
C
C     Initialize FMSCOL
      CALL FMSCOL (0, ACOL, LU_A11)
C
C     Write first column
      DO 10 I = 2,N1
         ACOL(I) = C_MONE
   10 CONTINUE
      A11_Re  = DFLOAT(N1+N2)
      ACOL(1) = DCMPLX(A11_Re,R_ZERO)
      CALL FMSCOL (1, ACOL, LU_A11)
C
C     Write remaining columns:
      DO 20 I = 2,N1
         ACOL(I) = C_ZERO
   20 CONTINUE
      ACOL(1) = C_MONE
      DO 30 I = 2,N1
         ACOL(I) = C_ONE
         CALL FMSCOL (I, ACOL, LU_A11)
         ACOL(I) = C_ZERO
   30 CONTINUE
C
C     End FMSCOL
      CALL FMSCOL (N1+1, ACOL, LU_A11)
      CALL FMSPOP(MYNAME)
      RETURN
 2000 FORMAT (' Populating [A11]:')
      END
C=======================================================================
      SUBROUTINE A12GEN (ACOL, N1, N2, LU_A12)
C=======================================================================
      COMPLEX*16 ACOL(N1)
      INTEGER    N1, N2
```

```
      INTEGER    LU_A12(25)
      CHARACTER*6 MYNAME
      PARAMETER (MYNAME='A12GEN')
      COMPLEX*16 C_ZERO, C_MONE
      DATA C_ZERO/( 0.0D0,0.0D0)/
      DATA C_MONE/(-1.0D0,0.0D0)/
C
      CALL FMSPSH(MYNAME)
      WRITE(6,2000)
C
      DO 10 I=2,N1
         ACOL(I) = C_ZERO
   10 CONTINUE
      ACOL(1) = C_MONE
C
C     Initialize FMSPUT:
      CALL FMSPUT (LU_A12, 0, 0, 0, 0, ACOL, 0)
C
C     Write out columns of [A12]:
      DO 20 I = 1,N2
         CALL FMSPUT (LU_A12, N1, 1, 1, I, ACOL, N1)
   20 CONTINUE
C
C     End FMSPUT:
      CALL FMSPUT (LU_A12, 0, 0, N1+1, 0, ACOL, 0)
C
C     Set status:
      LU_A12(7) = 0
      CALL FMSPOP(MYNAME)
      RETURN
 2000 FORMAT (' Populating [A12]:')
      END
C=======================================================================
      SUBROUTINE A21GEN (AROW, N1, N2, LU_A21)
C=======================================================================
      COMPLEX*16 AROW(N1)
      INTEGER    N1, N2
      INTEGER    LU_A21(25)
      CHARACTER*6 MYNAME
```

```
        PARAMETER (MYNAME='A21GEN')
        COMPLEX*16 C_ZERO, C_MONE
        DATA C_ZERO/( 0.0D0,0.0D0)/
        DATA C_MONE/(-1.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
           AROW(I) = C_ZERO
   10   CONTINUE
        AROW(1) = C_MONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_A21, 0, 0, 0, 0, AROW, 0)
C
C       Write out rows of [A21], which is stored transposed:
        DO 20 I = 1,N2
           CALL FMSPUT (LU_A21, N1, 1, 1, I, AROW, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_A21, 0, 0, N1+1, 0, AROW, 0)
C
C       Set status:
        LU_A21(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating [A21]:')
        END
C=======================================================================
        SUBROUTINE A22GEN (ACOL, N2, LU_A22)
C=======================================================================
        COMPLEX*16 ACOL(N2)
        INTEGER    N2
        INTEGER    LU_A22(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A22GEN')
        COMPLEX*16 C_ZERO, C_ONE
```

```
        DATA C_ZERO/( 0.0D0,0.0D0)/
        DATA C_ONE /( 1.0D0,0.0D0)/
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
C       Initialize FMSCOL
        CALL FMSCOL (0, ACOL, LU_A22)
C
        DO 10 I = 1,N2
           ACOL(I) = C_ZERO
   10   CONTINUE
C
        DO 20 I = 1,N2
           ACOL(I) = C_ONE
           CALL FMSCOL (I, ACOL, LU_A22)
           ACOL(I) = C_ZERO
   20   CONTINUE
C
C       End FMSCOL
        CALL FMSCOL (N2+1, ACOL, LU_A22)
C
C       Set status:
        LU_A22(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating [A22]:')
        END
C=======================================================================
        SUBROUTINE B1GEN (RHS, N1, NRHS, LU_B1)
C=======================================================================
        COMPLEX*16 RHS(N1)
        INTEGER    N1, NRHS
        INTEGER    LU_B1(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B1GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA C_ZERO/(0.0D0,0.0D0)/
        DATA C ONE /(1.0D0,0.0D0)/
```

```
C
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
          RHS(I) = C_ZERO
   10   CONTINUE
        RHS(1) = C_ONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B1}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B1, N1, 1, 1, I, RHS, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, N1+1, 0, RHS, 0)
C
C       Set status:
        LU_B1(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating {B1}:')
        END
C=======================================================================
        SUBROUTINE B2GEN (RHS, N2, NRHS, LU_B2)
C=======================================================================
        COMPLEX*16 RHS(N2)
        INTEGER    N2, NRHS
        INTEGER    LU_B2(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B2GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA C_ZERO/(0.0D0,0.0D0)/
        DATA C_ONE /(1.0D0,0.0D0)/
C
```

```
        CALL FMSPSH(MYNAME)
        WRITE(6,2000)
C
        DO 10 I=1,N2
          RHS(I) = C_ZERO
   10   CONTINUE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B2, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B2}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B2, N2, 1, 1, I, RHS, N2)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_B2, 0, 0, N2+1, 0, RHS, 0)
C
C       Set status:
        LU_B2(7) = 0
        CALL FMSPOP(MYNAME)
        RETURN
 2000   FORMAT (' Populating {B2}:')
        END
C=======================================================================
        SUBROUTINE J_GS
     1  (LU_A11, LU_F11,
     2   LU_A12,
     2   LU_A21,
     4   LU_A22, LU_F22,
     3   LU_B1,  LU_X1,
     4   LU_B2,  LU_X2,
     5   NEWA11, NEWA22,
     6   INORM, MAXITR, ERRMAX, METHOD, NOFMSP)
C=======================================================================
C
C       DESCRIPTION:
C          This subroutine solves the linear system
C
```

```
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | | X1 |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | | X2 |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C          using block Jacobi or Gauss_Seidel iteration.
C
C          On input, the following matrix terms are specified:
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | |    |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | |    |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          The following intermediate work arrays are used:
C
C          +-    +    -+ +-      -+   +-  -+
C          |     |     | | X1_OLD |   |    |
C          +-----+-----+ +--------+ = +----+
C          |     |     | | X2_OLD |   |    |
C          +-    +    -+ +-      -+   +-  -+
C
C
C          On output, the following terms are returned:
C
C          +-    +    -+ +-  -+   +-  -+
C          | F11 |     | | X1 |   |    |
C          +-----+-----+ +----+ = +----+
C          |     | F22 | | X2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
C
C       You may overlay any of the corresponding input or output terms
C       to save storage by specifying the same actual parameter on the
C       subroutine call.
C
C       FORMAL PARAMETERS:
```

```
C          (R ) LU_A11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A11]
C
C          (R ) LU_F11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F11]
C                             May be the same as LU_A11.
C
C          (R ) LU_A12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A12]
C                             Data on this file is not changed.
C
C          (R ) LU_A21(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A21]
C                             Data on this file is not changed.
C
C          (R ) LU_A22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A22]
C
C          (R ) LU_F22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F22]
C
C          (R ) LU_B1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B1}
C
C          (R ) LU_X1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X1}.
C                             May be the same as LU_B1.
C
C          (R ) LU_B2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B2}
C
C          (R ) LU_X2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X2}.
C                             May be the same as LU_B2.
C
C          (R ) NEWA11      = LOGICAL
C                             Indicates that [A11] has changed.
C
C          (R ) NEWA22      = LOGICAL
```

```
C                           Indicates that [A22] has changed.
C
C          (R ) INORM       = Integer
C                           Norm for computing convergence:
C                           = 0, Infinity norm (max. abs. value)
C                           = 1, First norm
C                           = 2, Second norm
C
C          (R ) MAXITR      = Integer
C                           The maximum number of iterations to
C                           perform.
C
C          (R ) ERRMAX      = REAL*8
C                           = Maximum error for convergence
C
C          (R ) METHOD      = Integer
C                           = 1, Jacobi
C                           = 2, Gauss-Seidel
C
C          (R ) NOFMSP      = Logical
C                           Print FMS output in iteration loop.
C
C       FMSCOM PARAMETERS:
C          (RW) IACCOM      = Multiply accumulate flag
C          (R ) LUPR        = FORTRAN unit for printing
C
C       CALLED FROM:
C          EXAMPLE_17
C
C       SUBPROGRAMS USED:
C          CNDF
C          CNDS
C          FMSMM
C
C       ERROR CONDITIONS:
C          None.
C
C       HISTORY:
C
```

```
C       NOTES:
C
C-----------------------------------------------------------------------
C       FORMAL PARAMETERS
C-----------------------------------------------------------------------
        INTEGER    LU_A11(25)
        INTEGER    LU_F11(25)
C
        INTEGER    LU_A12(25)
C
        INTEGER    LU_A21(25)
C
        INTEGER    LU_A22(25)
        INTEGER    LU_F22(25)
C
        INTEGER    LU_B1 (25)
        INTEGER    LU_X1 (25)
C
        INTEGER    LU_B2 (25)
        INTEGER    LU_X2 (25)
C
        LOGICAL    NEWA11
        LOGICAL    NEWA22
        INTEGER    INORM
        INTEGER    MAXITR
        REAL*8     ERRMAX
        INTEGER    METHOD
        LOGICAL    NOFMSP
C-----------------------------------------------------------------------
C       LOCAL VARIABLES
C-----------------------------------------------------------------------
C
        CHARACTER*4 MYNAME
        PARAMETER (MYNAME='J_GS')
C
C       FMS Memory management:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD PTR, IMD)
```

```
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C
C     Data type = complex:
      INTEGER    IDTYPE
      PARAMETER (IDTYPE = 2)
C
C     Skip operations during solving (no):
      INTEGER    ISKIP
      PARAMETER (ISKIP=0)
C
C     Number of solution steps:
      INTEGER    NSTEPS
      PARAMETER (NSTEPS=11)
C
C     Method of solution:
      INTEGER    METHOD_JACOBI
      PARAMETER (METHOD_JACOBI=1)
      INTEGER    METHOD_GAUSS_SEIDEL
      PARAMETER (METHOD_GAUSS_SEIDEL=2)
C
C     Constants:
      COMPLEX*16 C_ONE(1)
      DATA       C_ONE  /( 1.0D0,0.0D0)/
      COMPLEX*16 C_MONE
      DATA       C_MONE /(-1.0D0,0.0D0)/
      REAL*8     R_ZERO
      DATA       R_ZERO /0.0D0/
      REAL*8     R_ONE
      DATA       R_ONE  /1.0D0/
      REAL*8     R_TWO
      DATA       R_TWO  /2.0D0/
      REAL*8     R_THREE
      DATA       R_THREE/3.0D0/
      REAL*8     R_EIGHT
      DATA       R_EIGHT/8.0D0/
C
```

```
C       FMS Parameters:
        INTEGER    LUPR
        INTEGER    IPRF
        INTEGER    IPRS
        INTEGER    IPRVV
C
C       Matrix dimensions:
        INTEGER    N1
        INTEGER    N2
        INTEGER    NRHS
        REAL*8     RN1
        REAL*8     RN2
        REAL*8     RNRHS
C
C       Convergence norms:
        REAL*8     R1
        REAL*8     R2
C
C       Temporary files:
        INTEGER    LU_X1_OLD(25)
        INTEGER    LU_X2_OLD(25)
        INTEGER     LUA0  (25)
        DATA        LUA0(1)/0/
        INTEGER     LUS0  (25)
        INTEGER     LUX0  (25)
C
C       CPU, Wall and I/O time:
        REAL*8     TCPU(NSTEPS)
        REAL*8     T1, TCPU_T1  ,TCPU_T2
        REAL*8     TWALL(NSTEPS)
        REAL*8     T2, TWALL_T1 ,TWALL_T2
        REAL*8     TIO(NSTEPS)
        REAL*8     T3, TIO_T1   ,TIO_T2
C
C       Floating point operations:
        REAL*8     OPS(NSTEPS), TOPS
        INTEGER    NCALLS(NSTEPS)
C-----------------------------------------------------------------------
C       FORTRAN PROCEDURE
```

```
C-----------------------------------------------------------------------
        CALL FMSPSH(MYNAME)
C
C       Get the FORTRAN unit for printng:
        CALL FMSIGT ('LUPR, LUPR)
C
C       Get the memory pointer for arrays IMD, RMD and CMD:
        CALL FMSIGT ('MEMPTR, IMD_PTR)
        CALL FMSIGT ('MEMPTR, RMD_PTR)
        CALL FMSIGT ('MEMPTR, CMD_PTR)
C
C       Determine the size of the matrices:
        N1    = LU_A11(8)
        N2    = LU_A22(8)
        NRHS  = LU_B1 (6)
        NUMEQ = N1 + N2
        RN1   = DFLOAT(N1)
        RN2   = DFLOAT(N2)
        RNRHS = DFLOAT(NRHS)
        WRITE(LUPR,2000) NUMEQ, N1, N2, NRHS, MAXITR, ERRMAX
C
C       Initialize operation counts and timers for each step:
        DO I=1,NSTEPS
           OPS   (I) = R_ZERO
           TCPU  (I) = R_ZERO
           TWALL (I) = R_ZERO
           TIO   (I) = R_ZERO
           NCALLS(I) = 0
        END DO
C
C       Open temporary files for previous solution:
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_X1_OLD' , LU_X1_OLD )
C
        CALL FMSTIM (TCPU_T1, TWALL_T1, TIO_T1)
        CALL  TIMER (0, R_ZERO, NCALLS, OPS, TCPU, TWALL, TIO)
C
C       Initialize {X1_OLD} and {X2_OLD}:
C
C  (1)  Factor [A11] into [F11]=[L11][U11]:
```

```
         ISTEP =  1
         IF(NEWA11) THEN
            WRITE(LUPR,2001)
            CALL CNDAF (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11,
     1                  LUX0, LUX0, 0)
            CALL TIMER (1, R_EIGHT*RN1*RN1*RN1/R_THREE,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
         END IF
C
C  (2)  Solve [F11]{X1_OLD} = {B1}:
         WRITE(LUPR,2002)
         CALL CNDS (LU_F11, LU_B1, LU_X1_OLD, NRHS, 0)
         CALL TIMER (2, R_EIGHT*RN1*RN1*RNRHS,
     1              NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (3)  Factor [A22] into [F22]=[L22][U22]:
         IF(NEWA22) THEN
            WRITE(LUPR,2003)
            CALL CNDAF (LU_A22, C_ONE, 1, LUS0, 0, LUA0, LU_F22,
     1                  LUX0, LUX0, 0)
            CALL TIMER (3, R_EIGHT*RN2*RN2*RN2/R_THREE,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
         END IF
C
         IF(METHOD .EQ. METHOD_JACOBI) THEN
C
C  (4)     Not performed:
C
C  (5)     Solve [F22]{X2_OLD}  = {B2}:
            WRITE(LUPR,2105)
            CALL CNDS (LU_F22, LU_B2,  LU_X2_OLD,  NRHS, 0)
            CALL TIMER (5, R_EIGHT*RN2*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
         ELSE IF(METHOD .EQ. METHOD_GAUSS_SEIDEL) THEN
C
C  (4)     Multiply {X2_OLD} = {B2} - [A21]{X1_OLD}:
            WRITE(LUPR,2204)
            CALL FMSMM (LU_X2_OLD, LU_B2, -1, 'T', LU_A21, LU_X1_OLD)
```

```
            LU_X1_OLD(7) = 0
            CALL TIMER (4, R_EIGHT*RN2*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (5)     Solve [F22]{X2_OLD}  = {X2_OLD}:
            WRITE(LUPR,2205)
            CALL CNDS (LU_F22, LU_X2_OLD,  LU_X2_OLD,  NRHS, 0)
            CALL TIMER (5, R_EIGHT*RN2*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
         END IF
C
C        Turn off FMS printing in loop, if required:
         IF(NOFMSP) THEN
         CALL FMSIGT ('IPRF, IPRF)
         CALL FMSIST ('IPRF, 0)
         CALL FMSIGT ('IPRS, IPRS)
         CALL FMSIST ('IPRS, 0)
         CALL FMSIGT ('IPRVV, IPRVV)
         CALL FMSIST ('IPRVV, 0)
         END IF
C
C        Loop over iterations:
         DO 100 NITER = 1,MAXITR
C
C  (6)     Multiply {X1} = {B1} - [A12]{X2_OLD}:
            IF(.NOT.NOFMSP) WRITE(LUPR,2006)
            CALL FMSMM (LU_X1, LU_B1, -1, 'N', LU_A12, LU_X2_OLD)
            LU_X1(7) = 0
            CALL TIMER (6, R_EIGHT*RN1*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (7)     Solve [F11]{X1} = {X1}:
            IF(.NOT.NOFMSP) WRITE(LUPR,2007)
            CALL CNDS (LU_F11, LU_X1, LU_X1, NRHS, 0)
            CALL TIMER (7, R_EIGHT*RN1*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
            IF(METHOD .EQ. METHOD_JACOBI) THEN
```

```
C
C  (8)        Multiply {X2} = {B2} - [A21]{X1_OLD}:
             IF(.NOT.NOFMSP) WRITE(LUPR,2108)
             CALL FMSMM (LU_X2, LU_B2, -1, 'T', LU_A21, LU_X1_OLD)
C
           ELSE IF(METHOD .EQ. METHOD_GAUSS_SEIDEL) THEN
C
C  (8)        Multiply {X2} = {B2} - [A21]{X1}:
             IF(.NOT.NOFMSP) WRITE(LUPR,2208)
             CALL FMSMM (LU_X2, LU_B2, -1, 'T', LU_A21, LU_X1)
C
           END IF
           LU_X2(7) = 0
           CALL TIMER (8, R_EIGHT*RN2*RN1*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (9)     Solve [F22]{X2} = {X2}:
           IF(.NOT.NOFMSP) WRITE(LUPR,2009)
           CALL CNDS (LU_F22, LU_X2, LU_X2, NRHS, 0)
           CALL TIMER (9, R_EIGHT*RN2*RN2*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
C (10)     R1=|| ( {X1} - {X1_OLD} )/ {X1} ||; {X1}->{X1_OLD}:
           IF(.NOT.NOFMSP) WRITE(LUPR,2010)
           CALL FMSVAN (LU_X1_OLD, -1, LU_X1, INORM, R1, 3)
           IF(INORM .GT. 1) THEN
              R1 = R1/RN1
           ELSE IF(INORM .EQ. 2) THEN
              R1 = R1/SQRT(RN1)
           END IF
           CALL TIMER (10, R_TWO*RN1*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
C (11)     R2=|| ( {X2} - {X2_OLD} )/ {X2} ||; {X2}->{X2_OLD}:
           IF(.NOT.NOFMSP) WRITE(LUPR,2011)
           CALL FMSVAN (LU_X2_OLD, -1, LU_X2, INORM, R2, 3)
           IF(INORM .GT. 1) THEN
              R2 = R2/RN2
           ELSE IF(INORM .EQ. 2) THEN
```

```
               R2 = R2/SQRT(RN2)
            END IF
            CALL TIMER (11, R_TWO*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
            WRITE(LUPR,2050) NITER, R1, R2
C
            IF( ( R1 .LE. ERRMAX) .AND.
     1          ( R2 .LE. ERRMAX) ) GO TO 101
  100    CONTINUE
  101    CONTINUE
C
C       Close temporary files:
            CALL FMSCV (LU_X1_OLD)
            CALL FMSCV (LU_X2_OLD)
C
C       Print timings:
        WRITE(LUPR,2020)
        CALL FMSTIM (TCPU_T2, TWALL_T2, TIO_T2)
        TOPS = R_ZERO
        DO 200 ISTEP = 1,NSTEPS
           NCALL = NCALLS(ISTEP)
           IF(NCALL .EQ. 0) THEN
C
C             Operation skipped:
              WRITE(LUPR,2022) ISTEP
           ELSE
              TOPS = TOPS + OPS(ISTEP)
C
C             CPU time and Mflops:
              T1 = TCPU(ISTEP)
              IF(T1 .GT. R_ZERO) THEN
                 MF1   = ((1.0D-6)*OPS(ISTEP)/T1)
              ELSE
                 MF1   = 0
              END IF
C
C             Wall time, Mflops and ratio:
```

```
               T2 = TWALL(ISTEP)
               T3 = TIO(ISTEP)
               IF((T2 .GT. R_ZERO) .AND.
     1            (T2 .GT.  T3   ) ) THEN
                  MF2   = ((1.0D-6)*OPS(ISTEP)/T2)
                  RATIO = T1/(T2-T3)
               ELSE
                  MF2   = 0
                  RATIO = R_ONE
               END IF
               WRITE(LUPR,2021) ISTEP, NCALL, T1, T2, T3, MF1, MF2, RATIO
            END IF
  200    CONTINUE
         T1 = TCPU_T2  - TCPU_T1
         T2 = TWALL_T2 - TWALL_T1
         T3 = TIO_T2   - TIO_T1
         IF(T1 .GT. R_ZERO) THEN
            MF1   = ((1.0D-6)*TOPS/T1)
         ELSE
            MF1   = 0
         END IF
         IF(T2 .GT. R_ZERO) THEN
            MF2   = ((1.0D-6)*TOPS/T2)
            RATIO = T1/(T2-T3)
         ELSE
            MF2   = 0
            RATIO = R_ONE
         END IF
         IF(NOFMSP) THEN
C           Restore print settings:
         CALL FMSIST ('IPRF, IPRF)
         CALL FMSIST ('IPRS, IPRS)
         CALL FMSIST ('IPRVV, IPRVV)
         END IF
         WRITE(LUPR,2023) T1, T2, T3, MF1, MF2, RATIO
         CALL FMSPOP(MYNAME)
         RETURN
 2000    FORMAT (/
     1   ' Start of Subroutine J_GS',/
```

```
     2  ' Rank of full matrix.............=',I8/
     3  ' Rank of submatrix #1............=',I8/
     4  ' Rank of submatrix #2............=',I8/
     5  ' # of right-hand-sides...........=',I8/
     6  ' Maximum number of iterations...=',I8/
     7  ' Maximum error norm..............=',E8.1/)
  2001   FORMAT (/
     1  ' STEP  1. Factor   [A11] into [F11]'/
     2  ' ================================')
  2002   FORMAT (/
     1  ' STEP  2. Solve    [F11]{X1_OLD}  = {B1}'/
     2  ' ====================================')
  2003   FORMAT (/
     1  ' STEP  3. Factor   [A22] into [F22]'/
     2  ' ================================')
  2204   FORMAT (/
     1  ' STEP  4. Multiply {X2_OLD} = {B2} - [A21]{X1_OLD}'/
     2  ' ================================================')
  2105   FORMAT (/
     1  ' STEP  5. Solve    [F22]{X2_OLD}  = {B2}'/
     2  ' ====================================')
  2205   FORMAT (/
     1  ' STEP  5. Solve    [F22]{X2_OLD}  = {X2_OLD}'/
     2  ' ============================================')
  2006   FORMAT (/
     1  ' STEP  6. Multiply {X1} = {B1} - [A12]{X2_OLD}'/
     2  ' ============================================')
  2007   FORMAT (/
     1  ' STEP  7. Solve    [F11]{X1} = {X1}'/
     2  ' ================================')
  2108   FORMAT (/
     1  ' STEP  8. Multiply {X2} = {B2} - [A21]{X1_OLD}'/
     2  ' ============================================')
  2208   FORMAT (/
     1  ' STEP  8. Multiply {X2} = {B2} - [A21]{X1}'/
     2  ' ====================================')
  2009   FORMAT (/
     1  ' STEP  9. Solve    [F22]{X2} = {X2}'/
     2  ' ================================')
```

```
 2010   FORMAT (/
     1  ' STEP 10. |( {X1} - {X1_OLD} )/{X1}; {X1}->{X1_OLD}'/,
     2  ' ================================================')
 2011   FORMAT (/
     1  ' STEP 11. |( {X2} - {X2_OLD} )/{X2}; {X2}->{X2_OLD}'/,
     2  ' ================================================')
 2020   FORMAT (/
     1  ' Summary of Subroutine J_GS:'//
     2  '                <----------TIMES (Sec.)---------->',
     2  '    <--MFLOPS-->'/
     2  '  STEP   RUN          CPU         WALL          I/O',
     2  '     CPU    WALL',
     2  '    RATIO'/
     3  ' =====   ===    =========    =========    =========',
     3  '    ====    ====',
     3  '    =====')
 2021   FORMAT (2I6,3F13.3,2I8,F9.3)
 2022   FORMAT (I6,12X,'Skipped')
 2023   FORMAT (' TOTAL',6X,3F13.3,2I8,F9.3)
 2050   FORMAT (
     1  ' ITERATION=',I3,
     2  ', {DX1}NORM=',E12.3,
     3  ', {DX2}NORM=',E12.3)
        END
C=======================================================================
        SUBROUTINE JACOBI2
     1  (LU_A11, LU_F11,
     2   LU_A12,
     2   LU_A21,
     4   LU_A22, LU_F22,
     3   LU_B1,  LU_X1,
     4   LU_B2,  LU_X2,
     5   NEWA11, NEWA22,
     6   INORM, MAXITR, ERRMAX, NOFMSP)
C=======================================================================
C
C       DESCRIPTION:
C          This subroutine solves the linear system
C
```

```
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | | X1 |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | | X2 |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C          using an incremental form of block Jacobi iteration.
C
C
C          On input, the following matrix terms are specified:
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | |    |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | |    |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          The following intermediate work arrays are used:
C
C          +-    +    -+ +-      -+   +-  -+
C          |     |     | | DX1(2) |   |    |
C          +-----+-----+ +--------+ = +----+
C          |     |     | | DX2(2) |   |    |
C          +-    +    -+ +-      -+   +-  -+
C
C
C          On output, the following terms are returned:
C
C          +-    +    -+ +-  -+   +-  -+
C          | F11 |     | | X1 |   |    |
C          +-----+-----+ +----+ = +----+
C          |     | F22 | | X2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
C
C       You may overlay any of the corresponding input or output terms
C       to save storage by specifying the same actual parameter on the
C       subroutine call.
C
```

```
C       FORMAL PARAMETERS:
C          (R ) LU_A11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A11]
C
C          (R ) LU_F11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F11]
C                             May be the same as LU_A11.
C
C          (R ) LU_A12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A12]
C
C          (R ) LU_A21(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A21]
C                             Data on this file is not changed.
C
C          (R ) LU_A22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A22]
C
C          (R ) LU_F22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F22]
C
C          (R ) LU_B1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B1}
C
C          (R ) LU_X1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X1}.
C                             May be the same as LU_B1.
C
C          (R ) LU_B2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B2}
C
C          (R ) LU_X2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X2}.
C                             May be the same as LU_B2.
C
C          (R ) NEWA11      = LOGICAL
C                             Indicates that [A11] has changed.
C
C          (R ) NEWA22      = LOGICAL
```

```
C                           Indicated that [A22] has changed.
C
C          (R ) INORM       = Integer
C                           Norm for computing convergence:
C                           = 0, Infinity norm (max. abs. value)
C                           = 1, First norm
C                           = 2, Second norm
C
C          (R ) MAXITR      = Integer
C                           The maximum number of iterations to
C                           perform.
C
C          (R ) ERRMAX      = REAL*8
C                           = Maximum error for convergence
C
C          (R ) NOFMSP      = Logical
C                           Print FMS output in iteration loop.
C
C       FMSCOM PARAMETERS:
C          (RW) IACCOM      = Multiply accumulate flag
C          (R ) LUPR        = FORTRAN unit for printing
C
C       CALLED FROM:
C          EXAMPLE_17
C
C       SUBPROGRAMS USED:
C          CNDF
C          CNDS
C          FMSMM
C
C       ERROR CONDITIONS:
C          None.
C
C       HISTORY:
C
C       NOTES:
C
C       In this iterative approach, let {X1} and {X2} represent the
```

```
C       exact solution and {DX1} and {DX2} the error.
C
C       Expanding the partitioned system gives:
C
C       [A11]({X1}+{DX1}) + [A12]({X2}+{DX2}) = {B1}      (1)
C       [A21]({X1}+{DX1}) + [A22]({X2}+{DX2}) = {B2}      (2)
C
C       or
C
C       [A11]{X1} + [A11]{DX1} + [A12]{X2} + [A12]{DX2} = {B1}  (3)
C       [A21]{X1} + [A21]{DX1} + [A22]{X2} + [A22]{DX2} = {B2}  (4)
C
C       or
C
C       ([A11]{X1}+[A12]{X2}-{B1}) + [A11]{DX1} = -[A12]{DX2}   (5)
C       ([A21]{X1}+[A22]{X2}-{B2}) + [A22]{DX2} = -[A21]{DX1}   (6)
C
C       Noting that the first term in (5) and (6) is zero gives
C
C       [A11]{DX1} = -[A12]{DX2}     (7)
C       [A22]{DX2} = -[A21]{DX1}     (8)
C
C       which may be used to compute improved values of the error
C       {DX1} and {DX2}.
C-----------------------------------------------------------------------
C       FORMAL PARAMETERS
C-----------------------------------------------------------------------
        INTEGER    LU_A11(25)
        INTEGER    LU_F11(25)
C
        INTEGER    LU_A12(25)
C
        INTEGER    LU_A21(25)
C
        INTEGER    LU_A22(25)
        INTEGER    LU_F22(25)
C
        INTEGER    LU_B1 (25)
        INTEGER    LU_X1 (25)
```

```
C
        INTEGER    LU_B2 (25)
        INTEGER    LU_X2 (25)
C
        LOGICAL    NEWA11
        LOGICAL    NEWA22
        INTEGER    INORM
        INTEGER    MAXITR
        REAL*8     ERRMAX
        LOGICAL    NOFMSP
C-----------------------------------------------------------------------
C       LOCAL VARIABLES
C-----------------------------------------------------------------------
C
        CHARACTER*7 MYNAME
        PARAMETER (MYNAME='JACOBI2')
C
C       FMS Memory management:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Data type = complex:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE = 2)
C
C       Skip operations during solving (no):
        INTEGER    ISKIP
        PARAMETER (ISKIP=0)
C
C       Number of solution steps:
        INTEGER    NSTEPS
        PARAMETER (NSTEPS=10)
C
C       Constants:
        COMPLEX*16 C_ONE(1)
```

```
      DATA       C_ONE  /( 1.0D0,0.0D0)/
      COMPLEX*16 C_MONE
      DATA       C_MONE /(-1.0D0,0.0D0)/
      REAL*8     R_ZERO
      DATA       R_ZERO / 0.0D0/
      REAL*8     R_ONE
      DATA       R_ONE  / 1.0D0/
      REAL*8     R_TWO
      DATA       R_TWO  / 2.0D0/
      REAL*8     R_THREE
      DATA       R_THREE/ 3.0D0/
      REAL*8     R_EIGHT
      DATA       R_EIGHT/ 8.0D0/
C
C     FMS Parameters:
      INTEGER    LUPR
      INTEGER    IPRF
      INTEGER    IPRS
      INTEGER    IPRVV
C
C     Matrix dimensions:
      INTEGER    N1
      INTEGER    N2
      INTEGER    NRHS
      REAL*8     RN1
      REAL*8     RN2
      REAL*8     RNRHS
C
C     Convergence ratios:
      REAL*8     R1
      REAL*8     R2
C
C     Temporary files:
      INTEGER    LU_DX1(25,2), LU_DX1_SAVE(25)
      INTEGER    LU_DX2(25,2), LU_DX2_SAVE(25)
      INTEGER     LUA0  (25)
      DATA        LUA0(1)/0/
      INTEGER     LUS0  (25)
      INTEGER     LUX0  (25)
```

```
      INTEGER     LUD   (25)
C
C       CPU, Wall and I/O time:
      REAL*8      TCPU(NSTEPS)
      REAL*8      T1, TCPU_T1  ,TCPU_T2
      REAL*8      TWALL(NSTEPS)
      REAL*8      T2, TWALL_T1 ,TWALL_T2
      REAL*8      TIO(NSTEPS)
      REAL*8      T3, TIO_T1   ,TIO_T2
C
C       Floating point operations:
      REAL*8      OPS(NSTEPS), TOPS
      INTEGER     NCALLS(NSTEPS)
C
C       Sign for adding solution:
      INTEGER     ISIGN
C-----------------------------------------------------------------------
C       FORTRAN PROCEDURE
C-----------------------------------------------------------------------
      CALL FMSPSH(MYNAME)
C
C       Get the FORTRAN unit for printng:
      CALL FMSIGT ('LUPR, LUPR)
C
C       Get the memory pointer for arrays IMD, RMD and CMD:
      CALL FMSIGT ('MEMPTR, IMD_PTR)
      CALL FMSIGT ('MEMPTR, RMD_PTR)
      CALL FMSIGT ('MEMPTR, CMD_PTR)
C
C       Determine the size of the matrices:
      N1    = LU_A11(8)
      N2    = LU_A22(8)
      NRHS  = LU_B1 (6)
      NUMEQ = N1 + N2
      RN1   = DFLOAT(N1)
      RN2   = DFLOAT(N2)
      RNRHS = DFLOAT(NRHS)
      WRITE(LUPR,2000) NUMEQ, N1, N2, NRHS, MAXITR, ERRMAX
C
```

```
C       Initialize operation counts and timers for each step:
        DO I=1,NSTEPS
           OPS   (I) = R_ZERO
           TCPU  (I) = R_ZERO
           TWALL (I) = R_ZERO
           TIO   (I) = R_ZERO
           NCALLS(I) = 0
        END DO
C
C       Open temporary files for residual vectors:
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_DX1_1' , LU_DX1(1,1))
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_DX1_2' , LU_DX1(1,2))
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_DX2_1' , LU_DX2(1,1))
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_DX2_2' , LU_DX2(1,2))
        LUD(1) = 0
C
        CALL FMSTIM (TCPU_T1, TWALL_T1, TIO_T1)
        CALL TIMER  (0, R_ZERO, NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (1)  Factor [A11] into [F11]=[L11][U11]:
        IF(NEWA11) THEN
           WRITE(LUPR,2001)
           CALL CNDAF (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11,
     1                 LUX0, LUX0, 0)
           CALL TIMER (1, R_EIGHT*RN1*RN1*RN1/R_THREE,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
        END IF
C
C  (2)  Solve [F11][X1] = [B1]:
        WRITE(LUPR,2002)
        CALL CNDS (LU_F11, LU_B1, LU_X1, NRHS,   0)
        CALL TIMER (2, R_EIGHT*RN1*RN1*RNRHS,
     1              NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (3)  Factor [A22] into [F22]=[L22][U22]:
        IF(NEWA22) THEN
           WRITE(LUPR,2003)
           CALL CNDAF (LU_A22, C_ONE, 1, LUS0, 0, LUA0, LU_F22,
     1                 LUX0, LUX0, 0)
```

```
            CALL TIMER (3, R_EIGHT*RN2*RN2*RN2/R_THREE,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
         END IF
C
C  (4)  Solve [F22]{X2}  = {B2}:
         WRITE(LUPR,2004)
         CALL CNDS (LU_F22, LU_B2,  LU_X2,  NRHS, 0)
         CALL TIMER (4, R_EIGHT*RN2*RN2*RNRHS,
     1              NCALLS, OPS, TCPU, TWALL, TIO)
C
C        On the first pass, use {X1} and {X2} for {DX1}(KOLD) and
C        {DX2}(KOLD):
         K     = 2
         DO I=1,25
            LU_DX1_SAVE(I) = LU_DX1(I,K)
            LU_DX2_SAVE(I) = LU_DX2(I,K)
            LU_DX1(I,K)    = LU_X1(I)
            LU_DX2(I,K)    = LU_X2(I)
         END DO
C
C        Turn off FMS printing in loop, if required:
         IF(NOFMSP) THEN
         CALL FMSIGT ('IPRF, IPRF)
         CALL FMSIST ('IPRF, 0)
         CALL FMSIGT ('IPRS, IPRS)
         CALL FMSIST ('IPRS, 0)
         CALL FMSIGT ('IPRVV, IPRVV)
         CALL FMSIST ('IPRVV, 0)
         END IF
C
C        Loop over iterations:
         ISIGN = 1
         DO 100 NITER = 1, MAXITR
            KOLD  = K
            K     = 3 - K
            ISIGN = -ISIGN
C
C  (5)     Multiply {DX1}(K) = [A12]{DX2}(KOLD}:
```

```
            IF(.NOT.NOFMSP) WRITE(LUPR,2005) K, KOLD
            CALL FMSMM (LU_DX1(1,K), LUD, 0, 'N', LU_A12,
     1                  LU_DX2(1,KOLD))
            LU_DX1(7,K) = 0
            CALL TIMER (5, R_EIGHT*RN1*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (6)     Solve [F11]{DX1}(K) = {DX1}(K):
            IF(.NOT.NOFMSP) WRITE(LUPR,2006) K, K
            CALL CNDS (LU_F11, LU_DX1(1,K), LU_DX1(1,K), NRHS, 0)
            CALL TIMER (6, R_EIGHT*RN1*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (7)     Compute {X1} = {X1} + (ISIGN)*{DX1} and compute norm R1:
            IF(.NOT.NOFMSP) WRITE(LUPR,2007) K
            CALL FMSVAN (LU_X1, ISIGN, LU_DX1(1,K), INORM, R1, 1)
            IF(INORM .GT. 1) THEN
               R1 = R1/RN1
            ELSE IF(INORM .EQ. 2) THEN
               R1 = R1/SQRT(RN1)
            END IF
            CALL TIMER (7, R_TWO*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (8)     Multiply {DX2}(K) = [A21]{DX1}(KOLD):
            IF(.NOT.NOFMSP) WRITE(LUPR,2008) K, KOLD
            CALL FMSMM (LU_DX2(1,K), LUD, 0, 'T', LU_A21,
     1                  LU_DX1(1,KOLD))
            LU_DX2(7,K) = 0
            CALL TIMER (8, R_EIGHT*RN2*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (9)     Solve [F22]{DX2}(K) = {DX2}(K):
            IF(.NOT.NOFMSP) WRITE(LUPR,2009) K, K
            CALL CNDS (LU_F22, LU_DX2(1,K), LU_DX2(1,K), NRHS, 0)
            CALL TIMER (9, R_EIGHT*RN2*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C (10)     Compute {X2} = {X2} + (ISIGN)*{DX2}(K):
```

```
            IF(.NOT. NOFMSP) WRITE(LUPR,2010) K
            CALL FMSVAN (LU_X2, ISIGN, LU_DX2(1,K), INORM, R2, 1)
            IF(INORM .GT. 1) THEN
               R2 = R2/RN2
            ELSE IF(INORM .EQ. 2) THEN
               R2 = R2/SQRT(RN2)
            END IF
            CALL TIMER (10, R_TWO*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
            WRITE(LUPR,2050) NITER, R1, R2
            IF(NITER .EQ. 1) THEN
C              Redefine LU_DX1 and LU_DX2:
               DO I=1,25
                  LU_DX1(I,KOLD) = LU_DX1_SAVE(I)
                  LU_DX2(I,KOLD) = LU_DX2_SAVE(I)
               END DO
            END IF
C
C           Test for convergence:
            IF( (R1 .LT. ERRMAX) .AND.
     1          (R2 .LT. ERRMAX) ) GO TO 101
  100    CONTINUE
  101    CONTINUE
C
C        Close temporary files:
            CALL FMSCV (LU_DX1(1,1))
            CALL FMSCV (LU_DX1(1,2))
            CALL FMSCV (LU_DX2(1,1))
            CALL FMSCV (LU_DX2(1,2))
C
C        Print timings:
         WRITE(LUPR,2020)
         CALL FMSTIM (TCPU_T2, TWALL_T2, TIO_T2)
         TOPS = R_ZERO
         DO 200 ISTEP = 1,NSTEPS
            NCALL = NCALLS(ISTEP)
            IF(NCALL .EQ. 0) THEN
C
```

```
C            Operation skipped:
             WRITE(LUPR,2022) ISTEP
          ELSE
             TOPS = TOPS + OPS(ISTEP)
C
C            CPU time and Mflops:
             T1 = TCPU(ISTEP)
             IF(T1 .GT. R_ZERO) THEN
                MF1   = ((1.0D-6)*OPS(ISTEP)/T1)
             ELSE
                MF1   = 0
             END IF
C
C            Wall time, Mflops and ratio:
             T2 = TWALL(ISTEP)
             T3 = TIO(ISTEP)
             IF((T2 .GT. R_ZERO) .AND.
     1          (T2 .GT.  T3   ) ) THEN
                MF2   = ((1.0D-6)*OPS(ISTEP)/T2)
                RATIO = T1/(T2-T3)
             ELSE
                MF2   = 0
                RATIO = R_ONE
             END IF
             WRITE(LUPR,2021) ISTEP, NCALL, T1, T2, T3, MF1, MF2, RATIO
          END IF
  200   CONTINUE
        T1 = TCPU_T2  - TCPU_T1
        T2 = TWALL_T2 - TWALL_T1
        T3 = TIO_T2   - TIO_T1
        IF(T1 .GT. R_ZERO) THEN
           MF1   = ((1.0D-6)*TOPS/T1)
        ELSE
           MF1   = 0
        END IF
        IF(T2 .GT. R_ZERO) THEN
           MF2   = ((1.0D-6)*TOPS/T2)
           RATIO = T1/(T2-T3)
        ELSE
```

```
            MF2   = 0
            RATIO = R_ONE
         END IF
         IF(NOFMSP) THEN
C           Restore print settings:
         CALL FMSIST ('IPRF, IPRF)
         CALL FMSIST ('IPRS, IPRS)
         CALL FMSIST ('IPRVV, IPRVV)
         END IF
         WRITE(LUPR,2023) T1, T2, T3, MF1, MF2, RATIO
         CALL FMSPOP(MYNAME)
         RETURN
 2000    FORMAT (/
     1   ' Start of Subroutine JACOBI2',/
     2   ' Rank of full matrix.....=',I8/
     3   ' Rank of submatrix #1....=',I8/
     4   ' Rank of submatrix #2....=',I8/
     5   ' # of right-hand-sides...=',I8/
     6   ' Maximum number of iterations...=',I8/
     7   ' Maximum error norm......=',E8.1/)
 2001    FORMAT (/
     1   ' STEP  1. Factor   [A11] into [F11]'/
     2   ' ================================')
 2002    FORMAT (/
     1   ' STEP  2. Solve    [F11]{X1}  = {B1}'/
     2   ' ================================')
 2003    FORMAT (/
     1   ' STEP  3. Factor   [A22] into [F22]'/
     2   ' ================================')
 2004    FORMAT (/
     1   ' STEP  4. Solve    [F22]{X2}  = {B2}'/
     2   ' ================================')
 2005    FORMAT (/
     1   ' STEP  5. Multiply {DX1}(',i1,') = [A12]{DX2}(',i1,')'/
     2   ' ========================================')
 2006    FORMAT (/
     1   ' STEP  6. Solve    [F11]{DX1}(',I1,') = {DX1}(',I1,')'/
     2   ' ========================================')
 2007    FORMAT (/
```

```
     1  ' STEP  7. Add      {X1} = {X1} + (ISIGN)*{DX1}(',I1,')'/
     2  ' ============================================')
  2008   FORMAT (/
     1  ' STEP  8. Multiply {DX2}(',I1,') = [A21]{DX1}(',I1,')'/
     2  ' ========================================')
  2009   FORMAT (/
     1  ' STEP  9. Solve    [F22]{DX2}(',I1,') = {DX2}(',I1,')'/
     2  ' ========================================')
  2010   FORMAT (/
     1  ' STEP 10. Add      {X2} = {X2} + (ISIGN)*{DX2}(',I1,')'/
     2  ' ============================================')
  2020   FORMAT (/
     1  ' Summary of Subroutine JACOBI2:'//
     2  '                <----------TIMES (Sec.)---------->',
     2  '    <--MFLOPS-->'/
     2  '  STEP   RUN          CPU         WALL          I/O',
     2  '     CPU    WALL',
     2  '    RATIO'/
     3  ' =====   ===    =========    =========    =========',
     3  '    ====    ====',
     3  '    =====')
  2021   FORMAT (2I6,3F13.3,2I8,F9.3)
  2022   FORMAT (I6,12X,'Skipped')
  2023   FORMAT (' TOTAL',6X,3F13.3,2I8,F9.3)
  2050   FORMAT (
     1  ' ITERATION=',I3,
     2  ', {DX1}NORM=',E12.3,
     3  ', {DX2}NORM=',E12.3)
        END
C=======================================================================
        SUBROUTINE TIMER (ICALL, OPSINC,
     1   NCALLS, OPS, TCPU, TWALL, TIO)
C=======================================================================
        INTEGER ICALL
        REAL*8  OPSINC
        INTEGER NCALLS(*)
        REAL*8  OPS(*)
        REAL*8  TCPU(*)
        REAL*8  TWALL(*)
```

```
      REAL*8  TIO(*)
      COMMON /FMSTIMES/TCPU1, TWALL1, TIO1
      REAL*8  TCPU1, TWALL1, TIO1
      REAL*8  TCPU2, TWALL2, TIO2
      IF(ICALL .EQ. 0) THEN
         CALL FMSTIM (TCPU1, TWALL1, TIO1)
      ELSE
         CALL FMSTIM (TCPU2, TWALL2, TIO2)
         NCALLS(ICALL) = NCALLS(ICALL) + 1
         OPS   (ICALL) = OPS   (ICALL) + OPSINC
         TCPU  (ICALL) = TCPU  (ICALL) + (TCPU2  - TCPU1 )
         TWALL (ICALL) = TWALL (ICALL) + (TWALL2 - TWALL1)
         TIO   (ICALL) = TIO   (ICALL) + (TIO2   - TIO1  )
         TCPU1  = TCPU2
         TWALL1 = TWALL2
         TIO1   = TIO2
      END IF
      RETURN
      END
```

Copyright © Multipath Corporation

```
C       T E S T    J A C O B I   A N D   G A U S S - S E I D E L
C
C       This program tests subroutines J_GS and Jacobi2, which
C       illustrate how the FMS subroutines may be used for a block
C       iterative solution.
C
C       The test matrix is of the form
C
C              [A]        {X}  =  {B}
C       +-            -+ +   +   +   +
C       | N -1 -1 -1 -1| | 1 |   | 1 |
C       |-1  1  0  0  0| | 1 |   | 0 |
C       |-1  0  1  0  0| | 1 | = | 0 |
C       |-1  0  0  1  0| | 1 |   | 0 |
C       |-1  0  0  0  1| | 1 |   | 0 |
C       +-            -+ +   +   +   +
C
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_17')
C
C       FMS matrix and vector file attributes:
        INTEGER     LU_A11(25)
        INTEGER     LU_A12(25)
        INTEGER     LU_A21(25)
        INTEGER     LU_A22(25)
        INTEGER     LU_B1 (25), LU_X1(25)
        INTEGER     LU_B2 (25), LU_X2(25)
C
C       FMS memory management requires the following arrays:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Data Type:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE=2)
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
        REAL*8  ASK_R
C
C       Solution methods:
        INTEGER    METHOD_JACOBI
        PARAMETER (METHOD_JACOBI           =1)
        INTEGER    METHOD_GAUSS_SEIDEL
        PARAMETER (METHOD_GAUSS_SEIDEL     =2)
        INTEGER    METHOD_JACOBI_INCREMENT
        PARAMETER (METHOD_JACOBI_INCREMENT =3)
C
C       LOCAL VARIABLES:
C       Number of equations in [A11]:
           INTEGER N1
C       Number of equations in [A22]:
           INTEGER N2
C       Maximum of N1 and N2:
           INTEGER NMAX
C       Number of solution vectors:
```

```
            INTEGER NRHS
C       Form a new [A11] matrix:
            LOGICAL NEWA11
C       Form a new [A22] matrix:
            LOGICAL NEWA22
C       Type of solution method:
            INTEGER METHOD
C       Print FMS output during interation:
            LOGICAL NOFMSP
C       Maximum number of iterations:
            INTEGER MAXITR
C       Maximum iteration error:
            REAL*8  ERRMAX
C       Error on current term:
            REAL*8  EI
C       Maximum overall error:
            REAL*8  ERROR
C       CMD Pointer to scratch vector:
            INTEGER L_TEMP
C       Profile vector for a full matrix:
            INTEGER    LOWEQ(1)
            DATA       LOWEQ/-1/
C
C       CONSTANTS:
            REAL*8     R_ZERO
            DATA R_ZERO/0.0D0/
            COMPLEX*16 C_ONE
            DATA C_ONE/(1.0D0,0.0D0)/
C
C (1)   Initialize FMS:
            CALL FMSINI
            CALL FMSPSH (MYNAME)
C
C       Use full column partial pivoting (SLAB format):
            CALL FMSIST ('MFMAT', 3)
C
C       Read in problem size parameters:
   10   CONTINUE
            N1     = ASK_I('Enter the number of equations in A11.....')
            N2     = ASK_I('Enter the number of equations in A22.....')
            NRHS   = ASK_I('Enter the number of solution vectors.....')
            MAXITR = ASK_I('Enter the maximum number of iterations...')
            ERRMAX = ASK_R('Enter the maximum error..................')
            WRITE(6,*) 'The following solution methods are available:'
            WRITE(6,*) '   1 = Jacobi,'
            WRITE(6,*) '   2 = Gauss-Seidel,'
            WRITE(6,*) '   3 = Incremental Jacobi.'
            METHOD = ASK_I('Enter solution method (1 | 2 | 3 ).......')
            WRITE(6,*) 'The following vector norms are available:'
            WRITE(6,*) '   0 = Infinity norm (MAX(ABS(X(i)))'
            WRITE(6,*) '   1 =  SUM( ABS( X(i) ) )'
            WRITE(6,*) '   2 =( SUM( ABS( X(i) )**2) )**(1/2)'
            INORM  = ASK_I('Enter the norm number (0 | 1 | 2)........')
            NOFMSP = ASK  ('Print FMS output during iterations?')
            NOFMSP = (.NOT.NOFMSP)
            WRITE (6,*) 'You may now alter any FMS parameter.'
            WRITE (6,*) 'When you are finished, type the letters RETURN'
            CALL FMSSET
            CALL FMSIGT ('MEMPTR', IMD_PTR)
            CALL FMSIGT ('MEMPTR', RMD_PTR)
            CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Get a temporary vector for generating columns and RHS's:
            NMAX = MAX0(N1,N2)
            CALL FMSCMG (CMD, L_TEMP, NMAX)
```

```
C
C (2)   Open FMS files:
        CALL CNDI  (LOWEQ, N1,           'LU_A11', LU_A11)
        CALL FMSOV (N1, IDTYPE, N2  , 'LU_A12', LU_A12)
        CALL FMSOV (N1, IDTYPE, N2  , 'LU_A21', LU_A21)
        CALL CNDI  (LOWEQ, N2,           'LU_A22', LU_A22)
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_B1' , LU_B1 )
        CALL FMSOV (N1, IDTYPE, NRHS, 'LU_X1' , LU_X1 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_B2' , LU_B2 )
        CALL FMSOV (N2, IDTYPE, NRHS, 'LU_X2' , LU_X2 )
C
C (3)   Write data to FMS files:
        NEWA11 = .TRUE.
        NEWA22 = .TRUE.
C
C       Branch back to here for refinement solutions:
   30   CONTINUE
C
C       Generate [A11]:
        IF(NEWA11) CALL A11GEN (CMD(L_TEMP), N1, N2,   LU_A11)
C
C       Generate [A12]:
        CALL A12GEN (CMD(L_TEMP), N1, N2,   LU_A12)
C
C       Generate [A21]:
        CALL A21GEN (CMD(L_TEMP), N1, N2,   LU_A21)
C
C       Generate [A22]:
        IF(NEWA22) CALL A22GEN (CMD(L_TEMP), N2,       LU_A22)
C
C       Generate {B1}:
        CALL B1GEN  (CMD(L_TEMP), N1, NRHS, LU_B1 )
C
C       Generate {B2}:
        CALL B2GEN  (CMD(L_TEMP), N2, NRHS, LU_B2 )
C
C (4)   Perform matrix algebra:
        IF( (METHOD .EQ. METHOD_JACOBI) .OR.
     1      (METHOD .EQ. METHOD_GAUSS_SEIDEL) ) THEN
           CALL J_GS
     1     (LU_A11, LU_A11,
     2      LU_A12,
     3      LU_A21,
     4      LU_A22, LU_A22,
     5      LU_B1,  LU_X1,
     6      LU_B2,  LU_X2,
     7      NEWA11, NEWA22,
     8      INORM, MAXITR, ERRMAX, METHOD, NOFMSP)
C
        ELSE IF(METHOD .EQ. METHOD_JACOBI_INCREMENT) THEN
           CALL JACOBI2
     1     (LU_A11, LU_A11,
     2      LU_A12,
     3      LU_A21,
     4      LU_A22, LU_A22,
     5      LU_B1,  LU_X1,
     6      LU_B2,  LU_X2,
     7      NEWA11, NEWA22,
     8      INORM, MAXITR, ERRMAX, NOFMSP)
        END IF
C
C (5)   Read data from FMS files:
C       Check the answer:
        ERROR  = R_ZERO
C
```

```
C       Check {X1}:
        CALL FMSGET (LU_X1, 0, 0, 0, 0, CMD, 0)
        DO IRHS = 1,NRHS
           CALL FMSGET (LU_X1, N1, 1, 1, IRHS, CMD(L_TEMP), N1)
           DO I = 0,(N1-1)
              EI = ABS(CMD(L_TEMP+I) - C_ONE)
              IF(EI .GT. ERROR) ERROR = EI
           END DO
        END DO
        CALL FMSGET (LU_X1, 0, 0, N1+1, 1, CMD, 0)
C
C       Check {X2}:
        CALL FMSGET (LU_X2, 0, 0, 0, 0, CMD, 0)
        DO IRHS = 1,NRHS
           CALL FMSGET (LU_X2, N2, 1, 1, IRHS, CMD(L_TEMP), N2)
           DO I = 0,(N2-1)
              EI = ABS(CMD(L_TEMP+I) - C_ONE)
              IF(EI .GT. ERROR) ERROR = EI
           END DO
        END DO
        CALL FMSGET (LU_X2, 0, 0, N2+1, 1, CMD, 0)
        WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C       Modify the matrix and solve it again:
        IF(.NOT. ASK('Do you want another solution with this matrix?'))
     1           GO TO 100
        NEWA11 = ASK('Do you want a new A11?')
        NEWA22 = ASK('Do you want a new A22?')
C
        IF( NEWA11 .OR.
     1      NEWA22) GO TO 30
C
C (6)   Close FMS files:
  100   CONTINUE
        CALL FMSCM (LU_A11)
        CALL FMSCV (LU_A12)
        CALL FMSCV (LU_A21)
        CALL FMSCM (LU_A22)
        CALL FMSCV (LU_B1 )
        CALL FMSCV (LU_X1 )
        CALL FMSCV (LU_B2 )
        CALL FMSCV (LU_X2 )
        CALL FMSCMR (CMD, L_TEMP, NMAX)
        IF(ASK('Do you want to solve another problem?')) GO TO 10
        CALL FMSPOP (MYNAME)
        CALL FMSEND
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
```

```
      INTEGER FUNCTION ASK_I(STRING)
C=====================================================================
      CHARACTER* (*) STRING
      WRITE(6,2000) STRING
      READ (5,*) ASK_I
      RETURN
 2000 FORMAT (1X,A,'>')
      END
C=====================================================================
      REAL*8 FUNCTION ASK_R(STRING)
C=====================================================================
      CHARACTER* (*) STRING
      WRITE(6,2000) STRING
      READ (5,*) ASK_R
      RETURN
 2000 FORMAT (1X,A,'>')
      END
C=====================================================================
      SUBROUTINE A11GEN (ACOL, N1, N2, LU_A11)
C=====================================================================
      COMPLEX*16 ACOL(N1)
      INTEGER    N1, N2
      INTEGER    LU_A11(25)
      CHARACTER*6 MYNAME
      PARAMETER (MYNAME='A11GEN')
      REAL*8     A11_Re, R_ZERO
      COMPLEX*16 C_ZERO, C_ONE, C_MONE
      DATA R_ZERO/0.0D0/
      DATA C_ZERO/( 0.0D0,0.0D0)/
      DATA C_MONE/(-1.0D0,0.0D0)/
      DATA C_ONE /( 1.0D0,0.0D0)/
C
      CALL FMSPSH (MYNAME)
      WRITE(6,2000)
C
C     Initialize FMSCOL
      CALL FMSCOL (0, ACOL, LU_A11)
C
C     Write first column
      DO 10 I = 2,N1
         ACOL(I) = C_MONE
   10 CONTINUE
      A11_Re  = DFLOAT(N1+N2)
      ACOL(1) = DCMPLX(A11_Re,R_ZERO)
      CALL FMSCOL (1, ACOL, LU_A11)
C
C     Write remaining columns:
      DO 20 I = 2,N1
         ACOL(I) = C_ZERO
   20 CONTINUE
      ACOL(1) = C_MONE
      DO 30 I = 2,N1
         ACOL(I) = C_ONE
         CALL FMSCOL (I, ACOL, LU_A11)
         ACOL(I) = C_ZERO
   30 CONTINUE
C
C     End FMSCOL
      CALL FMSCOL (N1+1, ACOL, LU_A11)
      CALL FMSPOP (MYNAME)
      RETURN
 2000 FORMAT (' Populating [A11]:')
      END
C=====================================================================
      SUBROUTINE A12GEN (ACOL, N1, N2, LU_A12)
```

```
C=======================================================================
        COMPLEX*16 ACOL(N1)
        INTEGER    N1, N2
        INTEGER    LU_A12(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A12GEN')
        COMPLEX*16 C_ZERO, C_MONE
        DATA C_ZERO/( 0.0D0,0.0D0)/
        DATA C_MONE/(-1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
           ACOL(I) = C_ZERO
   10   CONTINUE
        ACOL(1) = C_MONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_A12, 0, 0, 0, 0, ACOL, 0)
C
C       Write out columns of [A12]:
        DO 20 I = 1,N2
           CALL FMSPUT (LU_A12, N1, 1, 1, I, ACOL, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_A12, 0, 0, N1+1, 0, ACOL, 0)
C
C       Set status:
        LU_A12(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating [A12]:')
        END
C=======================================================================
        SUBROUTINE A21GEN (AROW, N1, N2, LU_A21)
C=======================================================================
        COMPLEX*16 AROW(N1)
        INTEGER    N1, N2
        INTEGER    LU_A21(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A21GEN')
        COMPLEX*16 C_ZERO, C_MONE
        DATA C_ZERO/( 0.0D0,0.0D0)/
        DATA C_MONE/(-1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
           AROW(I) = C_ZERO
   10   CONTINUE
        AROW(1) = C_MONE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_A21, 0, 0, 0, 0, AROW, 0)
C
C       Write out rows of [A21], which is stored transposed:
        DO 20 I = 1,N2
           CALL FMSPUT (LU_A21, N1, 1, 1, I, AROW, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU A21, 0, 0, N1+1, 0, AROW, 0)
```

```
C
C       Set status:
        LU_A21(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating [A21]:')
        END
C=======================================================================
        SUBROUTINE A22GEN (ACOL, N2, LU_A22)
C=======================================================================
        COMPLEX*16 ACOL(N2)
        INTEGER    N2
        INTEGER    LU_A22(25)
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='A22GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA C_ZERO/( 0.0D0,0.0D0)/
        DATA C_ONE /( 1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
C       Initialize FMSCOL
        CALL FMSCOL (0, ACOL, LU_A22)
C
        DO 10 I = 1,N2
           ACOL(I) = C_ZERO
   10   CONTINUE
C
        DO 20 I = 1,N2
           ACOL(I) = C_ONE
           CALL FMSCOL (I, ACOL, LU_A22)
           ACOL(I) = C_ZERO
   20   CONTINUE
C
C       End FMSCOL
        CALL FMSCOL (N2+1, ACOL, LU_A22)
C
C       Set status:
        LU_A22(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating [A22]:')
        END
C=======================================================================
        SUBROUTINE B1GEN (RHS, N1, NRHS, LU_B1)
C=======================================================================
        COMPLEX*16 RHS(N1)
        INTEGER    N1, NRHS
        INTEGER    LU_B1(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B1GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA C_ZERO/(0.0D0,0.0D0)/
        DATA C_ONE /(1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
        DO 10 I=2,N1
          RHS(I) = C_ZERO
   10   CONTINUE
        RHS(1) = C_ONE
C
```

```
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B1}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B1, N1, 1, 1, I, RHS, N1)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_B1, 0, 0, N1+1, 0, RHS, 0)
C
C       Set status:
        LU_B1(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating {B1}:')
        END
C=======================================================================
        SUBROUTINE B2GEN (RHS, N2, NRHS, LU_B2)
C=======================================================================
        COMPLEX*16 RHS(N2)
        INTEGER    N2, NRHS
        INTEGER    LU_B2(25)
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='B2GEN')
        COMPLEX*16 C_ZERO, C_ONE
        DATA C_ZERO/(0.0D0,0.0D0)/
        DATA C_ONE /(1.0D0,0.0D0)/
C
        CALL FMSPSH (MYNAME)
        WRITE(6,2000)
C
        DO 10 I=1,N2
          RHS(I) = C_ZERO
   10   CONTINUE
C
C       Initialize FMSPUT:
        CALL FMSPUT (LU_B2, 0, 0, 0, 0, RHS, 0)
C
C       Write out columns of {B2}:
        DO 20 I = 1,NRHS
           CALL FMSPUT (LU_B2, N2, 1, 1, I, RHS, N2)
   20   CONTINUE
C
C       End FMSPUT:
        CALL FMSPUT (LU_B2, 0, 0, N2+1, 0, RHS, 0)
C
C       Set status:
        LU_B2(7) = 0
        CALL FMSPOP (MYNAME)
        RETURN
 2000   FORMAT (' Populating {B2}:')
        END
C=======================================================================
        SUBROUTINE J_GS
     1  (LU_A11, LU_F11,
     2   LU_A12,
     2   LU_A21,
     4   LU_A22, LU_F22,
     3   LU_B1,  LU_X1,
     4   LU_B2,  LU_X2,
     5   NEWA11, NEWA22,
     6   INORM, MAXITR, ERRMAX, METHOD, NOFMSP)
C=======================================================================
C
```

```
C       DESCRIPTION:
C          This subroutine solves the linear system
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | | X1 |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | | X2 |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C          using block Jacobi or Gauss_Seidel iteration.
C
C          On input, the following matrix terms are specified:
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | |    |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | |    |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          The following intermediate work arrays are used:
C
C          +-    +    -+ +-      -+   +-  -+
C          |     |     | | X1_OLD |   |    |
C          +-----+-----+ +--------+ = +----+
C          |     |     | | X2_OLD |   |    |
C          +-    +    -+ +-      -+   +-  -+
C
C
C          On output, the following terms are returned:
C
C          +-    +    -+ +-  -+   +-  -+
C          | F11 |     | | X1 |   |    |
C          +-----+-----+ +----+ = +----+
C          |     | F22 | | X2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
C
C       You may overlay any of the corresponding input or output terms
C       to save storage by specifying the same actual parameter on the
C       subroutine call.
C
C       FORMAL PARAMETERS:
C          (R ) LU_A11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A11]
C
C          (R ) LU_F11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F11]
C                             May be the same as LU_A11.
C
C          (R ) LU_A12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A12]
C                             Data on this file is not changed.
C
C          (R ) LU_A21(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A21]
C                             Data on this file is not changed.
C
C          (R ) LU_A22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A22]
C
C          (R ) LU_F22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F22]
C
C          (R ) LU_B1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B1}
```

```
C
C          (R ) LU_X1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X1}.
C                             May be the same as LU_B1.
C
C          (R ) LU_B2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B2}
C
C          (R ) LU_X2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X2}.
C                             May be the same as LU_B2.
C
C          (R ) NEWA11      = LOGICAL
C                             Indicates that [A11] has changed.
C
C          (R ) NEWA22      = LOGICAL
C                             Indicates that [A22] has changed.
C
C          (R ) INORM       = Integer
C                             Norm for computing convergence:
C                           = 0, Infinity norm (max. abs. value)
C                           = 1, First norm
C                           = 2, Second norm
C
C          (R ) MAXITR      = Integer
C                             The maximum number of iterations to
C                             perform.
C
C          (R ) ERRMAX      = REAL*8
C                           = Maximum error for convergence
C
C          (R ) METHOD      = Integer
C                           = 1, Jacobi
C                           = 2, Gauss-Seidel
C
C          (R ) NOFMSP      = Logical
C                             Print FMS output in iteration loop.
C
C       FMSCOM PARAMETERS:
C          (RW) IACCOM      = Multiply accumulate flag
C          (R ) LUPR        = FORTRAN unit for printing
C
C       CALLED FROM:
C          EXAMPLE_17
C
C       SUBPROGRAMS USED:
C          CNDF
C          CNDS
C          FMSMM
C
C       ERROR CONDITIONS:
C          None.
C
C       HISTORY:
C
C       NOTES:
C
C-----------------------------------------------------------------------
C       FORMAL PARAMETERS
C-----------------------------------------------------------------------
        INTEGER    LU_A11(25)
        INTEGER    LU_F11(25)
C
        INTEGER    LU_A12(25)
```

```
C
        INTEGER    LU_A21(25)
C
        INTEGER    LU_A22(25)
        INTEGER    LU_F22(25)
C
        INTEGER    LU_B1 (25)
        INTEGER    LU_X1 (25)
C
        INTEGER    LU_B2 (25)
        INTEGER    LU_X2 (25)
C
        LOGICAL    NEWA11
        LOGICAL    NEWA22
        INTEGER    INORM
        INTEGER    MAXITR
        REAL*8     ERRMAX
        INTEGER    METHOD
        LOGICAL    NOFMSP
C-----------------------------------------------------------------------
C       LOCAL VARIABLES
C-----------------------------------------------------------------------
C
        CHARACTER*4 MYNAME
        PARAMETER (MYNAME='J_GS')
C
C       FMS Memory management:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Data type = complex:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE = 2)
C
C       Skip operations during solving (no):
        INTEGER    ISKIP
        PARAMETER (ISKIP=0)
C
C       Number of solution steps:
        INTEGER    NSTEPS
        PARAMETER (NSTEPS=11)
C
C       Method of solution:
        INTEGER    METHOD_JACOBI
        PARAMETER (METHOD_JACOBI=1)
        INTEGER    METHOD_GAUSS_SEIDEL
        PARAMETER (METHOD_GAUSS_SEIDEL=2)
C
C       Constants:
        COMPLEX*16 C_ONE(1)
        DATA       C_ONE  /( 1.0D0,0.0D0)/
        COMPLEX*16 C_MONE
        DATA       C_MONE /(-1.0D0,0.0D0)/
        REAL*8     R_ZERO
        DATA       R_ZERO /0.0D0/
        REAL*8     R_ONE
        DATA       R_ONE  /1.0D0/
        REAL*8     R_TWO
        DATA       R_TWO  /2.0D0/
        REAL*8     R_THREE
        DATA       R_THREE/3.0D0/
```

```
      REAL*8     R_EIGHT
      DATA       R_EIGHT/8.0D0/
C
C     FMS Parameters:
      INTEGER    LUPR
      INTEGER    IPRF
      INTEGER    IPRS
      INTEGER    IPRVV
C
C     Matrix dimensions:
      INTEGER    N1
      INTEGER    N2
      INTEGER    NRHS
      REAL*8     RN1
      REAL*8     RN2
      REAL*8     RNRHS
C
C     Convergence norms:
      REAL*8     R1
      REAL*8     R2
C
C     Temporary files:
      INTEGER    LU_X1_OLD(25)
      INTEGER    LU_X2_OLD(25)
      INTEGER     LUA0  (25)
      DATA        LUA0(1)/0/
      INTEGER     LUS0  (25)
      INTEGER     LUX0  (25)
C
C     CPU, Wall and I/O time:
      REAL*8     TCPU(NSTEPS)
      REAL*8     T1, TCPU_T1  ,TCPU_T2
      REAL*8     TWALL(NSTEPS)
      REAL*8     T2, TWALL_T1 ,TWALL_T2
      REAL*8     TIO(NSTEPS)
      REAL*8     T3, TIO_T1   ,TIO_T2
C
C     Floating point operations:
      REAL*8     OPS(NSTEPS), TOPS
      INTEGER    NCALLS(NSTEPS)
C-----------------------------------------------------------------------
C     FORTRAN PROCEDURE
C-----------------------------------------------------------------------
      CALL FMSPSH (MYNAME)
C
C     Get the FORTRAN unit for printng:
      CALL FMSIGT ('LUPR', LUPR)
C
C     Get the memory pointer for arrays IMD, RMD and CMD:
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C     Determine the size of the matrices:
      N1    = LU_A11(8)
      N2    = LU_A22(8)
      NRHS  = LU_B1 (6)
      NUMEQ = N1 + N2
      RN1   = DFLOAT(N1)
      RN2   = DFLOAT(N2)
      RNRHS = DFLOAT(NRHS)
      WRITE(LUPR,2000) NUMEQ, N1, N2, NRHS, MAXITR, ERRMAX
C
C     Initialize operation counts and timers for each step:
      DO I=1,NSTEPS
```

```
            OPS   (I) = R_ZERO
            TCPU  (I) = R_ZERO
            TWALL (I) = R_ZERO
            TIO   (I) = R_ZERO
            NCALLS(I) = 0
         END DO
C
C        Open temporary files for previous solution:
         CALL FMSOV (N1, IDTYPE, NRHS, 'LU_X1_OLD' , LU_X1_OLD )
         CALL FMSOV (N2, IDTYPE, NRHS, 'LU_X2_OLD' , LU_X2_OLD )
C
         CALL FMSTIM (TCPU_T1, TWALL_T1, TIO_T1)
         CALL  TIMER (0, R_ZERO, NCALLS, OPS, TCPU, TWALL, TIO)
C
C        Initialize {X1_OLD} and {X2_OLD}:
C
C  (1)   Factor [A11] into [F11]=[L11][U11]:
         ISTEP =  1
         IF(NEWA11) THEN
            WRITE(LUPR,2001)
            CALL CNDAF (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11,
     1                  LUX0, LUX0, 0)
            CALL TIMER (1, R_EIGHT*RN1*RN1*RN1/R_THREE,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
         END IF
C
C  (2)   Solve [F11]{X1_OLD} = {B1}:
         WRITE(LUPR,2002)
         CALL CNDS  (LU_F11, LU_B1, LU_X1_OLD, NRHS, 0)
         CALL TIMER (2, R_EIGHT*RN1*RN1*RNRHS,
     1               NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (3)   Factor [A22] into [F22]=[L22][U22]:
         IF(NEWA22) THEN
            WRITE(LUPR,2003)
            CALL CNDAF (LU_A22, C_ONE, 1, LUS0, 0, LUA0, LU_F22,
     1                  LUX0, LUX0, 0)
            CALL TIMER (3, R_EIGHT*RN2*RN2*RN2/R_THREE,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
         END IF
C
         IF(METHOD .EQ. METHOD_JACOBI) THEN
C
C  (4)      Not performed:
C
C  (5)      Solve [F22]{X2_OLD}  = {B2}:
            WRITE(LUPR,2105)
            CALL CNDS  (LU_F22, LU_B2,  LU_X2_OLD,  NRHS, 0)
            CALL TIMER (5, R_EIGHT*RN2*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
         ELSE IF(METHOD .EQ. METHOD_GAUSS_SEIDEL) THEN
C
C  (4)      Multiply {X2_OLD} = {B2} - [A21]{X1_OLD}:
            WRITE(LUPR,2204)
            CALL FMSMM (LU_X2_OLD, LU_B2, -1, 'T', LU_A21, LU_X1_OLD)
            LU_X1_OLD(7) = 0
            CALL TIMER (4, R_EIGHT*RN2*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (5)      Solve [F22]{X2_OLD}  = {X2_OLD}:
            WRITE(LUPR,2205)
            CALL CNDS  (LU_F22, LU_X2_OLD,  LU_X2_OLD,  NRHS, 0)
            CALL TIMER (5, R_EIGHT*RN2*RN2*RNRHS,
```

```
     1                NCALLS, OPS, TCPU, TWALL, TIO)
C
         END IF
C
C        Turn off FMS printing in loop, if required:
         IF(NOFMSP) THEN
            CALL FMSIGT ('IPRF' ,  IPRF)
            CALL FMSIST ('IPRF' ,     0)
            CALL FMSIGT ('IPRS' ,  IPRS)
            CALL FMSIST ('IPRS' ,     0)
            CALL FMSIGT ('IPRVV', IPRVV)
            CALL FMSIST ('IPRVV',     0)
         END IF
C
C        Loop over iterations:
         DO 100 NITER = 1,MAXITR
C
C  (6)      Multiply {X1} = {B1} - [A12]{X2_OLD}:
            IF(.NOT.NOFMSP) WRITE(LUPR,2006)
            CALL FMSMM (LU_X1, LU_B1, -1, 'N', LU_A12, LU_X2_OLD)
            LU_X1(7) = 0
            CALL TIMER (6, R_EIGHT*RN1*RN2*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (7)      Solve [F11]{X1} = {X1}:
            IF(.NOT.NOFMSP) WRITE(LUPR,2007)
            CALL CNDS  (LU_F11, LU_X1, LU_X1, NRHS, 0)
            CALL TIMER (7, R_EIGHT*RN1*RN1*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
            IF(METHOD .EQ. METHOD_JACOBI) THEN
C
C  (8)          Multiply {X2} = {B2} - [A21]{X1_OLD}:
               IF(.NOT.NOFMSP) WRITE(LUPR,2108)
               CALL FMSMM (LU_X2, LU_B2, -1, 'T', LU_A21, LU_X1_OLD)
C
            ELSE IF(METHOD .EQ. METHOD_GAUSS_SEIDEL) THEN
C
C  (8)          Multiply {X2} = {B2} - [A21]{X1}:
               IF(.NOT.NOFMSP) WRITE(LUPR,2208)
               CALL FMSMM (LU_X2, LU_B2, -1, 'T', LU_A21, LU_X1)
C
            END IF
            LU_X2(7) = 0
            CALL TIMER (8, R_EIGHT*RN2*RN1*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (9)      Solve [F22]{X2} = {X2}:
            IF(.NOT.NOFMSP) WRITE(LUPR,2009)
            CALL CNDS  (LU_F22, LU_X2, LU_X2, NRHS, 0)
            CALL TIMER (9, R_EIGHT*RN2*RN2*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
C (10)      R1=|| ( {X1} - {X1_OLD} )/ {X1} ||; {X1}->{X1_OLD}:
            IF(.NOT.NOFMSP) WRITE(LUPR,2010)
            CALL FMSVAN (LU_X1_OLD, -1, LU_X1, INORM, R1, 3)
            IF(INORM .GT. 1) THEN
               R1 = R1/RN1
            ELSE IF(INORM .EQ. 2) THEN
               R1 = R1/SQRT(RN1)
            END IF
            CALL TIMER (10, R_TWO*RN1*RNRHS,
     1                 NCALLS, OPS, TCPU, TWALL, TIO)
C
C (11)      R2=|| ( {X2} - {X2_OLD} )/ {X2} ||; {X2}->{X2_OLD}:
```

```
            IF(.NOT.NOFMSP) WRITE(LUPR,2011)
            CALL FMSVAN (LU_X2_OLD, -1, LU_X2, INORM, R2, 3)
            IF(INORM .GT. 1) THEN
               R2 = R2/RN2
            ELSE IF(INORM .EQ. 2) THEN
               R2 = R2/SQRT(RN2)
            END IF
            CALL TIMER (11, R_TWO*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
            WRITE(LUPR,2050) NITER, R1, R2
C
            IF( ( R1 .LE. ERRMAX) .AND.
     1          ( R2 .LE. ERRMAX) ) GO TO 101
  100   CONTINUE
  101   CONTINUE
C
C       Close temporary files:
        CALL FMSCV (LU_X1_OLD)
        CALL FMSCV (LU_X2_OLD)
C
C       Print timings:
        WRITE(LUPR,2020)
        CALL FMSTIM (TCPU_T2, TWALL_T2, TIO_T2)
        TOPS = R_ZERO
        DO 200 ISTEP = 1,NSTEPS
           NCALL = NCALLS(ISTEP)
           IF(NCALL .EQ. 0) THEN
C
C             Operation skipped:
              WRITE(LUPR,2022) ISTEP
           ELSE
              TOPS = TOPS + OPS(ISTEP)
C
C             CPU time and Mflops:
              T1 = TCPU(ISTEP)
              IF(T1 .GT. R_ZERO) THEN
                 MF1   = ((1.0D-6)*OPS(ISTEP)/T1)
              ELSE
                 MF1   = 0
              END IF
C
C             Wall time, Mflops and ratio:
              T2 = TWALL(ISTEP)
              T3 = TIO(ISTEP)
              IF((T2 .GT. R_ZERO) .AND.
     1           (T2 .GT.  T3   ) ) THEN
                 MF2   = ((1.0D-6)*OPS(ISTEP)/T2)
                 RATIO = T1/(T2-T3)
              ELSE
                 MF2   = 0
                 RATIO = R_ONE
              END IF
              WRITE(LUPR,2021) ISTEP, NCALL, T1, T2, T3, MF1, MF2, RATIO
           END IF
  200   CONTINUE
        T1 = TCPU_T2  - TCPU_T1
        T2 = TWALL_T2 - TWALL_T1
        T3 = TIO_T2   - TIO_T1
        IF(T1 .GT. R_ZERO) THEN
           MF1   = ((1.0D-6)*TOPS/T1)
        ELSE
           MF1   = 0
        END IF
        IF(T2 .GT. R ZERO) THEN
```

```
            MF2   = ((1.0D-6)*TOPS/T2)
            RATIO = T1/(T2-T3)
         ELSE
            MF2   = 0
            RATIO = R_ONE
         END IF
         IF(NOFMSP) THEN
C           Restore print settings:
            CALL FMSIST ('IPRF' ,  IPRF)
            CALL FMSIST ('IPRS' ,  IPRS)
            CALL FMSIST ('IPRVV', IPRVV)
         END IF
         WRITE(LUPR,2023) T1, T2, T3, MF1, MF2, RATIO
         CALL FMSPOP (MYNAME)
         RETURN
 2000    FORMAT (/
     1   ' Start of Subroutine J_GS',/
     2   ' Rank of full matrix.............=',I8/
     3   ' Rank of submatrix #1............=',I8/
     4   ' Rank of submatrix #2............=',I8/
     5   ' # of right-hand-sides...........=',I8/
     6   ' Maximum number of iterations...=',I8/
     7   ' Maximum error norm..............=',E8.1/)
 2001    FORMAT (/
     1   ' STEP  1. Factor   [A11] into [F11]'/
     2   ' ==================================')
 2002    FORMAT (/
     1   ' STEP  2. Solve    [F11]{X1_OLD}  = {B1}'/
     2   ' ========================================')
 2003    FORMAT (/
     1   ' STEP  3. Factor   [A22] into [F22]'/
     2   ' ==================================')
 2204    FORMAT (/
     1   ' STEP  4. Multiply {X2_OLD} = {B2} - [A21]{X1_OLD}'/
     2   ' ================================================')
 2105    FORMAT (/
     1   ' STEP  5. Solve    [F22]{X2_OLD}  = {B2}'/
     2   ' ====================================')
 2205    FORMAT (/
     1   ' STEP  5. Solve    [F22]{X2_OLD}  = {X2_OLD}'/
     2   ' ==========================================')
 2006    FORMAT (/
     1   ' STEP  6. Multiply {X1} = {B1} - [A12]{X2_OLD}'/
     2   ' ==============================================')
 2007    FORMAT (/
     1   ' STEP  7. Solve    [F11]{X1} = {X1}'/
     2   ' ==================================')
 2108    FORMAT (/
     1   ' STEP  8. Multiply {X2} = {B2} - [A21]{X1_OLD}'/
     2   ' ==============================================')
 2208    FORMAT (/
     1   ' STEP  8. Multiply {X2} = {B2} - [A21]{X1}'/
     2   ' ==========================================')
 2009    FORMAT (/
     1   ' STEP  9. Solve    [F22]{X2} = {X2}'/
     2   ' ==================================')
 2010    FORMAT (/
     1   ' STEP 10. |( {X1} - {X1_OLD} )/{X1}; {X1}->{X1_OLD}'/,
     2   ' ================================================')
 2011    FORMAT (/
     1   ' STEP 11. |( {X2} - {X2_OLD} )/{X2}; {X2}->{X2_OLD}'/,
     2   ' ================================================')
 2020    FORMAT (/
     1   ' Summary of Subroutine J_GS:'//
```

```
     2  '               <----------TIMES (Sec.)----------->',
     2  '    <--MFLOPS-->'/
     2  '  STEP   RUN          CPU         WALL          I/O',
     2  '     CPU    WALL',
     2  '    RATIO'/
     3  ' =====   ===    =========    =========    =========',
     3  '    ====    ====',
     3  '    =====')
 2021   FORMAT (2I6,3F13.3,2I8,F9.3)
 2022   FORMAT (I6,12X,'Skipped')
 2023   FORMAT (' TOTAL',6X,3F13.3,2I8,F9.3)
 2050   FORMAT (
     1  ' ITERATION=',I3,
     2  ', {DX1}NORM=',E12.3,
     3  ', {DX2}NORM=',E12.3)
        END
C=======================================================================
        SUBROUTINE JACOBI2
     1  (LU_A11, LU_F11,
     2   LU_A12,
     2   LU_A21,
     4   LU_A22, LU_F22,
     3   LU_B1,  LU_X1,
     4   LU_B2,  LU_X2,
     5   NEWA11, NEWA22,
     6   INORM, MAXITR, ERRMAX, NOFMSP)
C=======================================================================
C
C       DESCRIPTION:
C          This subroutine solves the linear system
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | | X1 |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | | X2 |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C          using an incremental form of block Jacobi iteration.
C
C
C          On input, the following matrix terms are specified:
C
C          +-    +    -+ +-  -+   +-  -+
C          | A11 | A12 | |    |   | B1 |
C          +-----+-----+ +----+ = +----+
C          | A21 | A22 | |    |   | B2 |
C          +-    +    -+ +-  -+   +-  -+
C
C
C          The following intermediate work arrays are used:
C
C          +-    +    -+ +-      -+   +-  -+
C          |     |     | | DX1(2) |   |    |
C          +-----+-----+ +--------+ = +----+
C          |     |     | | DX2(2) |   |    |
C          +-    +    -+ +-      -+   +-  -+
C
C
C          On output, the following terms are returned:
C
C          +-    +    -+ +-  -+   +-  -+
C          | F11 |     | | X1 |   |    |
C          +-----+-----+ +----+ = +----+
C          |     | F22 | | X2 |   |    |
C          +-    +    -+ +-  -+   +-  -+
```

```
C
C       You may overlay any of the corresponding input or output terms
C       to save storage by specifying the same actual parameter on the
C       subroutine call.
C
C       FORMAL PARAMETERS:
C          (R ) LU_A11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A11]
C
C          (R ) LU_F11(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F11]
C                             May be the same as LU_A11.
C
C          (R ) LU_A12(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A12]
C
C          (R ) LU_A21(25)  = INTEGER ARRAY
C                             FMS vector file attributes for [A21]
C                             Data on this file is not changed.
C
C          (R ) LU_A22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [A22]
C
C          (R ) LU_F22(25)  = INTEGER ARRAY
C                             FMS matrix file attributes for [F22]
C
C          (R ) LU_B1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B1}
C
C          (R ) LU_X1(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X1}.
C                             May be the same as LU_B1.
C
C          (R ) LU_B2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {B2}
C
C          (R ) LU_X2(25)   = INTEGER ARRAY
C                             FMS vector file attributes for {X2}.
C                             May be the same as LU_B2.
C
C          (R ) NEWA11      = LOGICAL
C                             Indicates that [A11] has changed.
C
C          (R ) NEWA22      = LOGICAL
C                             Indicated that [A22] has changed.
C
C          (R ) INORM       = Integer
C                             Norm for computing convergence:
C                           = 0, Infinity norm (max. abs. value)
C                           = 1, First norm
C                           = 2, Second norm
C
C          (R ) MAXITR      = Integer
C                             The maximum number of iterations to
C                             perform.
C
C          (R ) ERRMAX      = REAL*8
C                           = Maximum error for convergence
C
C          (R ) NOFMSP      = Logical
C                             Print FMS output in iteration loop.
C
C       FMSCOM PARAMETERS:
C          (RW) IACCOM      = Multiply accumulate flag
C          (R ) LUPR        = FORTRAN unit for printing
```

```
C
C       CALLED FROM:
C          EXAMPLE_17
C
C       SUBPROGRAMS USED:
C          CNDF
C          CNDS
C          FMSMM
C
C       ERROR CONDITIONS:
C          None.
C
C       HISTORY:
C
C       NOTES:
C
C       In this iterative approach, let {X1} and {X2} represent the
C       exact solution and {DX1} and {DX2} the error.
C
C       Expanding the partitioned system gives:
C
C       [A11]({X1}+{DX1}) + [A12]({X2}+{DX2}) = {B1}      (1)
C       [A21]({X1}+{DX1}) + [A22]({X2}+{DX2}) = {B2}      (2)
C
C       or
C
C       [A11]{X1} + [A11]{DX1} + [A12]{X2} + [A12]{DX2} = {B1}  (3)
C       [A21]{X1} + [A21]{DX1} + [A22]{X2} + [A22]{DX2} = {B2}  (4)
C
C       or
C
C       ([A11]{X1}+[A12]{X2}-{B1}) + [A11]{DX1} = -[A12]{DX2}   (5)
C       ([A21]{X1}+[A22]{X2}-{B2}) + [A22]{DX2} = -[A21]{DX1}   (6)
C
C       Noting that the first term in (5) and (6) is zero gives
C
C       [A11]{DX1} = -[A12]{DX2}     (7)
C       [A22]{DX2} = -[A21]{DX1}     (8)
C
C       which may be used to compute improved values of the error
C       {DX1} and {DX2}.
C-----------------------------------------------------------------------
C       FORMAL PARAMETERS
C-----------------------------------------------------------------------
        INTEGER    LU_A11(25)
        INTEGER    LU_F11(25)
C
        INTEGER    LU_A12(25)
C
        INTEGER    LU_A21(25)
C
        INTEGER    LU_A22(25)
        INTEGER    LU_F22(25)
C
        INTEGER    LU_B1 (25)
        INTEGER    LU_X1 (25)
C
        INTEGER    LU_B2 (25)
        INTEGER    LU_X2 (25)
C
        LOGICAL    NEWA11
        LOGICAL    NEWA22
        INTEGER    INORM
        INTEGER    MAXITR
```

```
        REAL*8     ERRMAX
        LOGICAL    NOFMSP
C-----------------------------------------------------------------------
C       LOCAL VARIABLES
C-----------------------------------------------------------------------
C
        CHARACTER*7 MYNAME
        PARAMETER (MYNAME='JACOBI2')
C
C       FMS Memory management:
        POINTER   (CMD_PTR, CMD)
        POINTER   (RMD_PTR, RMD)
        POINTER   (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Data type = complex:
        INTEGER    IDTYPE
        PARAMETER (IDTYPE = 2)
C
C       Skip operations during solving (no):
        INTEGER    ISKIP
        PARAMETER (ISKIP=0)
C
C       Number of solution steps:
        INTEGER    NSTEPS
        PARAMETER (NSTEPS=10)
C
C       Constants:
        COMPLEX*16 C_ONE(1)
        DATA       C_ONE  /( 1.0D0,0.0D0)/
        COMPLEX*16 C_MONE
        DATA       C_MONE /(-1.0D0,0.0D0)/
        REAL*8     R_ZERO
        DATA       R_ZERO / 0.0D0/
        REAL*8     R_ONE
        DATA       R_ONE  / 1.0D0/
        REAL*8     R_TWO
        DATA       R_TWO  / 2.0D0/
        REAL*8     R_THREE
        DATA       R_THREE/ 3.0D0/
        REAL*8     R_EIGHT
        DATA       R_EIGHT/ 8.0D0/
C
C       FMS Parameters:
        INTEGER    LUPR
        INTEGER    IPRF
        INTEGER    IPRS
        INTEGER    IPRVV
C
C       Matrix dimensions:
        INTEGER    N1
        INTEGER    N2
        INTEGER    NRHS
        REAL*8     RN1
        REAL*8     RN2
        REAL*8     RNRHS
C
C       Convergence ratios:
        REAL*8     R1
        REAL*8     R2
C
C       Temporary files:
        INTEGER    LU_DX1(25,2), LU_DX1_SAVE(25)
```

```
      INTEGER    LU_DX2(25,2), LU_DX2_SAVE(25)
      INTEGER     LUA0  (25)
      DATA        LUA0(1)/0/
      INTEGER     LUS0  (25)
      INTEGER     LUX0  (25)
      INTEGER     LUD   (25)
C
C     CPU, Wall and I/O time:
      REAL*8     TCPU(NSTEPS)
      REAL*8     T1, TCPU_T1  ,TCPU_T2
      REAL*8     TWALL(NSTEPS)
      REAL*8     T2, TWALL_T1 ,TWALL_T2
      REAL*8     TIO(NSTEPS)
      REAL*8     T3, TIO_T1   ,TIO_T2
C
C     Floating point operations:
      REAL*8     OPS(NSTEPS), TOPS
      INTEGER    NCALLS(NSTEPS)
C
C     Sign for adding solution:
      INTEGER    ISIGN
C-----------------------------------------------------------------------
C     FORTRAN PROCEDURE
C-----------------------------------------------------------------------
      CALL FMSPSH (MYNAME)
C
C     Get the FORTRAN unit for printng:
      CALL FMSIGT ('LUPR', LUPR)
C
C     Get the memory pointer for arrays IMD, RMD and CMD:
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C     Determine the size of the matrices:
      N1    = LU_A11(8)
      N2    = LU_A22(8)
      NRHS  = LU_B1 (6)
      NUMEQ = N1 + N2
      RN1   = DFLOAT(N1)
      RN2   = DFLOAT(N2)
      RNRHS = DFLOAT(NRHS)
      WRITE(LUPR,2000) NUMEQ, N1, N2, NRHS, MAXITR, ERRMAX
C
C     Initialize operation counts and timers for each step:
      DO I=1,NSTEPS
         OPS   (I) = R_ZERO
         TCPU  (I) = R_ZERO
         TWALL (I) = R_ZERO
         TIO   (I) = R_ZERO
         NCALLS(I) = 0
      END DO
C
C     Open temporary files for residual vectors:
      CALL FMSOV (N1, IDTYPE, NRHS, 'LU_DX1_1' , LU_DX1(1,1))
      CALL FMSOV (N1, IDTYPE, NRHS, 'LU_DX1_2' , LU_DX1(1,2))
      CALL FMSOV (N2, IDTYPE, NRHS, 'LU_DX2_1' , LU_DX2(1,1))
      CALL FMSOV (N2, IDTYPE, NRHS, 'LU_DX2_2' , LU_DX2(1,2))
      LUD(1) = 0
C
      CALL FMSTIM (TCPU_T1, TWALL_T1, TIO_T1)
      CALL TIMER  (0, R_ZERO, NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (1)  Factor [A11] into [F11]=[L11][U11]:
      IF(NEWA11) THEN
```

```
            WRITE(LUPR,2001)
            CALL CNDAF (LU_A11, C_ONE, 1, LUS0, 0, LUA0, LU_F11,
     1                  LUX0, LUX0, 0)
            CALL TIMER (1, R_EIGHT*RN1*RN1*RN1/R_THREE,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
         END IF
C
C  (2)  Solve [F11][X1] = [B1]:
         WRITE(LUPR,2002)
         CALL CNDS  (LU_F11, LU_B1, LU_X1, NRHS,   0)
         CALL TIMER (2, R_EIGHT*RN1*RN1*RNRHS,
     1               NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (3)  Factor [A22] into [F22]=[L22][U22]:
         IF(NEWA22) THEN
            WRITE(LUPR,2003)
            CALL CNDAF (LU_A22, C_ONE, 1, LUS0, 0, LUA0, LU_F22,
     1                  LUX0, LUX0, 0)
            CALL TIMER (3, R_EIGHT*RN2*RN2*RN2/R_THREE,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
         END IF
C
C  (4)  Solve [F22]{X2}  = {B2}:
         WRITE(LUPR,2004)
         CALL CNDS  (LU_F22, LU_B2,  LU_X2,  NRHS, 0)
         CALL TIMER (4, R_EIGHT*RN2*RN2*RNRHS,
     1               NCALLS, OPS, TCPU, TWALL, TIO)
C
C        On the first pass, use {X1} and {X2} for {DX1}(KOLD) and
C        {DX2}(KOLD):
         K     = 2
         DO I=1,25
            LU_DX1_SAVE(I) = LU_DX1(I,K)
            LU_DX2_SAVE(I) = LU_DX2(I,K)
            LU_DX1(I,K)    = LU_X1(I)
            LU_DX2(I,K)    = LU_X2(I)
         END DO
C
C        Turn off FMS printing in loop, if required:
         IF(NOFMSP) THEN
            CALL FMSIGT ('IPRF' ,  IPRF)
            CALL FMSIST ('IPRF' ,     0)
            CALL FMSIGT ('IPRS' ,  IPRS)
            CALL FMSIST ('IPRS' ,     0)
            CALL FMSIGT ('IPRVV', IPRVV)
            CALL FMSIST ('IPRVV',     0)
         END IF
C
C        Loop over iterations:
         ISIGN = 1
         DO 100 NITER = 1, MAXITR
            KOLD  = K
            K     = 3 - K
            ISIGN = -ISIGN
C
C  (5)      Multiply {DX1}(K) = [A12]{DX2}(KOLD):
            IF(.NOT.NOFMSP) WRITE(LUPR,2005) K, KOLD
            CALL FMSMM (LU_DX1(1,K), LUD, 0, 'N', LU_A12,
     1                  LU_DX2(1,KOLD))
            LU_DX1(7,K) = 0
            CALL TIMER (5, R_EIGHT*RN1*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (6)      Solve [F11]{DX1}(K) = {DX1}(K):
```

```
            IF(.NOT.NOFMSP) WRITE(LUPR,2006) K, K
            CALL CNDS  (LU_F11, LU_DX1(1,K), LU_DX1(1,K), NRHS, 0)
            CALL TIMER (6, R_EIGHT*RN1*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (7)     Compute {X1} = {X1} + (ISIGN)*{DX1} and compute norm R1:
            IF(.NOT.NOFMSP) WRITE(LUPR,2007) K
            CALL FMSVAN (LU_X1, ISIGN, LU_DX1(1,K), INORM, R1, 1)
            IF(INORM .GT. 1) THEN
               R1 = R1/RN1
            ELSE IF(INORM .EQ. 2) THEN
               R1 = R1/SQRT(RN1)
            END IF
            CALL TIMER (7, R_TWO*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (8)     Multiply {DX2}(K) = [A21]{DX1}(KOLD):
            IF(.NOT.NOFMSP) WRITE(LUPR,2008) K, KOLD
            CALL FMSMM (LU_DX2(1,K), LUD, 0, 'T', LU_A21,
     1                  LU_DX1(1,KOLD))
            LU_DX2(7,K) = 0
            CALL TIMER (8, R_EIGHT*RN2*RN1*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C  (9)     Solve [F22]{DX2}(K) = {DX2}(K):
            IF(.NOT.NOFMSP) WRITE(LUPR,2009) K, K
            CALL CNDS  (LU_F22, LU_DX2(1,K), LU_DX2(1,K), NRHS, 0)
            CALL TIMER (9, R_EIGHT*RN2*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
C (10)     Compute {X2} = {X2} + (ISIGN)*{DX2}(K):
            IF(.NOT. NOFMSP) WRITE(LUPR,2010) K
            CALL FMSVAN (LU_X2, ISIGN, LU_DX2(1,K), INORM, R2, 1)
            IF(INORM .GT. 1) THEN
               R2 = R2/RN2
            ELSE IF(INORM .EQ. 2) THEN
               R2 = R2/SQRT(RN2)
            END IF
            CALL TIMER (10, R_TWO*RN2*RNRHS,
     1                  NCALLS, OPS, TCPU, TWALL, TIO)
C
            WRITE(LUPR,2050) NITER, R1, R2
            IF(NITER .EQ. 1) THEN
C              Redefine LU_DX1 and LU_DX2:
               DO I=1,25
                  LU_DX1(I,KOLD) = LU_DX1_SAVE(I)
                  LU_DX2(I,KOLD) = LU_DX2_SAVE(I)
               END DO
            END IF
C
C           Test for convergence:
            IF( (R1 .LT. ERRMAX) .AND.
     1          (R2 .LT. ERRMAX) ) GO TO 101
  100    CONTINUE
  101    CONTINUE
C
C        Close temporary files:
         CALL FMSCV  (LU_DX1(1,1))
         CALL FMSCV  (LU_DX1(1,2))
         CALL FMSCV  (LU_DX2(1,1))
         CALL FMSCV  (LU_DX2(1,2))
C
C        Print timings:
         WRITE(LUPR,2020)
         CALL FMSTIM (TCPU_T2, TWALL_T2, TIO_T2)
```

```
         TOPS = R_ZERO
         DO 200 ISTEP = 1,NSTEPS
            NCALL = NCALLS(ISTEP)
            IF(NCALL .EQ. 0) THEN
C
C              Operation skipped:
               WRITE(LUPR,2022) ISTEP
            ELSE
               TOPS = TOPS + OPS(ISTEP)
C
C              CPU time and Mflops:
               T1 = TCPU(ISTEP)
               IF(T1 .GT. R_ZERO) THEN
                  MF1   = ((1.0D-6)*OPS(ISTEP)/T1)
               ELSE
                  MF1   = 0
               END IF
C
C              Wall time, Mflops and ratio:
               T2 = TWALL(ISTEP)
               T3 = TIO(ISTEP)
               IF((T2 .GT. R_ZERO) .AND.
     1            (T2 .GT.  T3   ) ) THEN
                  MF2   = ((1.0D-6)*OPS(ISTEP)/T2)
                  RATIO = T1/(T2-T3)
               ELSE
                  MF2   = 0
                  RATIO = R_ONE
               END IF
               WRITE(LUPR,2021) ISTEP, NCALL, T1, T2, T3, MF1, MF2, RATIO
            END IF
  200    CONTINUE
         T1 = TCPU_T2  - TCPU_T1
         T2 = TWALL_T2 - TWALL_T1
         T3 = TIO_T2   - TIO_T1
         IF(T1 .GT. R_ZERO) THEN
            MF1   = ((1.0D-6)*TOPS/T1)
         ELSE
            MF1   = 0
         END IF
         IF(T2 .GT. R_ZERO) THEN
            MF2   = ((1.0D-6)*TOPS/T2)
            RATIO = T1/(T2-T3)
         ELSE
            MF2   = 0
            RATIO = R_ONE
         END IF
         IF(NOFMSP) THEN
C           Restore print settings:
            CALL FMSIST ('IPRF' ,  IPRF)
            CALL FMSIST ('IPRS' ,  IPRS)
            CALL FMSIST ('IPRVV', IPRVV)
         END IF
         WRITE(LUPR,2023) T1, T2, T3, MF1, MF2, RATIO
         CALL FMSPOP (MYNAME)
         RETURN
 2000    FORMAT (/
     1   ' Start of Subroutine JACOBI2',/
     2   ' Rank of full matrix.............=',I8/
     3   ' Rank of submatrix #1............=',I8/
     4   ' Rank of submatrix #2............=',I8/
     5   ' # of right-hand-sides...........=',I8/
     6   ' Maximum number of iterations...=',I8/
     7   ' Maximum error norm..............=',E8.1/)
 2001    FORMAT (/
```

```
     1  ' STEP  1. Factor   [A11] into [F11]'/
     2  ' ==================================')
  2002   FORMAT (/
     1  ' STEP  2. Solve    [F11]{X1}  = {B1}'/
     2  ' ==================================')
  2003   FORMAT (/
     1  ' STEP  3. Factor   [A22] into [F22]'/
     2  ' ==================================')
  2004   FORMAT (/
     1  ' STEP  4. Solve    [F22]{X2}  = {B2}'/
     2  ' ==================================')
  2005   FORMAT (/
     1  ' STEP  5. Multiply {DX1}(',i1,') = [A12]{DX2}(',i1,')'/
     2  ' ========================================')
  2006   FORMAT (/
     1  ' STEP  6. Solve    [F11]{DX1}(',I1,') = {DX1}(',I1,')'/
     2  ' ========================================')
  2007   FORMAT (/
     1  ' STEP  7. Add      {X1} = {X1} + (ISIGN)*{DX1}(',I1,')'/
     2  ' ================================================')
  2008   FORMAT (/
     1  ' STEP  8. Multiply {DX2}(',I1,') = [A21]{DX1}(',I1,')'/
     2  ' ========================================')
  2009   FORMAT (/
     1  ' STEP  9. Solve    [F22]{DX2}(',I1,') = {DX2}(',I1,')'/
     2  ' ========================================')
  2010   FORMAT (/
     1  ' STEP 10. Add      {X2} = {X2} + (ISIGN)*{DX2}(',I1,')'/
     2  ' ================================================')
  2020   FORMAT (/
     1  ' Summary of Subroutine JACOBI2:'//
     2  '                <----------TIMES (Sec.)----------->',
     2  '    <--MFLOPS-->'/
     2  '  STEP   RUN          CPU         WALL          I/O',
     2  '     CPU    WALL',
     2  '    RATIO'/
     3  ' =====   ===    =========    =========    =========',
     3  '    ====    ====',
     3  '    =====')
  2021   FORMAT (2I6,3F13.3,2I8,F9.3)
  2022   FORMAT (I6,12X,'Skipped')
  2023   FORMAT (' TOTAL',6X,3F13.3,2I8,F9.3)
  2050   FORMAT (
     1  ' ITERATION=',I3,
     2  ', {DX1}NORM=',E12.3,
     3  ', {DX2}NORM=',E12.3)
         END
C=======================================================================
         SUBROUTINE TIMER (ICALL, OPSINC,
     1   NCALLS, OPS, TCPU, TWALL, TIO)
C=======================================================================
         INTEGER ICALL
         REAL*8  OPSINC
         INTEGER NCALLS(*)
         REAL*8  OPS(*)
         REAL*8  TCPU(*)
         REAL*8  TWALL(*)
         REAL*8  TIO(*)
         COMMON /FMSTIMES/TCPU1, TWALL1, TIO1
         REAL*8  TCPU1, TWALL1, TIO1
         REAL*8  TCPU2, TWALL2, TIO2
         IF(ICALL .EQ. 0) THEN
            CALL FMSTIM (TCPU1, TWALL1, TIO1)
         ELSE
```

```
      CALL FMSTIM (TCPU2, TWALL2, TIO2)
      NCALLS(ICALL) = NCALLS(ICALL) + 1
      OPS   (ICALL) = OPS   (ICALL) + OPSINC
      TCPU  (ICALL) = TCPU  (ICALL) + (TCPU2  - TCPU1 )
      TWALL (ICALL) = TWALL (ICALL) + (TWALL2 - TWALL1)
      TIO   (ICALL) = TIO   (ICALL) + (TIO2   - TIO1  )
      TCPU1  = TCPU2
      TWALL1 = TWALL2
      TIO1   = TIO2
   END IF
   RETURN
   END
```

Copyright © Multipath Corporation

FMS EXAMPLE 18. SOLVING WITH [A]T

This example computes **V = {C}T [Ainv]{B}** by two methods:

1. Solve **[A]{X}={B}** for **{X}**, then multiply **VCX={C}T{X}**,
2. Solve **[A]T{Y}={C}** for **{Y}**, then multiply **VYB={Y}T{B}**

The maximum error in **|{B}-[A]{X}|** and **|VCX-VYB|** is computed and printed.

The matrix **[A]** is nonsymmetric. The data may be real or complex.

The following input may be specified:

- ***FMS*** Module

  RN = Real Nonsymmetric

  CN = Complex Nonsymmetric
- Number of equations
- Number of solution vectors
- Any FMS Parameter

```
C       E X A M P L E   18
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_18')
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUB(25), LUC(25), LUX(25), LUY(25), LUE(25)
        INTEGER     LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
```

```
      POINTER (IMD_PTR, IMD)
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C
C     Input parameters:
      LOGICAL     ASK
      INTEGER     ASK_I
C
C     Local variables:
      INTEGER     IDTYPE, I, LB, LC, LX, LY, LE, LDISK, NV
      INTEGER     LENB, LENC, LENX, LENY, LENE
      REAL*8      EI, ERROR, ERROR2
      REAL*8      R_VCX, R_VYB, RZERO, RONE
      COMPLEX*16  C_VCX, C_VYB, CZERO, CONE
C
C     Profile vector for a full matrix:
      INTEGER     LOWEQ(1)
      DATA        LOWEQ/-1/
C
C     Common block to communicate with fill routines:
      COMMON /MYDATA/MOD, N, NRHS
C
C (1) Initialize FMS:
      CALL FMSINI
      CALL FMSPSH(MYNAME)
      CALL FMSIST ('MDATAU', 1)
      RZERO = 0.0D0
      RONE  = 1.0D0
      CZERO = DCMPLX(0.0D0, 0.0D0)
      CONE  = DCMPLX(1.0D0, 1.0D0)
    1 CONTINUE
      WRITE (6,*) 'Enter the FMS module number (2 or 5)'
      WRITE (6,*) '   2 = Real Nonsymmetric'
      WRITE (6,*) '   5 = Complex Nonsymmetric'
      READ  (5,*) MOD
      IF(MOD .EQ. 2) THEN
         IDTYPE = 1
      ELSE IF (MOD .EQ. 5) THEN
```

```
            IDTYPE = 2
         ELSE
            GO TO 1
         END IF
         N     = ASK_I('Enter the number of equations')
         NRHS  = ASK_I('Enter the number of vectors')
         WRITE (6,*) 'You may now alter any FMS parameter.'
         WRITE (6,*) 'When you are finished, type the letters RETURN'
         CALL FMSSET
         CALL FMSIGT ('MEMPTR', IMD_PTR)
         CALL FMSIGT ('MEMPTR', RMD_PTR)
         CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C (2)    Open FMS files:
         IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
         IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUB', LUB)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUC', LUC)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUY', LUY)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUE', LUE)
C
C        Allocate storage for one {B}, {C}, {X}, {Y} and {E}:
C        Initialize {B} and {C}:
         IF(IDTYPE .EQ. 1) THEN
            LENB = LUB(4)
            CALL FMSRMG (RMD, LB, LENB)
            LENC = LUC(4)
            CALL FMSRMG (RMD, LC, LENC)
            LENX = LUX(4)
            CALL FMSRMG (RMD, LX, LENX)
            LENY = LUY(4)
            CALL FMSRMG (RMD, LY, LENY)
            LENE = LUE(4)
            CALL FMSRMG (RMD, LE, LENE)
            DO I = 0,(N-1)
               RMD(LB+I) = RONE
               RMD(LC+I) = RONE
```

```
            END DO
         ELSE
            LENB = LUB(4)/2
            CALL FMSCMG (CMD, LB, LENB)
            LENC = LUC(4)/2
            CALL FMSCMG (CMD, LC, LENC)
            LENX = LUX(4)/2
            CALL FMSCMG (CMD, LX, LENX)
            LENY = LUY(4)/2
            CALL FMSCMG (CMD, LY, LENY)
            LENE = LUE(4)/2
            CALL FMSCMG (CMD, LE, LENE)
            DO I = 0,(N-1)
               CMD(LB+I) = CONE
               CMD(LC+I) = CONE
            END DO
         END IF
C
C (3)    Write data to FMS files:
C
C        Matrix file:
         IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
         IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C        Vector files:
         LDISK  = 1
         DO NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSWRT (LUB(1), LDISK, RMD(LB), LUB(4))
               CALL FMSWRT (LUC(1), LDISK, RMD(LC), LUC(4))
            ELSE
               CALL FMSWRT (LUB(1), LDISK, CMD(LB), LUB(4))
               CALL FMSWRT (LUC(1), LDISK, CMD(LC), LUC(4))
            END IF
            LDISK = LDISK + LUB(4)
         END DO
C
C (4)    Perform matrix algebra:
C
```

```
C       Populate and factor matrix [A]:
        IF(MOD.EQ.2) CALL RNDF (LUF, LUF, LUB, LUX,  NUMRED)
        IF(MOD.EQ.5) CALL CNDF (LUF, LUF, LUB, LUX,  NUMRED)
C
C       Solve [A]{X} = {B}:
        IF(MOD.EQ.2) CALL RNDS (LUF, LUB, LUX, NRHS, ISKIP)
        IF(MOD.EQ.5) CALL CNDS (LUF, LUB, LUX, NRHS, ISKIP)
C
C       Solve [A]T{Y} = {C}:
C       Set the FMS Parameter LUTRAN to solve with [A]T:
        CALL FMSIST ('LUTRAN', 1)
        IF(MOD.EQ.2) CALL RNDS (LUF, LUC, LUY, NRHS, ISKIP)
        IF(MOD.EQ.5) CALL CNDS (LUF, LUC, LUY, NRHS, ISKIP)
C       Reset the FMS Parameter LUTRAN for a normal solve:
        CALL FMSIST ('LUTRAN', 0)
C
C       Repopulate matrix [A]:
        CALL FMSIST ('MDATAU',  1)
        IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
        IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C       Multiply {E} = {B} - [A]{X}:
        CALL FMSMM (LUE, LUB, -1, 'N', LUF, LUX)
C
C (5)   Read data from FMS files:
C       Compute VCX and VYB:
C       Check the answer:
        ERROR  = RZERO
        ERROR2 = RZERO
        LDISK  = 1
        NWREAD = IDTYPE*N
        DO NV = 1,NRHS
           IF(IDTYPE .EQ. 1) THEN
C             Read in {B}nv, {C}nv, {X}nv, {Y}nv and {E}nv
              CALL FMSRED (LUB(1), LDISK, RMD(LB), NWREAD)
              CALL FMSRED (LUC(1), LDISK, RMD(LC), NWREAD)
              CALL FMSRED (LUX(1), LDISK, RMD(LX), NWREAD)
              CALL FMSRED (LUY(1), LDISK, RMD(LY), NWREAD)
              CALL FMSRED (LUE(1), LDISK, RMD(LE), NWREAD)
```

```
               LDISK  = LDISK  + LUB(4)
               R_VCX  = RZERO
               R_VYB  = RZERO
               DO I = 0,(N-1)
                  R_VCX = R_VCX + RMD(LC+I) * RMD(LX+I)
                  R_VYB = R_VYB + RMD(LY+I) * RMD(LB+I)
                  EI    = ABS(RMD(LE+I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
               EI = ABS(R_VCX - R_VYB)
               IF(EI .GT. ERROR2) ERROR2 = EI
            ELSE
C              Read in {B}nv, {C}nv, {X}nv, {Y}nv and {E}nv
               CALL FMSRED (LUB(1), LDISK, CMD(LB), NWREAD)
               CALL FMSRED (LUC(1), LDISK, CMD(LC), NWREAD)
               CALL FMSRED (LUX(1), LDISK, CMD(LX), NWREAD)
               CALL FMSRED (LUY(1), LDISK, CMD(LY), NWREAD)
               CALL FMSRED (LUE(1), LDISK, CMD(LE), NWREAD)
               LDISK  = LDISK  + LUB(4)
               C_VCX  = CZERO
               C_VYB  = CZERO
               DO I = 0,(N-1)
                  C_VCX = C_VCX + CMD(LC+I) * CMD(LX+I)
                  C_VYB = C_VYB + CMD(LY+I) * CMD(LB+I)
                  EI    = ABS(CMD(LE+I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
               EI = ABS(C_VCX - C_VYB)
               IF(EI .GT. ERROR2) ERROR2 = EI
            END IF
         END DO
         WRITE(6,*) 'MAXIMUM ERROR IN {B}-[A]{X}      =', ERROR
         WRITE(6,*) 'MAXIMUM ERROR IN {C}T{X}-{Y}T{B} =', ERROR2
C
C (6)    Close FMS files:
         CALL FMSCV (LUE)
         CALL FMSCV (LUY)
         CALL FMSCV (LUX)
         CALL FMSCV (LUC)
```

```
         CALL FMSCV (LUB)
         CALL FMSCM (LUF)
C
C        Return the memory:
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LE, LENE)
            CALL FMSRMR (RMD, LY, LENY)
            CALL FMSRMR (RMD, LX, LENX)
            CALL FMSRMR (RMD, LC, LENC)
            CALL FMSRMR (RMD, LB, LENB)
         ELSE
            CALL FMSCMR (CMD, LE, LENE)
            CALL FMSCMR (CMD, LY, LENY)
            CALL FMSCMR (CMD, LX, LENX)
            CALL FMSCMR (CMD, LC, LENC)
            CALL FMSCMR (CMD, LB, LENB)
         END IF
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
         END IF
         RETURN
 1000    FORMAT (A)
 2000    FORMAT (1X,A,' (y,n)>')
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
```

```
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), AIJ
        LOGICAL    IDIAG
C
        IF(IROW1 .EQ. JCOL1) THEN
           IDIAG = .TRUE.
        ELSE
           IDIAG = .FALSE.
        END IF
        AIJ = SQRT(2.0)
        DO I = IROW1,IROW2
           J1 = MAX0(LOWEQ(I),JCOL1)
           J2 = MIN0(I-1,JCOL2)
           DO J = J1,J2
              A(LOCEQ(I)+IJSTEP*J) = AIJ
              AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
           END DO
           IF(IDIAG) THEN
              D(I) = AIJ
              AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
           END IF
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
```

```
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), AIJ
C
        AIJ = SQRT(2.0)
        DO J = JCOL1,JCOL2
           I1 = MAX0(LOWEQ(J),IROW1)
           I2 = MIN0(J-1,IROW2)
           DO I = I1,I2
              A(LOCEQ(J)+IJSTEP*I) = AIJ
              AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
           END DO
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     AIJ, AIJ_RE, AIJ_IM
        COMPLEX*16 A(0:*), D(*)
        LOGICAL    IDIAG
C
        IF(IROW1 .EQ. JCOL1) THEN
           IDIAG = .TRUE.
        ELSE
           IDIAG = .FALSE.
        END IF
        AIJ = SQRT(2.0)
        DO I = IROW1,IROW2
           J1 = MAX0(LOWEQ(I),JCOL1)
           J2 = MIN0(I-1,JCOL2)
           DO J = J1,J2
              AIJ_RE = AIJ
              AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
```

```
               AIJ_IM = AIJ
               A(LOCEQ(I)+IJSTEP*J) = DCMPLX(AIJ_RE,AIJ_IM)
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
            END DO
            IF(IDIAG) THEN
               AIJ_RE = AIJ
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
               AIJ_IM = AIJ
               D(I) = DCMPLX(AIJ_RE,AIJ_IM)
               AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
            END IF
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     AIJ, AIJ_RE, AIJ_IM
         COMPLEX*16 A(0:*), D(*)
C
         AIJ = SQRT(2.0)
         DO J = JCOL1,JCOL2
            I1 = MAX0(LOWEQ(J),IROW1)
            I2 = MIN0(J-1,IROW2)
            DO I = I1,I2
               AIJ_RE = AIJ
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
               AIJ_IM = AIJ
               A(LOCEQ(J)+IJSTEP*I) = DCMPLX(AIJ_RE,AIJ_IM)
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
            END DO
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
```

```
C=======================================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        REAL*8     A(0:*), AIJ
C
        AIJ = SQRT(2.0)
        DO I=1,NUMEQ
           DO J=JEQN1,JEQN2
              LIJ = LOCI(I) + LOCJ(J,LUFLAG(I))
              A(LIJ) = AIJ
              AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
           END DO
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C=======================================================================
        INTEGER    JEQN1, JEQN2, NUMEQ
        INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
        COMPLEX*16 A(0:*)
        REAL*8     AIJ, AIJ_RE, AIJ_IM
C
        AIJ = SQRT(2.0)
        DO I=1,NUMEQ
           DO J=JEQN1,JEQN2
              LIJ    = LOCI(I) + LOCJ(J,LUFLAG(I))
              AIJ_RE = AIJ
              AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
              AIJ_IM = AIJ
              A(LIJ) = DCMPLX(AIJ_RE,AIJ_IM)
              AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
           END DO
        END DO
        RETURN
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   18
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_18')
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER     LUF(25)
        INTEGER     LUB(25), LUC(25), LUX(25), LUY(25), LUE(25)
        INTEGER     LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Input parameters:
        LOGICAL     ASK
        INTEGER     ASK_I
C
C       Local variables:
        INTEGER     IDTYPE, I, LB, LC, LX, LY, LE, LDISK, NV
        INTEGER     LENB, LENC, LENX, LENY, LENE
        REAL*8      EI, ERROR, ERROR2
        REAL*8      R_VCX, R_VYB, RZERO, RONE
        COMPLEX*16  C_VCX, C_VYB, CZERO, CONE
C
C       Profile vector for a full matrix:
        INTEGER     LOWEQ(1)
        DATA        LOWEQ/-1/
C
C       Common block to communicate with fill routines:
        COMMON /MYDATA/MOD, N, NRHS
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('MDATAU', 1)
        RZERO = 0.0D0
        RONE  = 1.0D0
        CZERO = DCMPLX(0.0D0, 0.0D0)
        CONE  = DCMPLX(1.0D0, 1.0D0)
    1   CONTINUE
        WRITE (6,*) 'Enter the FMS module number (2 or 5)'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        READ  (5,*) MOD
        IF(MOD .EQ. 2) THEN
           IDTYPE = 1
        ELSE IF (MOD .EQ. 5) THEN
           IDTYPE = 2
        ELSE
           GO TO 1
        END IF
```

```
      N     = ASK_I('Enter the number of equations')
      NRHS  = ASK_I('Enter the number of vectors')
      WRITE (6,*) 'You may now alter any FMS parameter.'
      WRITE (6,*) 'When you are finished, type the letters RETURN'
      CALL FMSSET
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C (2)   Open FMS files:
      IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUF', LUF)
      IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUF', LUF)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUB', LUB)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUC', LUC)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUY', LUY)
      CALL FMSOV (N, IDTYPE, NRHS, 'LUE', LUE)
C
C       Allocate storage for one {B}, {C}, {X}, {Y} and {E}:
C       Initialize {B} and {C}:
      IF(IDTYPE .EQ. 1) THEN
         LENB = LUB(4)
         CALL FMSRMG (RMD, LB, LENB)
         LENC = LUC(4)
         CALL FMSRMG (RMD, LC, LENC)
         LENX = LUX(4)
         CALL FMSRMG (RMD, LX, LENX)
         LENY = LUY(4)
         CALL FMSRMG (RMD, LY, LENY)
         LENE = LUE(4)
         CALL FMSRMG (RMD, LE, LENE)
         DO I = 0,(N-1)
            RMD(LB+I) = RONE
            RMD(LC+I) = RONE
         END DO
      ELSE
         LENB = LUB(4)/2
         CALL FMSCMG (CMD, LB, LENB)
         LENC = LUC(4)/2
         CALL FMSCMG (CMD, LC, LENC)
         LENX = LUX(4)/2
         CALL FMSCMG (CMD, LX, LENX)
         LENY = LUY(4)/2
         CALL FMSCMG (CMD, LY, LENY)
         LENE = LUE(4)/2
         CALL FMSCMG (CMD, LE, LENE)
         DO I = 0,(N-1)
            CMD(LB+I) = CONE
            CMD(LC+I) = CONE
         END DO
      END IF
C
C (3)   Write data to FMS files:
C
C       Matrix file:
      IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
      IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C       Vector files:
      LDISK  = 1
      DO NV = 1,NRHS
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSWRT (LUB(1), LDISK, RMD(LB), LUB(4))
            CALL FMSWRT (LUC(1), LDISK, RMD(LC), LUC(4))
         ELSE
```

```
               CALL FMSWRT (LUB(1), LDISK, CMD(LB), LUB(4))
               CALL FMSWRT (LUC(1), LDISK, CMD(LC), LUC(4))
            END IF
            LDISK = LDISK + LUB(4)
         END DO
C
C (4)    Perform matrix algebra:
C
C        Populate and factor matrix [A]:
         IF(MOD.EQ.2) CALL RNDF (LUF, LUF, LUB, LUX,  NUMRED)
         IF(MOD.EQ.5) CALL CNDF (LUF, LUF, LUB, LUX,  NUMRED)
C
C        Solve [A]{X} = {B}:
         IF(MOD.EQ.2) CALL RNDS (LUF, LUB, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUB, LUX, NRHS, ISKIP)
C
C        Solve [A]T{Y} = {C}:
C        Set the FMS Parameter LUTRAN to solve with [A]T:
         CALL FMSIST ('LUTRAN', 1)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUC, LUY, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUC, LUY, NRHS, ISKIP)
C        Reset the FMS Parameter LUTRAN for a normal solve:
         CALL FMSIST ('LUTRAN', 0)
C
C        Repopulate matrix [A]:
         CALL FMSIST ('MDATAU',  1)
         IF(MOD.EQ.2) CALL RNDA (LUS0, 0, LUF)
         IF(MOD.EQ.5) CALL CNDA (LUS0, 0, LUF)
C
C        Multiply {E} = {B} - [A]{X}:
         CALL FMSMM (LUE, LUB, -1, 'N', LUF, LUX)
C
C (5)    Read data from FMS files:
C        Compute VCX and VYB:
C        Check the answer:
         ERROR  = RZERO
         ERROR2 = RZERO
         LDISK  = 1
         NWREAD = IDTYPE*N
         DO NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
C              Read in {B}nv, {C}nv, {X}nv, {Y}nv and {E}nv
               CALL FMSRED (LUB(1), LDISK, RMD(LB), NWREAD)
               CALL FMSRED (LUC(1), LDISK, RMD(LC), NWREAD)
               CALL FMSRED (LUX(1), LDISK, RMD(LX), NWREAD)
               CALL FMSRED (LUY(1), LDISK, RMD(LY), NWREAD)
               CALL FMSRED (LUE(1), LDISK, RMD(LE), NWREAD)
               LDISK  = LDISK  + LUB(4)
               R_VCX  = RZERO
               R_VYB  = RZERO
               DO I = 0,(N-1)
                  R_VCX = R_VCX + RMD(LC+I) * RMD(LX+I)
                  R_VYB = R_VYB + RMD(LY+I) * RMD(LB+I)
                  EI    = ABS(RMD(LE+I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
               EI = ABS(R_VCX - R_VYB)
               IF(EI .GT. ERROR2) ERROR2 = EI
            ELSE
C              Read in {B}nv, {C}nv, {X}nv, {Y}nv and {E}nv
               CALL FMSRED (LUB(1), LDISK, CMD(LB), NWREAD)
               CALL FMSRED (LUC(1), LDISK, CMD(LC), NWREAD)
               CALL FMSRED (LUX(1), LDISK, CMD(LX), NWREAD)
               CALL FMSRED (LUY(1), LDISK, CMD(LY), NWREAD)
               CALL FMSRED (LUE(1), LDISK, CMD(LE), NWREAD)
```

```
               LDISK  = LDISK  + LUB(4)
               C_VCX  = CZERO
               C_VYB  = CZERO
               DO I = 0,(N-1)
                  C_VCX = C_VCX + CMD(LC+I) * CMD(LX+I)
                  C_VYB = C_VYB + CMD(LY+I) * CMD(LB+I)
                  EI   = ABS(CMD(LE+I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
               EI = ABS(C_VCX - C_VYB)
               IF(EI .GT. ERROR2) ERROR2 = EI
            END IF
         END DO
         WRITE(6,*) 'MAXIMUM ERROR IN {B}-[A]{X}      =', ERROR
         WRITE(6,*) 'MAXIMUM ERROR IN {C}T{X}-{Y}T{B} =', ERROR2
C
C (6)    Close FMS files:
         CALL FMSCV (LUE)
         CALL FMSCV (LUY)
         CALL FMSCV (LUX)
         CALL FMSCV (LUC)
         CALL FMSCV (LUB)
         CALL FMSCM (LUF)
C
C        Return the memory:
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LE, LENE)
            CALL FMSRMR (RMD, LY, LENY)
            CALL FMSRMR (RMD, LX, LENX)
            CALL FMSRMR (RMD, LC, LENC)
            CALL FMSRMR (RMD, LB, LENB)
         ELSE
            CALL FMSCMR (CMD, LE, LENE)
            CALL FMSCMR (CMD, LY, LENY)
            CALL FMSCMR (CMD, LX, LENX)
            CALL FMSCMR (CMD, LC, LENC)
            CALL FMSCMR (CMD, LB, LENB)
         END IF
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP (MYNAME)
         CALL FMSEND
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
         END IF
         RETURN
 1000    FORMAT (A)
 2000    FORMAT (1X,A,' (y,n)>')
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
         CHARACTER* (*) STRING
         WRITE(6,2000) STRING
         READ (5,*) ASK_I
```

```
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), AIJ
        LOGICAL    IDIAG
C
        IF(IROW1 .EQ. JCOL1) THEN
           IDIAG = .TRUE.
        ELSE
           IDIAG = .FALSE.
        END IF
        AIJ = SQRT(2.0)
        DO I = IROW1,IROW2
           J1 = MAX0(LOWEQ(I),JCOL1)
           J2 = MIN0(I-1,JCOL2)
           DO J = J1,J2
              A(LOCEQ(I)+IJSTEP*J) = AIJ
              AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
           END DO
           IF(IDIAG) THEN
              D(I) = AIJ
              AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
           END IF
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*), AIJ
C
        AIJ = SQRT(2.0)
        DO J = JCOL1,JCOL2
           I1 = MAX0(LOWEQ(J),IROW1)
           I2 = MIN0(J-1,IROW2)
           DO I = I1,I2
              A(LOCEQ(J)+IJSTEP*I) = AIJ
              AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
           END DO
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C=======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     AIJ, AIJ_RE, AIJ_IM
        COMPLEX*16 A(0:*), D(*)
        LOGICAL    IDIAG
C
        IF(IROW1 .EQ. JCOL1) THEN
           IDIAG = .TRUE.
        ELSE
           IDIAG = .FALSE.
```

```
         END IF
         AIJ = SQRT(2.0)
         DO I = IROW1,IROW2
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            DO J = J1,J2
               AIJ_RE = AIJ
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
               AIJ_IM = AIJ
               A(LOCEQ(I)+IJSTEP*J) = DCMPLX(AIJ_RE,AIJ_IM)
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
            END DO
            IF(IDIAG) THEN
               AIJ_RE = AIJ
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
               AIJ_IM = AIJ
               D(I) = DCMPLX(AIJ_RE,AIJ_IM)
               AIJ  = SQRT(10.0*(AIJ - INT(AIJ)))
            END IF
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     AIJ, AIJ_RE, AIJ_IM
         COMPLEX*16 A(0:*), D(*)
C
         AIJ = SQRT(2.0)
         DO J = JCOL1,JCOL2
            I1 = MAX0(LOWEQ(J),IROW1)
            I2 = MIN0(J-1,IROW2)
            DO I = I1,I2
               AIJ_RE = AIJ
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
               AIJ_IM = AIJ
               A(LOCEQ(J)+IJSTEP*I) = DCMPLX(AIJ_RE,AIJ_IM)
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
            END DO
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C=======================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         REAL*8     A(0:*), AIJ
C
         AIJ = SQRT(2.0)
         DO I=1,NUMEQ
            DO J=JEQN1,JEQN2
               LIJ = LOCI(I) + LOCJ(J,LUFLAG(I))
               A(LIJ) = AIJ
               AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
            END DO
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C=======================================================================
```

```
      INTEGER    JEQN1, JEQN2, NUMEQ
      INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
      COMPLEX*16 A(0:*)
      REAL*8     AIJ, AIJ_RE, AIJ_IM
C
      AIJ = SQRT(2.0)
      DO I=1,NUMEQ
         DO J=JEQN1,JEQN2
            LIJ    = LOCI(I) + LOCJ(J,LUFLAG(I))
            AIJ_RE = AIJ
            AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
            AIJ_IM = AIJ
            A(LIJ) = DCMPLX(AIJ_RE,AIJ_IM)
            AIJ    = SQRT(10.0*(AIJ - INT(AIJ)))
         END DO
      END DO
      RETURN
      END
```

Copyright © Multipath Corporation

FMS EXAMPLE 19. READING A MATRIX FILE

This example shows how to use the user supplied subroutines and subroutine xxDAF to read all the elements of a matrix file.
The flow of this EXAMPLE is as follows:

1. Populate matrix [A] with test data using FMSROW.
2. Populate vector {X} with test data using FMSWRT.
3. Factor [A] into [F].
4. Solve [A]{X}={B}.
5. Read [A] and compute average values in coarse windows.
6. Read [F] and compute average values in coarse windows.
7. Check the answer.

The following input may be specified:

- ***FMS*** Module
  1 = Real Symmetric
  2 = Real Nonsymmetric
  3 = Complex Hermitian
  4 = Complex Symmetric
  5 = Complex Nonsymmetric
- Number of equations
- Number of solution vectors
- The cell size to accumulate average values.
  For example, if the matrix has rank 1000 and a cell size of 100 is specified, then an array of 10x10 cells will be computed and printed. Each cell will contain the average value of the 100x100 matrix elements it represents.
- Any FMS Parameter

```
C       E X A M P L E   1 9
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_19')
```

```
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
C       Unfactored matrix:
           INTEGER     LUA(25)
C       Factored matrix:
           INTEGER     LUF(25)
C       R.H.S. and solution Vectors:
           INTEGER     LUX(25)
        INTEGER LUA0(25)
        DATA    LUA0(1)/0/
        INTEGER LUX0(25)
        INTEGER LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Local variables:
C       FMS Module number:
           INTEGER     MOD
C       Number of equations:
           INTEGER     N
           REAL*8      R8_N
C       Number of R.H.S. vectors:
           INTEGER     NRHS
C       Data type:
           INTEGER     IDTYPE
C       Loop counter:
           INTEGER     I
```

```
C       Vector starting address:
           INTEGER     LX
C       Vector storage length:
           INTEGER     LENX
C       Matrix starting address:
           INTEGER     LA
C       Disk location:
           INTEGER     LDISK
C       Vector number:
           INTEGER     NV
C       Current error:
           REAL*8      EI
C       Maximum overall error:
           REAL*8      ERROR
C       Input function:
           LOGICAL     ASK
           INTEGER     ASK_I
C       Diagonal value:
           COMPLEX*16  C16_DIA
C       Constants:
           REAL*8      R8_ZERO
           DATA        R8_ZERO/0.0D0/
           COMPLEX*16  C16_ZERO
           DATA        C16_ZERO/(0.0D0,0.0D0)/
           REAL*8      R8_ONE
           DATA        R8_ONE/1.0D0/
           COMPLEX*16  C16_ONE(1)
           DATA        C16_ONE/(1.0D0,-0.0D0)/
           REAL*8      R8_ALPHA(1)
           DATA        R8_ALPHA/1.0D0/
C       Profile vector for a full matrix:
           INTEGER     LOWEQ(1)
           DATA        LOWEQ/-1/
C
C       Common block to communicate with user supplied routines:
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C (1)   Initialize FMS:
        CALL FMSINI
```

```
      CALL FMSPSH(MYNAME)
      CALL FMSIST ('MFMAT' ,       2)
      CALL FMSIST ('LOGTIM',       3)
      CALL FMSIST ('IPRF'  ,    1026)
      CALL FMSIST ('IPRS'  ,    1026)
  100 CONTINUE
    1 CONTINUE
      WRITE (6,*) 'The FMS modules are numbered as follows:'
      WRITE (6,*) '   1 = Real Symmetric'
      WRITE (6,*) '   2 = Real Nonsymmetric'
      WRITE (6,*) '   3 = Complex Hermitian'
      WRITE (6,*) '   4 = Complex Symmetric'
      WRITE (6,*) '   5 = Complex Nonsymmetric'
      MOD   = ASK_I('Enter the FMS module number (1 to 5)')
      IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
      N     = ASK_I('Enter the number of equations')
      NRHS  = ASK_I('Enter the number of solution vectors')
      NCELL = ASK_I('Enter the cell size to average and print')
      WRITE (6,*) 'You may now alter any FMS parameter.'
      WRITE (6,*) 'When you are finished, type the letters RETURN'
      CALL FMSSET
      CALL FMSIGT ('MEMPTR', IMD_PTR)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
      CALL FMSIGT ('MEMPTR', CMD_PTR)
C     Check MFMAT:
      CALL FMSIGT ('MFMAT', MFMAT)
      IF(MFMAT .EQ. 3) THEN
         WRITE(6,*) 'Cannot save [A] while pivoting'
         WRITE(6,*) 'Building [A] as a Block Matrix'
         MFMAT = 2
      END IF
      IDTYPE = 1
      R8_N = DFLOAT(N)
      IF(MOD .EQ. 3) IDTYPE = 2
      IF(MOD .EQ. 4) IDTYPE = 2
      IF(MOD .EQ. 5) IDTYPE = 2
      NCELLS = (N + NCELL - 1)/NCELL
C
C (2) Open FMS files:
```

```
         IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
         CALL FMSOM (LUA, 'LUF', LUF)
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C (3)    Write data to FMS files:
C
C        Matrix File:
         IF(IDTYPE .EQ. 1) THEN
C           Real data:
C           Allocate a scratch vector:
            LENX = LUX(4)
            CALL FMSRMG (RMD, LX, LENX)
C           Initialize it to zero:
            DO I = 1,LENX
               RMD(LX-1+I) = R8_ZERO
            END DO
C
C           Write the R.H.S. vector file:
            RMD(LX) = R8_ONE
            LDISK  = 1
            DO NV = 1,NRHS
            CALL FMSWRT (LUX(1), LDISK, RMD(LX), LUIX(4))
               LDISK = LDISK + LUX(4)
            END DO
            RMD(LX) = R8_ZERO
C
C           Write the matrix file:
            CALL FMSROW (0, RMD(LX), LUA)
            RMD(LX) = R8_N
            DO I=2,N
               RMD(LX-1+I) = -R8_ONE
            END DO
            CALL FMSROW (1, RMD(LX), LUA)
            RMD(LX) = -R8_ONE
```

```
            DO I=2,N
               RMD(LX-1+I) = R8_ZERO
            END DO
            DO I=2,N
               RMD(LX-1+I) = R8_ONE
            CALL FMSROW (I, RMD(LX), LUA)
               RMD(LX-1+I) = R8_ZERO
            END DO
            CALL FMSROW (N+1, RMD(LX), LUA)
         ELSE
C           Complex data:
C           Allocate a scratch vector:
            LENX = LUX(4)/2
            CALL FMSCMG (CMD, LX, LENX)
            DO I = 1,LENX
               CMD(LX-1+I) = C16_ZERO
            END DO
C
C           Populate the vector:
            IF(MOD.EQ.3) THEN
               C16_DIA = (1.0D0,0.0D0)
            ELSE
               C16_DIA = (1.0D0,1.0D0)
            END IF
            CMD(LX) = C16_DIA
            LDISK  = 1
            DO NV = 1,NRHS
            CALL FMSWRT (LUX(1), LDISK, CMD(LX), LUX(4))
               LDISK = LDISK + LUX(4)
            END DO
C
C           Populate the matrix:
            CALL FMSROW (0, CMD(LX), LUA)
            IF(MOD .EQ. 3) THEN
               CMD(LX) = DCMPLX(R8_N,0.0D0)
            ELSE
               CMD(LX) = DCMPLX(R8_N,R8_N)
            END IF
            DO I=2,N
```

```
               CMD(LX-1+I) = -C16_DIA
            END DO
            CALL FMSROW (1, CMD(LX), LUA)
            CMD(LX) = -C16_DIA
            DO I=2,N
               CMD(LX-1+I) = C16_ZERO
            END DO
            DO I=2,N
               CMD(LX-1+I) = C16_DIA
            CALL FMSROW (I, CMD(LX), LUA)
               CMD(LX-1+I) = C16_ZERO
            END DO
            CALL FMSROW (N+1, CMD(LX), LUA)
         END IF
C
C (4)    Perform matrix algebra:
C        Assemble and factor the matrix, saving [A]:
         CALL FMSIST ('MDATAU', 0)
         IF(MOD.EQ.1) CALL RSDAF (LUA, R8_ALPHA,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.2) CALL RNDAF (LUA, R8_ALPHA,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.3) CALL CHDAF (LUA, R8_ALPHA,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.4) CALL CSDAF (LUA, C16_ONE,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.5) CALL CNDAF (LUA, C16_ONE,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
C
C        Perform forward reduction and back-substitution:
         IF(MOD.EQ.1) CALL RSDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.3) CALL CHDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.4) CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C        Accumulate and print average windows of [A]:
C        Allocate storage for an incore array of cells:
         NN = NCELLS*NCELLS
```

```
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMG (RMD, LA, NN)
            DO I=1,NN
               RMD(LA-1+I) = R8_ZERO
            END DO
         ELSE
            CALL FMSCMG (CMD, LA, NN)
            DO I=1,NN
               CMD(LA-1+I) = C16_ZERO
            END DO
         END IF
         CALL FMSIST ('MDATAU', 2)
         IF(MOD.EQ.1) CALL RSDAF (LUA,  R8_ALPHA, 1, LUS0, 0, LUA0, 0LUA,
     1    LUX0, LUX0, 0)
         IF(MOD.EQ.2) CALL RNDAF (LUA,  R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1    LUX0, LUX0, 0)
         IF(MOD.EQ.3) CALL CHDAF (LUA, R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1    LUX0, LUX0, 0)
         IF(MOD.EQ.4) CALL CSDAF (LUA, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
     1    LUX0, LUX0, 0)
         IF(MOD.EQ.5) CALL CNDAF (LUA, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
     1    LUX0, LUX0, 0)
C
C        Fill upper triangle of symmetric matrices:
         IF(MOD.EQ.1) CALL RFILL (RMD(LA), NCELLS)
         IF(MOD.EQ.3) CALL CFILL (CMD(LA), NCELLS)
         IF(MOD.EQ.4) CALL CFILL (CMD(LA), NCELLS)
C
C        You can change the format of the output.
C        These are the defaults:
         CALL FMSCST ('RFMAT', '(E12.4)')
         CALL FMSCST ('CFMAT', '(E12.4,1H,,E11.4)')
C
         IF(IDTYPE .EQ. 1) THEN
            WRITE(6,2000)
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL RAVG   (RMD(LA), NCELLS, NCELL, N)
            WRITE(6,2001)
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
```

```
         ELSE
            WRITE(6,2000)
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL CAVG   (CMD(LA), NCELLS, NCELL, N)
            WRITE(6,2001)
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
         END IF
C
C        Accumulate and print average windows of [F]:
         IF(IDTYPE .EQ. 1) THEN
            DO I=1,NN
               RMD(LA-1+I) = R8_ZERO
            END DO
         ELSE
            DO I=1,NN
               CMD(LA-1+I) = C16_ZERO
            END DO
         END IF
         IF(MOD.EQ.1) CALL RSDAF (LUF,  R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.2) CALL RNDAF (LUF,  R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.3) CALL CHDAF (LUF, R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.4) CALL CSDAF (LUF, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.5) CALL CNDAF (LUF, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
C
C        Fill upper triangle of symmetric matrices:
         IF(MOD.EQ.1) CALL RFILL (RMD(LA), NCELLS)
         IF(MOD.EQ.3) CALL CFILL (CMD(LA), NCELLS)
         IF(MOD.EQ.4) CALL CFILL (CMD(LA), NCELLS)
C
         IF(IDTYPE .EQ. 1) THEN
            WRITE(6,2002)
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL RAVG   (RMD(LA), NCELLS, NCELL, N)
            WRITE(6,2003)
```

```
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
         ELSE
            WRITE(6,2002)
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL CAVG   (CMD(LA), NCELLS, NCELL, N)
            WRITE(6,2003)
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
         END IF
C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
         LDISK  = 1
         DO 60 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRED (LUX(1), LDISK, RMD(LX), N)
               DO 50 I = 1,N
                  EI = ABS(RMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   50          CONTINUE
            ELSE
               CALL FMSRED (LUX(1), LDISK, CMD(LX), 2*N)
               DO 51 I = 1,N
                  EI = ABS(CMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   51          CONTINUE
            END IF
            LDISK  = LDISK  + LUX(4)
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUF)
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LA, NN)
            CALL FMSRMR (RMD, LX, LENX)
         ELSE
```

```
            CALL FMSCMR (CMD, LA, NN)
            CALL FMSCMR (CMD, LX, LENX)
         END IF
         IF(ASK('Do you want another solution?')) GO TO 100
         CALL FMSPOP(MYNAME)
            CALL FMSEND
 2000    FORMAT (/'Accumulated values of [A] in cells.')
 2001    FORMAT (/'Average values of [A] in cells.')
 2002    FORMAT (/'Accumulated values of [F] in cells.')
 2003    FORMAT (/'Average values of [F] in cells.')
         END
C        ================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*)
         COMMON /MYDATA/NCELL, NCELLS, LA
C
C        FMS memory management requires the following arrays:
         POINTER (RMD_PTR, RMD)
         REAL*8     RMD(0:1)
         WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         CALL FMSIGT ('MEMPTR', RMD_PTR)
C
C        Add this data to the appropriate cells:
         DO I = IROW1, IROW2
            IC = (I+NCELL-1)/NCELL
            JMAX = MIN0(I, JCOL2)
            DO J = JCOL1, JMAX
               JC = (J+NCELL-1)/NCELL
               LCIJ = LA - 1 + IC + NCELLS*(JC-1)
               IF(J .LT. I) THEN
C                 This is an off-diagonal term:
                  LIJ = LOCEQ(I) + IJSTEP*J
                  RMD(LCIJ) = RMD(LCIJ) + A(LIJ)
               ELSE
```

```
C                This is a diagonal term:
                 RMD(LCIJ) = RMD(LCIJ) + D(I)
              END IF
           END DO
        END DO
        WRITE(6,2001)
        RETURN
 2000   FORMAT (
     1  'RSUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('RSUBLK ending.')
        END
C       ================================================================
        SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*)
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C       FMS memory management requires the following arrays:
        POINTER (RMD_PTR, RMD)
        REAL*8     RMD(0:1)
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        CALL FMSIGT ('MEMPTR', RMD_PTR)
C
C       Add this data to the appropriate cells:
        DO J = JCOL1, JCOL2
           JC = (J+NCELL-1)/NCELL
           IMAX = MIN0(J-1, IROW2)
           DO I = IROW1, IMAX
              IC = (I+NCELL-1)/NCELL
              LCIJ = LA - 1 + IC + NCELLS*(JC-1)
              LIJ = LOCEQ(J) + IJSTEP*I
              RMD(LCIJ) = RMD(LCIJ) + A(LIJ)
           END DO
        END DO
        WRITE(6,2001)
        RETURN
```

```
 2000   FORMAT (
     1  'RNUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('RNUBLK ending.')
        END
C       ================================================================
        SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     D(*)
        COMPLEX*16 A(0:*)
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        COMPLEX*16 CMD(0:1)
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Add this data to the appropriate cells:
        DO I = IROW1, IROW2
           IC = (I+NCELL-1)/NCELL
           JMAX = MIN0(I, JCOL2)
           DO J = JCOL1, JMAX
              JC = (J+NCELL-1)/NCELL
              LCIJ = LA - 1 + IC + NCELLS*(JC-1)
              IF(J .LT. I) THEN
C                This is an off-diagonal term:
                 LIJ = LOCEQ(I) + IJSTEP*J
                 CMD(LCIJ) = CMD(LCIJ) + A(LIJ)
              ELSE
C                This is a diagonal term:
                 CMD(LCIJ) = CMD(LCIJ) + DCMPLX(D(I),0.0D0)
              END IF
           END DO
        END DO
        WRITE(6,2001)
        RETURN
```

```
 2000   FORMAT (
     1  'CHUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('CHUBLK ending.')
        END
C       ================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*)
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        COMPLEX*16 CMD(0:1)
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Add this data to the appropriate cells:
        DO I = IROW1, IROW2
           IC = (I+NCELL-1)/NCELL
           JMAX = MIN0(I, JCOL2)
           DO J = JCOL1, JMAX
              JC = (J+NCELL-1)/NCELL
              LCIJ = LA - 1 + IC + NCELLS*(JC-1)
              IF(J .LT. I) THEN
C                This is an off-diagonal term:
                 LIJ = LOCEQ(I) + IJSTEP*J
                 CMD(LCIJ) = CMD(LCIJ) + A(LIJ)
              ELSE
C                This is a diagonal term:
                 CMD(LCIJ) = CMD(LCIJ) + D(I)
              END IF
           END DO
        END DO
        WRITE(6,2001)
        RETURN
 2000   FORMAT (
```

```
     1  'CSUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('CSUBLK ending.')
        END
C       ================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*)
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        COMPLEX*16 CMD(0:1)
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Add this data to the appropriate cells:
        DO J = JCOL1, JCOL2
           JC = (J+NCELL-1)/NCELL
           IMAX = MIN0(J-1, IROW2)
           DO I = IROW1, IMAX
              IC = (I+NCELL-1)/NCELL
              LCIJ = LA - 1 + IC + NCELLS*(JC-1)
              LIJ = LOCEQ(J) + IJSTEP*I
              CMD(LCIJ) = CMD(LCIJ) + A(LIJ)
           END DO
        END DO
        WRITE(6,2001)
        RETURN
 2000   FORMAT (
     1  'CNUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('CNUBLK ending.')
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
```

```
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        WRITE(6,2001) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
 2001   FORMAT (4X,'You entered ',A)
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE RFILL (A, N)
C=======================================================================
C       Fill upper triangle of [A]
        REAL*8 A(N,N)
        INTEGER N
        INTEGER I, J
        DO J=2,N
           DO I=1,(J-1)
              A(I,J) = A(J,I)
           END DO
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE CFILL (A, N)
```

```
C=======================================================================
C       Fill upper triangle of [A]
        COMPLEX*16 A(N,N)
        INTEGER N
        INTEGER I, J
        DO J=2,N
           DO I=1,(J-1)
              A(I,J) = A(J,I)
           END DO
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE RAVG (A, NCELLS, NCELL, N)
C=======================================================================
        REAL*8 A(NCELLS,NCELLS)
        INTEGER NCELLS, NCELL, N
        INTEGER IC, JC, NI, NJ
        NJ = NCELL
        DO JC=1,NCELLS
           IF(JC .EQ. NCELLS) NJ = N - NCELL*(JC-1)
           NI = NCELL
           DO IC=1,NCELLS
              IF(IC .EQ. NCELLS) NI = N - NCELL*(IC-1)
              A(IC,JC) = A(IC,JC)/DFLOAT(NI*NJ)
           END DO
        END DO
        RETURN
        END
C=======================================================================
        SUBROUTINE CAVG (A, NCELLS, NCELL, N)
C=======================================================================
        COMPLEX*16 A(NCELLS,NCELLS)
        INTEGER NCELLS, NCELL, N
        INTEGER IC, JC, NI, NJ
        NJ = NCELL
        DO JC=1,NCELLS
           IF(JC .EQ. NCELLS) NJ = N - NCELL*(JC-1)
```

```
            NI = NCELL
            DO IC=1,NCELLS
               IF(IC .EQ. NCELLS) NI = N - NCELL*(IC-1)
               A(IC,JC) = A(IC,JC)/DFLOAT(NI*NJ)
            END DO
         END DO
         RETURN
         END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   1 9
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_19')
C
C       Number of vectors to reduce during factoring:
        PARAMETER (NUMRED = 0)
C
C       Skip operations during solving (no):
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
C       Unfactored matrix:
           INTEGER     LUA(25)
C       Factored matrix:
           INTEGER     LUF(25)
C       R.H.S. and solution Vectors:
           INTEGER     LUX(25)
        INTEGER LUA0(25)
        DATA    LUA0(1)/0/
        INTEGER LUX0(25)
        INTEGER LUS0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Local variables:
C       FMS Module number:
           INTEGER     MOD
C       Number of equations:
           INTEGER     N
           REAL*8      R8_N
C       Number of R.H.S. vectors:
           INTEGER     NRHS
C       Data type:
           INTEGER     IDTYPE
C       Loop counter:
           INTEGER     I
C       Vector starting address:
           INTEGER     LX
C       Vector storage length:
           INTEGER     LENX
C       Matrix starting address:
           INTEGER     LA
C       Disk location:
           INTEGER     LDISK
C       Vector number:
           INTEGER     NV
C       Current error:
           REAL*8      EI
C       Maximum overall error:
           REAL*8      ERROR
C       Input function:
           LOGICAL     ASK
           INTEGER     ASK_I
C       Diagonal value:
           COMPLEX*16  C16_DIA
```

```
C       Constants:
           REAL*8      R8_ZERO
           DATA        R8_ZERO/0.0D0/
           COMPLEX*16  C16_ZERO
           DATA        C16_ZERO/(0.0D0,0.0D0)/
           REAL*8      R8_ONE
           DATA        R8_ONE/1.0D0/
           COMPLEX*16  C16_ONE(1)
           DATA        C16_ONE/(1.0D0,-0.0D0)/
           REAL*8      R8_ALPHA(1)
           DATA        R8_ALPHA/1.0D0/
C       Profile vector for a full matrix:
           INTEGER     LOWEQ(1)
           DATA        LOWEQ/-1/
C
C       Common block to communicate with user supplied routines:
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('MFMAT' ,       2)
        CALL FMSIST ('LOGTIM',       3)
        CALL FMSIST ('IPRF'  ,    1026)
        CALL FMSIST ('IPRS'  ,    1026)
  100   CONTINUE
    1   CONTINUE
        WRITE (6,*) 'The FMS modules are numbered as follows:'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        MOD   = ASK_I('Enter the FMS module number (1 to 5)')
        IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
        N     = ASK_I('Enter the number of equations')
        NRHS  = ASK_I('Enter the number of solution vectors')
        NCELL = ASK_I('Enter the cell size to average and print')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C       Check MFMAT:
        CALL FMSIGT ('MFMAT', MFMAT)
        IF(MFMAT .EQ. 3) THEN
           WRITE(6,*) 'Cannot save [A] while pivoting'
           WRITE(6,*) 'Building [A] as a Block Matrix'
           MFMAT = 2
        END IF
        IDTYPE = 1
        R8_N = DFLOAT(N)
        IF(MOD .EQ. 3) IDTYPE = 2
        IF(MOD .EQ. 4) IDTYPE = 2
        IF(MOD .EQ. 5) IDTYPE = 2
        NCELLS = (N + NCELL - 1)/NCELL
C
C (2)   Open FMS files:
        IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
        CALL FMSOM (LUA, 'LUF', LUF)
```

```
         CALL FMSOV (N, IDTYPE, NRHS, 'LUX', LUX)
C
C (3)    Write data to FMS files:
C
C        Matrix File:
         IF(IDTYPE .EQ. 1) THEN
C           Real data:
C           Allocate a scratch vector:
            LENX = LUX(4)
            CALL FMSRMG (RMD, LX, LENX)
C           Initialize it to zero:
            DO I = 1,LENX
               RMD(LX-1+I) = R8_ZERO
            END DO
C
C           Write the R.H.S. vector file:
            RMD(LX) = R8_ONE
            LDISK  = 1
            DO NV = 1,NRHS
               CALL FMSWRT (LUX(1), LDISK, RMD(LX), LUX(4))
               LDISK = LDISK + LUX(4)
            END DO
            RMD(LX) = R8_ZERO
C
C           Write the matrix file:
            CALL FMSROW (0, RMD(LX), LUA)
            RMD(LX) = R8_N
            DO I=2,N
               RMD(LX-1+I) = -R8_ONE
            END DO
            CALL FMSROW (1, RMD(LX), LUA)
            RMD(LX) = -R8_ONE
            DO I=2,N
               RMD(LX-1+I) = R8_ZERO
            END DO
            DO I=2,N
               RMD(LX-1+I) = R8_ONE
               CALL FMSROW (I, RMD(LX), LUA)
               RMD(LX-1+I) = R8_ZERO
            END DO
            CALL FMSROW (N+1, RMD(LX), LUA)
         ELSE
C           Complex data:
C           Allocate a scratch vector:
            LENX = LUX(4)/2
            CALL FMSCMG (CMD, LX, LENX)
            DO I = 1,LENX
               CMD(LX-1+I) = C16_ZERO
            END DO
C
C           Populate the vector:
            IF(MOD.EQ.3) THEN
               C16_DIA = (1.0D0,0.0D0)
            ELSE
               C16_DIA = (1.0D0,1.0D0)
            END IF
            CMD(LX) = C16_DIA
            LDISK  = 1
            DO NV = 1,NRHS
               CALL FMSWRT (LUX(1), LDISK, CMD(LX), LUX(4))
               LDISK = LDISK + LUX(4)
            END DO
C
C           Populate the matrix:
            CALL FMSROW (0, CMD(LX), LUA)
```

```
            IF(MOD .EQ. 3) THEN
               CMD(LX) = DCMPLX(R8_N,0.0D0)
            ELSE
               CMD(LX) = DCMPLX(R8_N,R8_N)
            END IF
            DO I=2,N
               CMD(LX-1+I) = -C16_DIA
            END DO
            CALL FMSROW (1, CMD(LX), LUA)
            CMD(LX) = -C16_DIA
            DO I=2,N
               CMD(LX-1+I) = C16_ZERO
            END DO
            DO I=2,N
               CMD(LX-1+I) = C16_DIA
               CALL FMSROW (I, CMD(LX), LUA)
               CMD(LX-1+I) = C16_ZERO
            END DO
            CALL FMSROW (N+1, CMD(LX), LUA)
         END IF
C
C (4)    Perform matrix algebra:
C        Assemble and factor the matrix, saving [A]:
         CALL FMSIST ('MDATAU', 0)
         IF(MOD.EQ.1) CALL RSDAF (LUA,  R8_ALPHA,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.2) CALL RNDAF (LUA,  R8_ALPHA,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.3) CALL CHDAF (LUA, R8_ALPHA,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.4) CALL CSDAF (LUA, C16_ONE,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
         IF(MOD.EQ.5) CALL CNDAF (LUA, C16_ONE,
     1    1, LUS0, 0, LUA, LUF, LUX, LUX,  NUMRED)
C
C        Perform forward reduction and back-substitution:
         IF(MOD.EQ.1) CALL RSDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.2) CALL RNDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.3) CALL CHDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.4) CALL CSDS (LUF, LUX, LUX, NRHS, ISKIP)
         IF(MOD.EQ.5) CALL CNDS (LUF, LUX, LUX, NRHS, ISKIP)
C
C        Accumulate and print average windows of [A]:
C        Allocate storage for an incore array of cells:
         NN = NCELLS*NCELLS
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMG (RMD, LA, NN)
            DO I=1,NN
               RMD(LA-1+I) = R8_ZERO
            END DO
         ELSE
            CALL FMSCMG (CMD, LA, NN)
            DO I=1,NN
               CMD(LA-1+I) = C16_ZERO
            END DO
         END IF
         CALL FMSIST ('MDATAU', 2)
         IF(MOD.EQ.1) CALL RSDAF (LUA,  R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1    LUX0, LUX0, 0)
         IF(MOD.EQ.2) CALL RNDAF (LUA,  R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1    LUX0, LUX0, 0)
         IF(MOD.EQ.3) CALL CHDAF (LUA, R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1    LUX0, LUX0, 0)
         IF(MOD.EQ.4) CALL CSDAF (LUA, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
```

```
     1   LUX0, LUX0, 0)
         IF(MOD.EQ.5) CALL CNDAF (LUA, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
     1   LUX0, LUX0, 0)
C
C        Fill upper triangle of symmetric matrices:
         IF(MOD.EQ.1) CALL RFILL (RMD(LA), NCELLS)
         IF(MOD.EQ.3) CALL CFILL (CMD(LA), NCELLS)
         IF(MOD.EQ.4) CALL CFILL (CMD(LA), NCELLS)
C
C        You can change the format of the output.
C        These are the defaults:
         CALL FMSCST ('RFMAT', '(E12.4)')
         CALL FMSCST ('CFMAT', '(E12.4,1H,,E11.4)')
C
         IF(IDTYPE .EQ. 1) THEN
            WRITE(6,2000)
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL RAVG   (RMD(LA), NCELLS, NCELL, N)
            WRITE(6,2001)
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
         ELSE
            WRITE(6,2000)
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL CAVG   (CMD(LA), NCELLS, NCELL, N)
            WRITE(6,2001)
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
         END IF
C
C        Accumulate and print average windows of [F]:
         IF(IDTYPE .EQ. 1) THEN
            DO I=1,NN
               RMD(LA-1+I) = R8_ZERO
            END DO
         ELSE
            DO I=1,NN
               CMD(LA-1+I) = C16_ZERO
            END DO
         END IF
         IF(MOD.EQ.1) CALL RSDAF (LUF,  R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.2) CALL RNDAF (LUF,  R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.3) CALL CHDAF (LUF, R8_ALPHA, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.4) CALL CSDAF (LUF, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
         IF(MOD.EQ.5) CALL CNDAF (LUF, C16_ONE, 1, LUS0, 0, LUA0, LUA0,
     1      LUX0, LUX0, 0)
C
C        Fill upper triangle of symmetric matrices:
         IF(MOD.EQ.1) CALL RFILL (RMD(LA), NCELLS)
         IF(MOD.EQ.3) CALL CFILL (CMD(LA), NCELLS)
         IF(MOD.EQ.4) CALL CFILL (CMD(LA), NCELLS)
C
         IF(IDTYPE .EQ. 1) THEN
            WRITE(6,2002)
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL RAVG   (RMD(LA), NCELLS, NCELL, N)
            WRITE(6,2003)
            CALL FMSPRF (RMD(LA), NCELLS, NCELLS, 2, IDTYPE)
         ELSE
            WRITE(6,2002)
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
            CALL CAVG   (CMD(LA), NCELLS, NCELL, N)
            WRITE(6,2003)
```

```
            CALL FMSPRF (CMD(LA), NCELLS, NCELLS, 2, IDTYPE)
         END IF
C
C (5)    Read data from FMS files:
C        Check the answer:
         ERROR = 0.0D0
         LDISK  = 1
         DO 60 NV = 1,NRHS
            IF(IDTYPE .EQ. 1) THEN
               CALL FMSRED (LUX(1), LDISK, RMD(LX), N)
               DO 50 I = 1,N
                  EI = ABS(RMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   50          CONTINUE
            ELSE
               CALL FMSRED (LUX(1), LDISK, CMD(LX), 2*N)
               DO 51 I = 1,N
                  EI = ABS(CMD(LX-1+I) - 1.0D0)
                  IF(EI .GT. ERROR) ERROR = EI
   51          CONTINUE
            END IF
            LDISK  = LDISK  + LUX(4)
   60    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
         CALL FMSCM (LUF)
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         IF(IDTYPE .EQ. 1) THEN
            CALL FMSRMR (RMD, LA, NN)
            CALL FMSRMR (RMD, LX, LENX)
         ELSE
            CALL FMSCMR (CMD, LA, NN)
            CALL FMSCMR (CMD, LX, LENX)
         END IF
         IF(ASK('Do you want another solution?')) GO TO 100
         CALL FMSPOP (MYNAME)
         CALL FMSEND
 2000    FORMAT (/'Accumulated values of [A] in cells.')
 2001    FORMAT (/'Average values of [A] in cells.')
 2002    FORMAT (/'Accumulated values of [F] in cells.')
 2003    FORMAT (/'Average values of [F] in cells.')
         END
C        ================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*)
         COMMON /MYDATA/NCELL, NCELLS, LA
C
C        FMS memory management requires the following arrays:
         POINTER (RMD_PTR, RMD)
         REAL*8     RMD(0:1)
         WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         CALL FMSIGT ('MEMPTR', RMD_PTR)
C
C        Add this data to the appropriate cells:
         DO I = IROW1, IROW2
            IC = (I+NCELL-1)/NCELL
            JMAX = MIN0(I, JCOL2)
            DO J = JCOL1, JMAX
               JC = (J+NCELL-1)/NCELL
```

```
               LCIJ = LA - 1 + IC + NCELLS*(JC-1)
               IF(J .LT. I) THEN
C                 This is an off-diagonal term:
                  LIJ = LOCEQ(I) + IJSTEP*J
                  RMD(LCIJ) = RMD(LCIJ) + A(LIJ)
               ELSE
C                 This is a diagonal term:
                  RMD(LCIJ) = RMD(LCIJ) + D(I)
               END IF
            END DO
         END DO
         WRITE(6,2001)
         RETURN
 2000    FORMAT (
     1   'RSUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001    FORMAT ('RSUBLK ending.')
         END
C        ================================================================
         SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*)
         COMMON /MYDATA/NCELL, NCELLS, LA
C
C        FMS memory management requires the following arrays:
         POINTER (RMD_PTR, RMD)
         REAL*8     RMD(0:1)
         WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         CALL FMSIGT ('MEMPTR', RMD_PTR)
C
C        Add this data to the appropriate cells:
         DO J = JCOL1, JCOL2
            JC = (J+NCELL-1)/NCELL
            IMAX = MIN0(J-1, IROW2)
            DO I = IROW1, IMAX
               IC = (I+NCELL-1)/NCELL
               LCIJ = LA - 1 + IC + NCELLS*(JC-1)
               LIJ = LOCEQ(J) + IJSTEP*I
               RMD(LCIJ) = RMD(LCIJ) + A(LIJ)
            END DO
         END DO
         WRITE(6,2001)
         RETURN
 2000    FORMAT (
     1   'RNUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001    FORMAT ('RNUBLK ending.')
         END
C        ================================================================
         SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     D(*)
         COMPLEX*16 A(0:*)
         COMMON /MYDATA/NCELL, NCELLS, LA
C
C        FMS memory management requires the following arrays:
         POINTER (CMD_PTR, CMD)
         COMPLEX*16 CMD(0:1)
         WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         CALL FMSIGT ('MEMPTR', CMD_PTR)
```

```
C
C       Add this data to the appropriate cells:
        DO I = IROW1, IROW2
           IC = (I+NCELL-1)/NCELL
           JMAX = MIN0(I, JCOL2)
           DO J = JCOL1, JMAX
              JC = (J+NCELL-1)/NCELL
              LCIJ = LA - 1 + IC + NCELLS*(JC-1)
              IF(J .LT. I) THEN
C                This is an off-diagonal term:
                 LIJ = LOCEQ(I) + IJSTEP*J
                 CMD(LCIJ) = CMD(LCIJ) + A(LIJ)
              ELSE
C                This is a diagonal term:
                 CMD(LCIJ) = CMD(LCIJ) + DCMPLX(D(I),0.0D0)
              END IF
           END DO
        END DO
        WRITE(6,2001)
        RETURN
 2000   FORMAT (
     1  'CHUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('CHUBLK ending.')
        END
C       ================================================================
        SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*)
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        COMPLEX*16 CMD(0:1)
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Add this data to the appropriate cells:
        DO I = IROW1, IROW2
           IC = (I+NCELL-1)/NCELL
           JMAX = MIN0(I, JCOL2)
           DO J = JCOL1, JMAX
              JC = (J+NCELL-1)/NCELL
              LCIJ = LA - 1 + IC + NCELLS*(JC-1)
              IF(J .LT. I) THEN
C                This is an off-diagonal term:
                 LIJ = LOCEQ(I) + IJSTEP*J
                 CMD(LCIJ) = CMD(LCIJ) + A(LIJ)
              ELSE
C                This is a diagonal term:
                 CMD(LCIJ) = CMD(LCIJ) + D(I)
              END IF
           END DO
        END DO
        WRITE(6,2001)
        RETURN
 2000   FORMAT (
     1  'CSUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('CSUBLK ending.')
        END
C       ================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
```

```
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*)
        COMMON /MYDATA/NCELL, NCELLS, LA
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        COMPLEX*16 CMD(0:1)
        WRITE(6,2000) IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        CALL FMSIGT ('MEMPTR', CMD_PTR)
C
C       Add this data to the appropriate cells:
        DO J = JCOL1, JCOL2
           JC = (J+NCELL-1)/NCELL
           IMAX = MIN0(J-1, IROW2)
           DO I = IROW1, IMAX
              IC = (I+NCELL-1)/NCELL
              LCIJ = LA - 1 + IC + NCELLS*(JC-1)
              LIJ = LOCEQ(J) + IJSTEP*I
              CMD(LCIJ) = CMD(LCIJ) + A(LIJ)
           END DO
        END DO
        WRITE(6,2001)
        RETURN
 2000   FORMAT (
     1  'CNUBLK filling A(',I6,':',I6,',',I6,':',I6,'), IJSTEP=',I6)
 2001   FORMAT ('CNUBLK ending.')
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        WRITE(6,2001) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
 2001   FORMAT (4X,'You entered ',A)
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
C=======================================================================
        SUBROUTINE RFILL (A, N)
C=======================================================================
C       Fill upper triangle of [A]
        REAL*8 A(N,N)
        INTEGER N
        INTEGER I, J
        DO J=2,N
           DO I=1,(J-1)
```

```
               A(I,J) = A(J,I)
            END DO
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE CFILL (A, N)
C=======================================================================
C        Fill upper triangle of [A]
         COMPLEX*16 A(N,N)
         INTEGER N
         INTEGER I, J
         DO J=2,N
            DO I=1,(J-1)
               A(I,J) = A(J,I)
            END DO
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE RAVG (A, NCELLS, NCELL, N)
C=======================================================================
         REAL*8 A(NCELLS,NCELLS)
         INTEGER NCELLS, NCELL, N
         INTEGER IC, JC, NI, NJ
         NJ = NCELL
         DO JC=1,NCELLS
            IF(JC .EQ. NCELLS) NJ = N - NCELL*(JC-1)
            NI = NCELL
            DO IC=1,NCELLS
               IF(IC .EQ. NCELLS) NI = N - NCELL*(IC-1)
               A(IC,JC) = A(IC,JC)/DFLOAT(NI*NJ)
            END DO
         END DO
         RETURN
         END
C=======================================================================
         SUBROUTINE CAVG (A, NCELLS, NCELL, N)
C=======================================================================
         COMPLEX*16 A(NCELLS,NCELLS)
         INTEGER NCELLS, NCELL, N
         INTEGER IC, JC, NI, NJ
         NJ = NCELL
         DO JC=1,NCELLS
            IF(JC .EQ. NCELLS) NJ = N - NCELL*(JC-1)
            NI = NCELL
            DO IC=1,NCELLS
               IF(IC .EQ. NCELLS) NI = N - NCELL*(IC-1)
               A(IC,JC) = A(IC,JC)/DFLOAT(NI*NJ)
            END DO
         END DO
         RETURN
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 20. ESTIMATING DISK SPACE

This example illustrates the use of subroutine FMSIN2 and FMS Parameter NOOPEN for estimating disk space requirements for matrix and vector files.
The following input may be specified:

- ***FMS*** Module

1 = Real Symmetric
2 = Real Nonsymmetric
3 = Complex Hermitian
4 = Complex Symmetric
5 = Complex Nonsymmetric

- Number of equations
- Number of solution vectors
- Type of Initialization

0 = Call FMSINI for a normal initialization,
1 = Call FMSIN2(1) for a reduced initialization,
2 = Call FMSIN2(2) for a reduced initialization with no output.

- Any FMS Parameter

```
C       E X A M P L E   20
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_20')
C
C       FMS matrix file attribute:
C       Unfactored matrix:
           INTEGER     LUA(25)
C       FMS matrix attribute elements:
           INTEGER     LUA_LENRLU
           PARAMETER  (LUA_LENRLU= 5)
           INTEGER     LUA_LENDIA
           PARAMETER  (LUA_LENDIA= 9)
           INTEGER     LUA_NUMRLU
```

```
            PARAMETER  (LUA_NUMRLU=10)
            INTEGER     LUA_LENTAB
            PARAMETER  (LUA_LENTAB=15)
C
C       FMS vector file attributes:
C       FMS vector file attributes:
            INTEGER     LUX(25)
C       FMS vector attribute elements:
            INTEGER     LUX_LENREC
            PARAMETER  (LUX_LENREC= 4)
            INTEGER     LUX_NUMVEC
            PARAMETER  (LUX_NUMVEC= 6)
C
C       FMS Parameter values:
            INTEGER     MFMAT
            INTEGER     INCORE, INCORE_S
            INTEGER     NOOPEN, NOOPEN_S
            INTEGER     NUMIOQ
C
C       Local variables:
C       FMS Module number:
            INTEGER     MOD
C       Number of equations:
            INTEGER     N
            REAL*8      R8_N
C       Number of R.H.S. vectors:
            INTEGER     NRHS
C       Data type:
            INTEGER     IDTYPE
C       Matrix symmetry:
            INTEGER     ISTYPE
C       Input function:
            LOGICAL     ASK
            INTEGER     ASK_I
C       Minimum file size
            REAL*8      R8SIZE
            INTEGER     KBMIN
C       FMS Off-diagonal matrix size:
            REAL*8      R8OFF
```

```
            INTEGER     KBOFF
C       FMS Striped file size (Kbytes)
            INTEGER     KBS
C       FMS Non-striped file size (Kbytes)
            INTEGER     KBNS
C       Ratio of FMS to minimum size:
            REAL*8      RATIO
C       Profile vector for a full matrix:
            INTEGER     LOWEQ(1)
            DATA        LOWEQ/-1/
  100   CONTINUE
    1   CONTINUE
        WRITE (6,*) 'The FMS modules are numbered as follows:'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        MOD    = ASK_I('Enter the FMS module number (1 to 5)')
        IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
        IDTYPE = 1
        IF(MOD .EQ. 3) IDTYPE = 2
        IF(MOD .EQ. 4) IDTYPE = 2
        IF(MOD .EQ. 5) IDTYPE = 2
        ISTYPE = 1
        IF((MOD .EQ. 2) .OR. (MOD .EQ. 5)) ISTYPE = 2
        IF (MOD .EQ. 3) ISTYPE = 3
        N      = ASK_I('Enter the number of equations')
        R8_N   = DFLOAT(N)
        NRHS   = ASK_I('Enter the number of solution vectors')
        WRITE (6,*) 'FMS may be initialized as follows:'
        WRITE (6,*) '   0 = Normal'
        WRITE (6,*) '   1 = Reduced with some output'
        WRITE (6,*) '   2 = Reduced with no output'
        INIT   = ASK_I('Type of initialization (Argument to FMSIN2)')
C
C (1)   Initialize FMS:
        CALL FMSIGT ('LICAPL' , LICAPL  )
        CALL FMSIN2 (INIT)
```

```
         CALL FMSPSH(MYNAME)
         WRITE (6,*) 'You may now alter any FMS parameter.'
         WRITE (6,*) 'When you are finished, type the letters RETURN'
         CALL FMSSET
         CALL FMSIGT ('MFMAT' , MFMAT   )
         CALL FMSIGT ('NUMIOQ', NUMIOQ  )
         CALL FMSIGT ('INCORE', INCORE_S)
         CALL FMSIST ('INCORE',        0)
         CALL FMSIGT ('NOOPEN', NOOPEN_S)
         CALL FMSIST ('NOOPEN',        1)
C
C (2)    Open FMS files:
         IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
         IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
C
C        Compute the size of the off-diagonal matrix files:
         R8OFF  = DFLOAT(LUA(LUA_LENRLU)) * DFLOAT(LUA(LUA_NUMRLU))
         IF(ISTYPE .EQ. 2) R8OFF = (2.0D0)*R8OFF
         KBOFF  = INT(R8OFF/(128.0D0))
C
C        Compute the size of the striped and non-striped files:
         KBNS = ( LUA(LUA_LENDIA) +
     1           LUA(LUA_LENTAB) )/128
         IF(NUMIOQ .EQ. 0) THEN
C           Matrix files are not striped:
            KBS  = 0
            KBNS = KBNS + KBOFF
         ELSE
C           Matrix files are striped:
            KBS  = KBOFF
         END IF
C
C        Compute the minimum size required:
         IF(ISTYPE .EQ. 2) THEN
            R8SIZE = R8_N * R8_N
         ELSE
```

```
            R8SIZE = R8_N * (R8_N + 1.0D0)/(2.0D0)
         END IF
         IF(IDTYPE .EQ. 2) R8SIZE = (2.0D0)*R8SIZE
         KBMIN = INT(R8SIZE/(128.0D0))
         RATIO = DFLOAT(KBS+KBNS)/DFLOAT(KBMIN)
         WRITE(6,2000) N, MFMAT, IDTYPE, ISTYPE, KBMIN, KBS, KBNS, RATIO
C
C        Compute the size of the FMS vector file:
         CALL FMSOV  (N, IDTYPE, NRHS, 'LUX', LUX)
         R8SIZE = DFLOAT(LUX(LUX_LENREC)) * DFLOAT(LUX(LUX_NUMVEC))
         KBX    = INT( R8SIZE/(128.0D0) )
         IF(NUMIOQ .EQ. 0) THEN
C           Vector file is not striped:
            KBNS = KBX
            KBS  = 0
         ELSE
C           Vector file is striped:
            KBNS = 0
            KBS  = KBX
         END IF
C
C        Compute the minimum space required:
         R8SIZE = R8_N * DFLOAT(NRHS)
         IF(IDTYPE .EQ. 2) R8SIZE = (2.0D0)*R8SIZE
         KBMIN = INT(R8SIZE/(128.0D0))
         RATIO = DFLOAT(KBS+KBNS)/DFLOAT(KBMIN)
         WRITE(6,2001) N, NRHS, IDTYPE, KBMIN, KBS, KBNS, RATIO
C
C        Show the status of the Matrix and Vector files:
         CALL FMSIST ('SHOW', 'FILES')
C
C (6)    Close FMS files:
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         CALL FMSIST ('INCORE', INCORE_S)
         CALL FMSIST ('NOOPEN', NOOPEN_S)
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         IF(ASK('Do you want another solution?')) GO TO 100
```

```
 2000   FORMAT (/
     1   'File size for matrix:'/
     2   '   Number of equations.............=',I10/
     3   '   Matrix format...................=',I10/
     4   '   Data type.......................=',I10/
     5   '   Symmetry........................=',I10/
     6   '   Minimum space...........(Kbytes)=',I10/
     7   '   FMS striped space.......(Kbytes)=',I10/
     8   '   FMS non-striped space...(Kbytes)=',I10/
     9   '   Ratio (FMS space/Minimum space).=',F10.2)
 2001   FORMAT (/
     1   'File size for vectors:'/
     2   '   Number of equations.............=',I10/
     3   '   Number of vectors...............=',I10/
     4   '   Data type.......................=',I10/
     5   '   Minimum space...........(Kbytes)=',I10/
     6   '   FMS striped space.......(Kbytes)=',I10/
     7   '   FMS non-striped space...(Kbytes)=',I10/
     8   '   Ratio (FMS space/Minimum space).=',F10.2)
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        WRITE(6,2001) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
 2001   FORMAT (4X,'You entered ',A)
        END
C=======================================================================
```

```
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   20
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_20')
C
C       FMS matrix file attributes:
C       Unfactored matrix:
           INTEGER     LUA(25)
C       FMS matrix attribute elements:
           INTEGER     LUA_LENRLU
           PARAMETER  (LUA_LENRLU= 5)
           INTEGER     LUA_LENDIA
           PARAMETER  (LUA_LENDIA= 9)
           INTEGER     LUA_NUMRLU
           PARAMETER  (LUA_NUMRLU=10)
           INTEGER     LUA_LENTAB
           PARAMETER  (LUA_LENTAB=15)
C
           INTEGER     LUX(25)
C       FMS vector attribute elements:
           INTEGER     LUX_LENREC
           PARAMETER  (LUX_LENREC= 4)
           INTEGER     LUX_NUMVEC
           PARAMETER  (LUX_NUMVEC= 6)
C
C       FMS Parameter values:
           INTEGER     MFMAT
           INTEGER     INCORE_S
           INTEGER     NOOPEN_S
           INTEGER     NUMIOQ
C
C       Local variables:
C       FMS Module number:
           INTEGER     MOD
C       Number of equations:
           INTEGER     N
           REAL*8      R8_N
C       Number of R.H.S. vectors:
           INTEGER     NRHS
C       Data type:
           INTEGER     IDTYPE
C       Matrix symmetry:
           INTEGER     ISTYPE
C       Input function:
           LOGICAL     ASK
           INTEGER     ASK_I
C       Minimum file size
           REAL*8      R8SIZE
           INTEGER     KBMIN
C       FMS Off-diagonal matrix size:
           REAL*8      R8OFF
           INTEGER     KBOFF
C       FMS Striped file size (Kbytes)
           INTEGER     KBS
C       FMS Non-striped file size (Kbytes)
           INTEGER     KBNS
C       Ratio of FMS to minimum size:
           REAL*8      RATIO
C       Profile vector for a full matrix:
           INTEGER     LOWEQ(1)
           DATA        LOWEQ/-1/
  100   CONTINUE
```

```
    1   CONTINUE
        WRITE (6,*) 'The FMS modules are numbered as follows:'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
        MOD    = ASK_I('Enter the FMS module number (1 to 5)')
        IF( (MOD.LT.1) .OR. (MOD.GT.5) ) GO TO 1
        IDTYPE = 1
        IF(MOD .EQ. 3) IDTYPE = 2
        IF(MOD .EQ. 4) IDTYPE = 2
        IF(MOD .EQ. 5) IDTYPE = 2
        ISTYPE = 1
        IF((MOD .EQ. 2) .OR. (MOD .EQ. 5)) ISTYPE = 2
        IF (MOD .EQ. 3) ISTYPE = 3
        N      = ASK_I('Enter the number of equations')
        R8_N   = DFLOAT(N)
        NRHS   = ASK_I('Enter the number of solution vectors')
        WRITE (6,*) 'FMS may be initialized as follows:'
        WRITE (6,*) '   0 = Normal'
        WRITE (6,*) '   1 = Reduced with some output'
        WRITE (6,*) '   2 = Reduced with no output'
        INIT   = ASK_I('Type of initialization (Argument to FMSIN2)')
C
C (1)   Initialize FMS:
        CALL FMSIN2 (INIT)
        CALL FMSIGT ('LICAPL' , LICAPL  )
        CALL FMSPSH (MYNAME)
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MFMAT' , MFMAT   )
        CALL FMSIGT ('NUMIOQ', NUMIOQ  )
        CALL FMSIGT ('INCORE', INCORE_S)
        CALL FMSIST ('INCORE',        0)
        CALL FMSIGT ('NOOPEN', NOOPEN_S)
        CALL FMSIST ('NOOPEN',        1)
C
C (2)   Open FMS files:
        IF(MOD.EQ.1) CALL RSDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.2) CALL RNDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.3) CALL CHDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.4) CALL CSDI (LOWEQ, N, 'LUA', LUA)
        IF(MOD.EQ.5) CALL CNDI (LOWEQ, N, 'LUA', LUA)
C
C       Compute the size of the off-diagonal matrix files:
        R8OFF  = DFLOAT(LUA(LUA_LENRLU)) * DFLOAT(LUA(LUA_NUMRLU))
        IF(ISTYPE .EQ. 2) R8OFF = (2.0D0)*R8OFF
        KBOFF  = INT(R8OFF/(128.0D0))
C
C       Compute the size of the striped and non-striped files:
        KBNS = ( LUA(LUA_LENDIA) +
     1           LUA(LUA_LENTAB) )/128
        IF(NUMIOQ .EQ. 0) THEN
C          Matrix files are not striped:
           KBS  = 0
           KBNS = KBNS + KBOFF
        ELSE
C          Matrix files are striped:
           KBS  = KBOFF
        END IF
C
C       Compute the minimum size required:
        IF(ISTYPE .EQ. 2) THEN
```

```
            R8SIZE = R8_N * R8_N
         ELSE
            R8SIZE = R8_N * (R8_N + 1.0D0)/(2.0D0)
         END IF
         IF(IDTYPE .EQ. 2) R8SIZE = (2.0D0)*R8SIZE
         KBMIN = INT(R8SIZE/(128.0D0))
         RATIO = DFLOAT(KBS+KBNS)/DFLOAT(KBMIN)
         WRITE(6,2000) N, MFMAT, IDTYPE, ISTYPE, KBMIN, KBS, KBNS, RATIO
C
C        Compute the size of the FMS vector file:
         CALL FMSOV  (N, IDTYPE, NRHS, 'LUX', LUX)
         R8SIZE = DFLOAT(LUX(LUX_LENREC)) * DFLOAT(LUX(LUX_NUMVEC))
         KBX    = INT( R8SIZE/(128.0D0) )
         IF(NUMIOQ .EQ. 0) THEN
C           Vector file is not striped:
            KBNS = KBX
            KBS  = 0
         ELSE
C           Vector file is striped:
            KBNS = 0
            KBS  = KBX
         END IF
C
C        Compute the minimum space required:
         R8SIZE = R8_N * DFLOAT(NRHS)
         IF(IDTYPE .EQ. 2) R8SIZE = (2.0D0)*R8SIZE
         KBMIN = INT(R8SIZE/(128.0D0))
         RATIO = DFLOAT(KBS+KBNS)/DFLOAT(KBMIN)
         WRITE(6,2001) N, NRHS, IDTYPE, KBMIN, KBS, KBNS, RATIO
C
C        Show the status of the Matrix and Vector files:
         CALL FMSCST ('SHOW', 'FILES')
C
C (6)    Close FMS files:
         CALL FMSCM (LUA)
         CALL FMSCV (LUX)
         CALL FMSIST ('INCORE', INCORE_S)
         CALL FMSIST ('NOOPEN', NOOPEN_S)
         CALL FMSPOP (MYNAME)
         CALL FMSEND
         IF(ASK('Do you want another solution?')) GO TO 100
 2000    FORMAT (/
     1    'File size for matrix:'/
     2    '   Number of equations.............=',I10/
     3    '   Matrix format...................=',I10/
     4    '   Data type.......................=',I10/
     5    '   Symmetry........................=',I10/
     6    '   Minimum space...........(Kbytes)=',I10/
     7    '   FMS striped space.......(Kbytes)=',I10/
     8    '   FMS non-striped space...(Kbytes)=',I10/
     9    '   Ratio (FMS space/Minimum space).=',F10.2)
 2001    FORMAT (/
     1    'File size for vectors:'/
     2    '   Number of equations.............=',I10/
     3    '   Number of vectors...............=',I10/
     4    '   Data type.......................=',I10/
     5    '   Minimum space...........(Kbytes)=',I10/
     6    '   FMS striped space.......(Kbytes)=',I10/
     7    '   FMS non-striped space...(Kbytes)=',I10/
     8    '   Ratio (FMS space/Minimum space).=',F10.2)
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
```

```
      CHARACTER*1 IYN
      WRITE(6,2000) QUESTION
      READ (5,1000) IYN
      WRITE(6,2001) IYN
      IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
         ASK = .TRUE.
      ELSE
         ASK = .FALSE.
      END IF
      RETURN
 1000 FORMAT (A)
 2000 FORMAT (1X,A,' (y,n)>')
 2001 FORMAT (4X,'You entered ',A)
      END
C=======================================================================
      INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
      CHARACTER* (*) STRING
      WRITE(6,2000) STRING
      READ (5,*) ASK_I
      RETURN
 2000 FORMAT (1X,A,'>')
      END
```

Copyright © Multipath Corporation

FMS EXAMPLE 21. GENERAL OUT-OF-CORE MATRIX MULTIPLY

Example 21. General Out-of-core Matrix Multiply

This example demonstrates the use of the ***FMS*** general matrix multiply routine FMSMM.

- ***FMS*** Module

RS = Real Symmetric

RN = Real Nonsymmetric

CH = Complex Hermitian

CS = Complex Symmetric

CN = Complex Nonsymmetric

- Size and type of matrices
- Any FMS Parameter

This example is designed to test the FMSMM subroutine.

```
C       E X A M P L E   21
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_21')
C
C       FMS Parameters:
        INTEGER IACCUM
C
C       FMS matrix and vector file attributes:
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
```

```
C       Input data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       Local variables:
        INTEGER     M, N, K
        INTEGER     IDTYPE, I, LX, LDISK, NV
        INTEGER     MBA, MBB, MBC, IORATE
        REAL*8      ERROR
        REAL*8      RALPHA(1)
        REAL*8      SIZE, TEMP
        REAL*8      TCPU_A1, TWALL_A1, TIO_A1
        REAL*8      TCPU_A2, TWALL_A2, TIO_A2
        REAL*8      TCPU_B1, TWALL_B1, TIO_B1
        REAL*8      TCPU_B2, TWALL_B2, TIO_B2
        COMPLEX*16  CALPHA(1)
        CHARACTER*1 TRANSA
        LOGICAL     IFILL
        DATA LOWEQ/-1/
        DATA LU0(1)  /0/
        DATA RALPHA(1)/1.0D0/
        DATA CALPHA(1)/(1.0D0,1.0D0)/
C
C       Common block to communicate with fill routines:
        COMMON /MYDATA/ERROR, IOPT
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
        CALL FMSIST ('MDATAU', 1)
        CALL FMSIST ('LOGTIM',    3)
        CALL FMSIST ('IPRMV' , 1026)
        CALL FMSIST ('IPRVV' , 1026)
    1   CONTINUE
        WRITE (6,*) 'The FMS modules are numbered as follows:'
        WRITE (6,*) '   1 = Real Symmetric'
```

```
      WRITE (6,*) '   2 = Real Nonsymmetric'
      WRITE (6,*) '   3 = Complex Hermitian'
      WRITE (6,*) '   4 = Complex Symmetric'
      WRITE (6,*) '   5 = Complex Nonsymmetric'
      MOD = ASK_I('Enter the FMS module number (1 to 5)')
      IF( (MOD.LT.1) .OR. (MOD.GT.5) ) THEN
         PRINT *,'Illegal FMS Module entered.  Try again.'
         GO TO 1
      END IF
    2 CONTINUE
      WRITE (6,*) 'The following cases are available:'
      WRITE (6,*) '   1 = {C} = {D} + s*[A]{B}'
      WRITE (6,*) '   3 = {C} = {D} + s*{A}{B}'
      WRITE (6,*) '   4 = {C} = {D} + s*{A}T{B}'
      WRITE (6,*) '   5 = [C] = [D] + s*{A}T{B}'
      ICASE = ASK_I('Enter the case number (1 to 5)')
      IF( (ICASE.LT.1) .OR. (ICASE.GT.5) ) THEN
         PRINT *,'Illegal case entered.  Try again.'
         GO TO 2
      END IF
      M = ASK_I('Enter the number of rows    in C')
      IF(ICASE .NE. 5) THEN
         N = ASK_I('Enter the number of columns in C')
      ELSE
         N = M
      END IF
      IF( (ICASE .NE. 1)   .AND.
     1    (ICASE .NE. 2) ) THEN
         K = ASK_I('Enter the number of rows    in {B}')
      ELSE
         K = M
      END IF
      IF(ICASE .EQ. 1) THEN
         WRITE (6,2001) MOD, M, N, M, M, M, N
      ELSE IF (ICASE .EQ. 3) THEN
         WRITE (6,2003) MOD, M, N, M, K, K, N
      ELSE IF (ICASE .EQ. 4) THEN
         WRITE (6,2004) MOD, M, N, K, M, K, N
      ELSE
```

```
            WRITE (6,2005) MOD, M, N, K, M, K, N
         END IF
         IFILL = ASK('Do you want to fill')
         WRITE (6,*) 'You may now alter any FMS parameter.'
         WRITE (6,*) 'When you are finished, type the letters RETURN'
         CALL FMSSET
         CALL FMSIGT ('MEMPTR', IMD_PTR)
         CALL FMSIGT ('MEMPTR', RMD_PTR)
         CALL FMSIGT ('MEMPTR', CMD_PTR)
         IF(MOD .LE. 2) THEN
            IDTYPE = 1
         ELSE
            IDTYPE = 2
         END IF
         IF(ICASE .LT. 4) THEN
            TRANSA = 'N'
         ELSE
            TRANSA = 'T'
         END IF
C (2)    Open FMS files:
C
C        LUA:
         IF(ICASE .EQ. 1) THEN
C           This is a matrix file:
            WRITE(6,*) 'Opening matrix file [A]:'
            IF(MOD.EQ.1) CALL RSDI (LOWEQ, M, 'LUA', LUA)
            IF(MOD.EQ.2) CALL RNDI (LOWEQ, M, 'LUA', LUA)
            IF(MOD.EQ.3) CALL CHDI (LOWEQ, M, 'LUA', LUA)
            IF(MOD.EQ.4) CALL CSDI (LOWEQ, M, 'LUA', LUA)
            IF(MOD.EQ.5) CALL CNDI (LOWEQ, M, 'LUA', LUA)
            SIZE = DFLOAT(LUA(5)) * DFLOAT(LUA(10))
            IF(LUA(11) .EQ. 2) SIZE = 2.0*SIZE
            MBA = (SIZE + 131071.)/131072.
            WRITE(6,*) '   Storage for [A] (Mbytes)=', MBA
         ELSE
C           This is a vector file:
            WRITE(6,*) 'Opening vector file {A}:'
            IF(TRANSA .EQ. 'N') THEN
C              {A} is not transposed:
```

```
            CALL FMSOV (M, IDTYPE, K, 'LUA', LUA)
         ELSE
C           {A} is transposed:
            CALL FMSOV (K, IDTYPE, M, 'LUA', LUA)
         END IF
         SIZE = DFLOAT(LUA(4)) * DFLOAT(LUA(6))
         MBA = (SIZE + 131071.)/131072.
         WRITE(6,*) '   Storage for {A} (Mbytes)=', MBA
      END IF
C
C     LUB:
      WRITE(6,*) 'Opening vector file {B}:'
      CALL FMSOV (K, IDTYPE, N, 'LUB', LUB)
      SIZE = DFLOAT(LUB(4)) * DFLOAT(LUB(6))
      MBB = (SIZE + 131071.)/131072.
      WRITE(6,*) '   Storage for {B} (Mbytes)=', MBB
C
C     LUC:
      IF(ICASE .EQ. 5) THEN
C        This is a matrix file:
         WRITE(6,*) 'Opening matrix file [C]:'
         IF(MOD.EQ.1) CALL RSDI (LOWEQ, M, 'LUC', LUC)
         IF(MOD.EQ.2) CALL RNDI (LOWEQ, M, 'LUC', LUC)
         IF(MOD.EQ.3) CALL CHDI (LOWEQ, M, 'LUC', LUC)
         IF(MOD.EQ.4) CALL CSDI (LOWEQ, M, 'LUC', LUC)
         IF(MOD.EQ.5) CALL CSDN (LOWEQ, M, 'LUC', LUC)
         SIZE = DFLOAT(LUC(5)) * DFLOAT(LUC(10))
         IF(LUC(11) .EQ. 2) SIZE = 2.0*SIZE
         MBC = (SIZE + 131071.)/131072.
         WRITE(6,*) '   Storage for [C] (Mbytes)=', MBA
      ELSE
C        This is a vector file:
         WRITE(6,*) 'Opening vector file {C}:'
         CALL FMSOV (M, IDTYPE, N, 'LUC', LUC)
         SIZE = DFLOAT(LUC(4)) * DFLOAT(LUC(6))
         MBC = (SIZE + 131071.)/131072.
         WRITE(6,*) '   Storage for {C} (Mbytes)=', MBC
      END IF
C
```

```
C (3)   Write data to FMS files:
        IF(IFILL) THEN
C
C       {A}
        CALL FMSTIM (TCPU_A1, TWALL_A1, TIO_A1)
        IOPT = 0
        IF(ICASE .EQ. 1) THEN
C          This is a matrix file:
           WRITE(6,*) 'Writing data to matrix file [A]:'
           IF(MOD.EQ.1) CALL RSDA (LU0, 0, LUA)
           IF(MOD.EQ.2) CALL RNDA (LU0, 0, LUA)
           IF(MOD.EQ.3) CALL CHDA (LU0, 0, LUA)
           IF(MOD.EQ.4) CALL CSDA (LU0, 0, LUA)
           IF(MOD.EQ.5) CALL CNDA (LU0, 0, LUA)
        ELSE
C          This is a vector file:
           WRITE(6,*) 'Writing data to vector file {A}:'
           CALL VFILL (LUA)
        END IF
        CALL FMSTIM (TCPU_A2, TWALL_A2, TIO_A2)
        TEMP = TWALL_A2 - TWALL_A1
        IF(TEMP .LT. .01) TEMP=.01
        TEMP = DFLOAT(MBA)/TEMP
        IORATE = TEMP
        WRITE(6,*) '   Transfer rate  (Mb/Sec.)=', IORATE
C
C       {B}
        WRITE(6,*) 'Writing data to vector file {B}:'
        CALL FMSTIM (TCPU_B1, TWALL_B1, TIO_B1)
        CALL VFILL (LUB)
        END IF
        CALL FMSTIM (TCPU_B2, TWALL_B2, TIO_B2)
        TEMP = TWALL_B2 - TWALL_B1
        IF(TEMP .LT. .01) TEMP=.01
        TEMP = DFLOAT(MBB)/TEMP
        IORATE = TEMP
        WRITE(6,*) '   Transfer rate  (Mb/Sec.)=', IORATE
C
C (4)   Perform matrix algebra:
```

```
         WRITE(6,*) 'Calling FMSMM'
C        Compute {C} = {A}{B}:
         CALL FMSMM (LUC, 0, 0, TRANSA, LUA, LUB)
C        Compute {C} = {C} - {A}{B}:
         CALL FMSMM (LUC, LUC, 0, TRANSA, LUA, LUB)
C
C (5)    Read data from FMS files:
C        Check the answer; {C} should be {0}:
         IOPT  = 1
         ERROR = 0.0D0
         IF(ICASE .EQ. 5) THEN
C           [C]is a matrix file:
            WRITE(6,*) 'Checking matrix file [C]:'
            IF(MOD.EQ.1) CALL RSDAF
     1         (LUC, RALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.2) CALL RNDAF (LU0, 0, LUC)
     1         (LUC, RALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.3) CALL CHDAF (LU0, 0, LUC)
     1         (LUC, RALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.4) CALL CSDAF (LU0, 0, LUC)
     1         (LUC, CALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.5) CALL CNDAF (LU0, 0, LUC)
     1         (LUC, CALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
         ELSE
C           {C} is a vector file:
            WRITE(6,*) 'Checking vector file {C}:'
            CALL VCHECK (LUC, ERROR)
         END IF
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
C
C        {A}
         IF(ICASE .EQ. 1) THEN
            CALL FMSCM (LUA)
         ELSE
            CALL FMSCV (LUA)
         END IF
```

```
C
C       {B}
        CALL FMSCV (LUB)
C
C       {C}
        IF(ICASE .EQ. 5) THEN
           CALL FMSCM (LUC)
        ELSE
           CALL FMSCV (LUC)
        END IF
        IF(ASK('Do you want another solution?')) GO TO 1
        CALL FMSPOP(MYNAME)
        CALL FMSEND
 2001   FORMAT (/'You selected case 1, MOD',I2/
     1   '   C{',I7,',',I7,'} = A[',I7,',',I7,'] * B{',I7,',',I7,'}')
 2003   FORMAT (/'You selected case 3, MOD',I2/
     1   '   C{',I7,',',I7,'} = A{',I7,',',I7,'} * B{',I7,',',I7,'}')
 2004   FORMAT (/'You selected case 4, MOD',I2/
     1   '   C{',I7,',',I7,'} = A{',I7,',',I7,'}T * B{',I7,',',I7,'}')
 2005   FORMAT (/'You selected case 5, MOD',I2/
     1   '   C[',I7,',',I7,'] = A{',I7,',',I7,'}T * B{',I7,',',I7,'}')
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
```

```
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END

C       ======================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ======================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*)
        COMMON /MYDATA/ERROR, IOPT
        REAL*8 ERROR
        REAL*8 EI
C
        IF(IOPT .EQ. 0) THEN
C          Populate the diagonal with test data:
           IF(IROW2 .EQ. JCOL2) THEN
C             This is a diagonal block:
              DO I = IROW1,IROW2
                 D(I) = RANAIJ(I,I)
              END DO
           END IF
C
C          Populate [AL] with random numbers:
C          The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
           DO I = IROW1,IROW2
              J1 = MAX0(LOWEQ(I),JCOL1)
              J2 = MIN0(I-1,JCOL2)
              L  = LOCEQ(I) + IJSTEP*J1
              DO J = J1,J2
                 A(L) = RANAIJ(I,J)
                 L = L + IJSTEP
              END DO
```

```
            END DO
         ELSE
C           Determine the maximum element in [AL]
            IF(IROW2 .EQ. JCOL2) THEN
C              This is a diagonal block:
               DO I = IROW1,IROW2
                  EI = ABS(D(I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END IF
            DO I = IROW1,IROW2
               J1 = MAX0(LOWEQ(I),JCOL1)
               J2 = MIN0(I-1,JCOL2)
               L  = LOCEQ(I) + IJSTEP*J1
               DO J = J1,J2
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
                  L = L + IJSTEP
               END DO
            END DO
         END IF
         RETURN
         END
C        ================================================================
         SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*)
         COMMON /MYDATA/ERROR, IOPT
         REAL*8     ERROR
         REAL*8     EI
C
         IF(IOPT .EQ. 0) THEN
C           Populate [AU] with test data:
C           The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
            DO J = JCOL1,JCOL2
               I1 = MAX0(LOWEQ(J),IROW1)
```

```
               I2 = MIN0(J-1,IROW2)
               L  = LOCEQ(J) + IJSTEP*I1
               DO I = I1,I2
                  A(L) = RANAIJ(I,J)
                  L = L + IJSTEP
               END DO
            END DO
         ELSE
C           Find the largest element in [AU]:
            DO J = JCOL1,JCOL2
               I1 = MAX0(LOWEQ(J),IROW1)
               I2 = MIN0(J-1,IROW2)
               L  = LOCEQ(J) + IJSTEP*I1
               DO I = I1,I2
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
                  L = L + IJSTEP
               END DO
            END DO
         END IF
         RETURN
         END
C        ================================================================
         SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     D(*)
         COMPLEX*16 A(0:*)
         COMMON /MYDATA/ERROR, IOPT
         REAL*8     ERROR
         REAL*8     EI
         REAL*8     AIJ
C
         IF(IOPT .EQ. 0) THEN
C           Populate the diagonal with test data:
            IF(IROW2 .EQ. JCOL2) THEN
C              This is a diagonal block:
```

```
            DO I = IROW1,IROW2
               D(I) = RANAIJ(I,I)
            END DO
         END IF
C
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
         DO I = IROW1,IROW2
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            L  = LOCEQ(I) + IJSTEP*J1
            DO J = J1,J2
               AIJ  = RANAIJ(I,J)
               A(L) = DCMPLX(AIJ,AIJ)
               L = L + IJSTEP
            END DO
         END DO
      ELSE
C        Determine the maximum element in [AL]:
         IF(IROW2 .EQ. JCOL2) THEN
C           This is a diagonal block:
            DO I = IROW1,IROW2
               EI = ABS(D(I))
               IF(EI .GT. ERROR) ERROR = EI
            END DO
         END IF
C
         DO I = IROW1,IROW2
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            L  = LOCEQ(I) + IJSTEP*J1
            DO J = J1,J2
               EI = ABS(A(L))
               IF(EI .GT. ERROR) ERROR = EI
               L = L + IJSTEP
            END DO
         END DO
      END IF
      RETURN
```

```
      END
C     ================================================================
      SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                   JCOL1, JCOL2, IJSTEP)
C     ================================================================
      INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
      INTEGER    LOWEQ(*), LOCEQ(*)
      COMPLEX*16 A(0:*), D(*)
      COMMON /MYDATA/ERROR, IOPT
      REAL*8     ERROR
      REAL*8     EI
      REAL*8     AIJ
C
      IF(IOPT .EQ. 0) THEN
C        Populate the diagonal with test data:
         IF(IROW2 .EQ. JCOL2) THEN
C           This is a diagonal block:
            DO I = IROW1,IROW2
               AIJ  = RANAIJ(I,I)
               D(I) = DCMPLX(AIJ,AIJ)
            END DO
         END IF
C
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
         DO I = IROW1,IROW2
            J1 = MAX0(LOWEQ(I),JCOL1)
            J2 = MIN0(I-1,JCOL2)
            L  = LOCEQ(I) + IJSTEP*J1
            DO J = J1,J2
               AIJ  = RANAIJ(I,J)
               A(L) = DCMPLX(AIJ,AIJ)
               L = L + IJSTEP
            END DO
         END DO
      ELSE
C        Find the maximum element in [AL]:
         IF(IROW2 .EQ. JCOL2) THEN
```

```
C             This is a diagonal block:
              DO I = IROW1,IROW2
                 EI = ABS(D(I))
                 IF(EI .GT. ERROR) ERROR = EI
              END DO
           END IF
C
           DO I = IROW1,IROW2
              J1 = MAX0(LOWEQ(I),JCOL1)
              J2 = MIN0(I-1,JCOL2)
              L  = LOCEQ(I) + IJSTEP*J1
              DO J = J1,J2
                 EI = ABS(A(L))
                 IF(EI .GT. ERROR) ERROR = EI
                 L = L + IJSTEP
              END DO
           END DO
        END IF
        RETURN
        END
C       ================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*)
        COMMON /MYDATA/ERROR, IOPT
        REAL*8     EI
        REAL*8     ERROR
        REAL*8     AIJ
C
        IF(IOPT .EQ. 0) THEN
C          Populate profile of [AL] with test data:
C          The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
           DO J = JCOL1,JCOL2
              I1 = MAX0(LOWEQ(J),IROW1)
              I2 = MIN0(J-1,IROW2)
              L  = LOCEQ(J) + IJSTEP*I1
```

```
               DO I = I1,I2
                  AIJ  = RANAIJ(I,J)
                  A(L) = DCMPLX(AIJ,AIJ)
                  L = L + IJSTEP
               END DO
            END DO
         ELSE
C           Determine the maximum element in [AU]:
C           The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
            DO J = JCOL1,JCOL2
               I1 = MAX0(LOWEQ(J),IROW1)
               I2 = MIN0(J-1,IROW2)
               L  = LOCEQ(J) + IJSTEP*I1
               DO I = I1,I2
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
                  L = L + IJSTEP
               END DO
            END DO
         END IF
         RETURN
         END
C        ================================================================
         SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         REAL*8     A(0:*)
         COMMON /MYDATA/ERROR, IOPT
         REAL*8     ERROR
         REAL*8     EI
C
         IF(IOPT .EQ. 0) THEN
C           Fill [A] with random numbers:
            DO J = JEQN1,JEQN2
               DO I = 1,NUMEQ
                  L = LOCI(I) + LOCJ(J,LUFLAG(I))
                  A(L) = RANAIJ(I,J)
               END DO
```

```
            END DO
         ELSE
C           Find the maximum element in [A]:
            DO J = JEQN1,JEQN2
               DO I = 1,NUMEQ
                  L = LOCI(I) + LOCJ(J,LUFLAG(I))
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END DO
         END IF
         RETURN
         END
C        ================================================================
         SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C        ================================================================
         INTEGER    JEQN1, JEQN2, NUMEQ
         INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
         COMPLEX*16 A(0:*), DIA, OFFDIA
         PARAMETER  (DIA   =( 1.0D0, 1.0D0))
         PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
         COMMON /MYDATA/ERROR, IOPT
         REAL*8     ERROR
         REAL*8     EI
         REAL*8     AIJ
C
         IF(IOPT .EQ. 0) THEN
C           Fill [A] with random numbers:
            DO J = JEQN1,JEQN2
               DO I = 1,NUMEQ
                  L = LOCI(I) + LOCJ(J,LUFLAG(I))
                  AIJ  = RANAIJ(I,J)
                  A(L) = DCMPLX(AIJ,AIJ)
               END DO
            END DO
         ELSE
C           Find the maximum element in [A]:
            DO J = JEQN1,JEQN2
               DO I = 1,NUMEQ
```

```
                  L = LOCI(I) + LOCJ(J,LUFLAG(I))
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END DO
         END IF
         RETURN
         END
C=======================================================================
         SUBROUTINE VFILL (LUX)
C=======================================================================
C        Fill vector file with random data
         INTEGER LUX(25)
         CHARACTER*5 MYNAME
         PARAMETER (MYNAME='VFILL')
         INTEGER MAXMD
         INTEGER MDUSED
         INTEGER MDAVIL
         REAL*8  RAVIL
         REAL*8  RREQD
C
C        FMS memory management requires the following arrays:
         POINTER (RMD_PTR, RMD)
         REAL*8     RMD(0:1)
         CALL FMSPSH(MYNAME)
         CALL FMSIGT ('MEMPTR', RMD_PTR)
C
         NUMEQ  = LUX(3)
         LENVEC = LUX(4)
         IDTYPE = LUX(5)
         NUMVEC = LUX(6)
C
C        Allocate storage for a block of vectors:
C
C        How much memory is available?
         CALL FMSIGT ('MEMPTR', MAXMD)
         CALL FMSIGT ('MEMPTR', MDUSED)
         MDAVIL = MAXMD - MDUSED
         RAVIL  = DFLOAT(MDAVIL)
```

```
         RREQD  = DFLOAT(LENVEC)*DFLOAT(NUMVEC)
         IF(RAVIL .GT. RREQD) THEN
C           The memory available is larger than the vector file.
            MDAVIL = LENVEC*NUMVEC
         END IF
C
C        Allocate MDAVIL words of memory:
         CALL FMSRMG (RMD, LV1, MDAVIL)
C
C        Fill blocks of vectors with random numbers and write to LUX:
         NVINC  = MDAVIL/LENVEC
         LDISK  = 1
         DO NV1 = 1,NUMVEC,NVINC
            NV2 = MIN0(NV1+NVINC-1,NUMVEC)
            NVEC = NV2 - NV1 + 1
            LV  = LV1
            DO NV = NV1,NV2
               CALL XRAN (RMD(LV), IDTYPE, NUMEQ, NV)
               LV = LV + LENVEC
            END DO
            NWRITE = NVEC*LENVEC
            CALL FMSWRT (LUX(1), LDISK, RMD(LV1), NWRITE)
            LDISK = LDISK + NWRITE
         END DO
C
C        Return the memory
         CALL FMSRMR (RMD, LV1, MDAVIL)
         CALL FMSPOP(MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE VCHECK (LUX, ERROR)
C=======================================================================
C        Find the largest absolute value in file LUX
         INTEGER LUX(25)
         REAL*8  ERROR
         REAL*8  EI
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='VCHECK')
```

```
      INTEGER MAXMD
      INTEGER MDUSED
      INTEGER MDAVIL
      REAL*8  RAVIL
      REAL*8  RREQD
C
C     FMS memory management requires the following arrays:
      POINTER (RMD_PTR, RMD)
      REAL*8     RMD(0:1)
      CALL FMSPSH(MYNAME)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
C
      NUMEQ  = LUX(3)
      LENVEC = LUX(4)
      IDTYPE = LUX(5)
      NUMVEC = LUX(6)
C
C     Allocate storage for a block of vectors:
C
C     How much memory is available?
      CALL FMSIGT ('MEMPTR', MAXMD)
      CALL FMSIGT ('MEMPTR', MDUSED)
      MDAVIL = MAXMD - MDUSED
      RAVIL  = DFLOAT(MDAVIL)
      RREQD  = DFLOAT(LENVEC)*DFLOAT(NUMVEC)
      IF(RAVIL .GT. RREQD) THEN
C        The memory available is larger than the vector file.
         MDAVIL = LENVEC*NUMVEC
      END IF
C
C     Allocate MDAVIL words of memory:
      CALL FMSRMG (RMD, LV1, MDAVIL)
C
C     Read blocks of vectors:
      NVINC  = MDAVIL/LENVEC
      LDISK  = 1
      DO NV1 = 1,NUMVEC,NVINC
         NV2 = MIN0(NV1+NVINC-1,NUMVEC)
```

```
            NVEC = NV2 - NV1 + 1
            NREAD= NVEC*LENVEC
            CALL FMSRED (LUX(1), LDISK, RMD(LV1), NREAD)
            LDISK = LDISK + NREAD
            LV  = LV1
            DO NV = NV1,NV2
               LVI = LV
               IMAX = IDTYPE*NUMEQ
               DO I = 1,IMAX
                  EI = ABS(RMD(LVI))
                  IF(EI .GT. ERROR) ERROR = EI
                  LVI = LVI + 1
               END DO
               LV = LV + LENVEC
            END DO
         END DO
C
C        Return the memory
         CALL FMSRMR (RMD, LV1, MDAVIL)
         CALL FMSPOP(MYNAME)
         RETURN
         END
C=======================================================================
         FUNCTION RANAIJ (I,J)
C=======================================================================
         REAL*8    RANAIJ
         INTEGER   I,J
         INTEGER   IY
         INTEGER   ISEEDR, ISEEDC
         PARAMETER (ISEEDR=11111111)
         PARAMETER (ISEEDC=12345678)
         IY     = I*ISEEDR + J*ISEEDC
         IY     = MOD(IY,65536)
         IY     = IY * 11111 + 1
         RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
         RANAIJ = RANAIJ - 0.5D0
         RETURN
         END
C=======================================================================
```

```
        SUBROUTINE XRAN (X, IDTYPE, NROWS, NCOL)
C=======================================================================
        REAL*8    X(IDTYPE, NROWS)
        INTEGER   IDTYPE, NROWS, NCOL
        REAL*8    AIJ
        INTEGER   ISEEDR, ISEEDC
        PARAMETER (ISEEDR=11111111)
        PARAMETER (ISEEDC=12345678)
        DO IROW = 1,NROWS
           IY     = IROW*ISEEDR + NCOL*ISEEDC
           IY     = MOD(IY,65536)
           IY     = IY * 11111 + 1
           AIJ    = DFLOAT((IY/256)+1) / 2844374.0D0
           AIJ    = AIJ - 0.5D0
           X(1,IROW) = AIJ
           IF(IDTYPE .EQ. 2) X(2,IROW) = AIJ
        END DO
        RETURN
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   21
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_21')
C
C       FMS Parameters:
        INTEGER IACCUM
C
C       FMS matrix and vector file attributes:
        INTEGER     LUA(25), LUB(25), LUC(25), LU0(25)
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C
C       Input data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Profile vector for a full matrix:
        INTEGER LOWEQ(1)
C
C       Local variables:
        INTEGER     M, N, K
        INTEGER     IDTYPE, I, LX, LDISK, NV
        INTEGER     MBA, MBB, MBC, IORATE
        REAL*8      ERROR
        REAL*8      RALPHA(1)
        REAL*8      SIZE, TEMP
        REAL*8      TCPU_A1, TWALL_A1, TIO_A1
        REAL*8      TCPU_A2, TWALL_A2, TIO_A2
        REAL*8      TCPU_B1, TWALL_B1, TIO_B1
        REAL*8      TCPU_B2, TWALL_B2, TIO_B2
        COMPLEX*16  CALPHA(1)
        CHARACTER*1 TRANSA
        LOGICAL     IFILL
        DATA LOWEQ/-1/
        DATA LU0(1)  /0/
        DATA RALPHA(1)/1.0D0/
        DATA CALPHA(1)/(1.0D0,1.0D0)/
C
C       Common block to communicate with fill routines:
        COMMON /MYDATA/ERROR, IOPT
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
        CALL FMSIST ('MDATAU', 1)
        CALL FMSIST ('LOGTIM',    3)
        CALL FMSIST ('IPRMV' , 1026)
        CALL FMSIST ('IPRVV' , 1026)
    1   CONTINUE
        WRITE (6,*) 'The FMS modules are numbered as follows:'
        WRITE (6,*) '   1 = Real Symmetric'
        WRITE (6,*) '   2 = Real Nonsymmetric'
        WRITE (6,*) '   3 = Complex Hermitian'
        WRITE (6,*) '   4 = Complex Symmetric'
        WRITE (6,*) '   5 = Complex Nonsymmetric'
```

```
            MOD = ASK_I('Enter the FMS module number (1 to 5)')
            IF( (MOD.LT.1) .OR. (MOD.GT.5) ) THEN
               PRINT *,'Illegal FMS Module entered.  Try again.'
               GO TO 1
            END IF
         2  CONTINUE
            WRITE (6,*) 'The following cases are available:'
            WRITE (6,*) '   1 = {C} = {D} + s*[A]{B}'
            WRITE (6,*) '   3 = {C} = {D} + s*{A}{B}'
            WRITE (6,*) '   4 = {C} = {D} + s*{A}T{B}'
            WRITE (6,*) '   5 = [C] = [D] + s*{A}T{B}'
            ICASE = ASK_I('Enter the case number (1 to 5)')
            IF( (ICASE.LT.1) .OR. (ICASE.GT.5) ) THEN
               PRINT *,'Illegal case entered.  Try again.'
               GO TO 2
            END IF
            M = ASK_I('Enter the number of rows    in C')
            IF(ICASE .NE. 5) THEN
               N = ASK_I('Enter the number of columns in C')
            ELSE
               N = M
            END IF
            IF( (ICASE .NE. 1)   .AND.
           1      (ICASE .NE. 2) ) THEN
               K = ASK_I('Enter the number of rows    in {B}')
            ELSE
               K = M
            END IF
            IF(ICASE .EQ. 1) THEN
               WRITE (6,2001) MOD, M, N, M, M, M, N
            ELSE IF (ICASE .EQ. 3) THEN
               WRITE (6,2003) MOD, M, N, M, K, K, N
            ELSE IF (ICASE .EQ. 4) THEN
               WRITE (6,2004) MOD, M, N, K, M, K, N
            ELSE
               WRITE (6,2005) MOD, M, N, K, M, K, N
            END IF
            IFILL = ASK('Do you want to fill')
            WRITE (6,*) 'You may now alter any FMS parameter.'
            WRITE (6,*) 'When you are finished, type the letters RETURN'
            CALL FMSSET
            CALL FMSIGT ('MEMPTR', IMD_PTR)
            CALL FMSIGT ('MEMPTR', RMD_PTR)
            CALL FMSIGT ('MEMPTR', CMD_PTR)
            IF(MOD .LE. 2) THEN
               IDTYPE = 1
            ELSE
               IDTYPE = 2
            END IF
            IF(ICASE .LT. 4) THEN
               TRANSA = 'N'
            ELSE
               TRANSA = 'T'
            END IF
      C (2) Open FMS files:
      C
      C     LUA:
            IF(ICASE .EQ. 1) THEN
      C        This is a matrix file:
               WRITE(6,*) 'Opening matrix file [A]:'
               IF(MOD.EQ.1) CALL RSDI (LOWEQ, M, 'LUA', LUA)
               IF(MOD.EQ.2) CALL RNDI (LOWEQ, M, 'LUA', LUA)
               IF(MOD.EQ.3) CALL CHDI (LOWEQ, M, 'LUA', LUA)
               IF(MOD.EQ.4) CALL CSDI (LOWEQ, M, 'LUA', LUA)
               IF(MOD.EQ.5) CALL CNDI (LOWEQ, M, 'LUA', LUA)
```

```
            SIZE = DFLOAT(LUA(5)) * DFLOAT(LUA(10))
            IF(LUA(11) .EQ. 2) SIZE = 2.0*SIZE
            MBA = (SIZE + 131071.)/131072.
            WRITE(6,*) '   Storage for [A] (Mbytes)=', MBA
         ELSE
C           This is a vector file:
            WRITE(6,*) 'Opening vector file {A}:'
            IF(TRANSA .EQ. 'N') THEN
C              {A} is not transposed:
               CALL FMSOV (M, IDTYPE, K, 'LUA', LUA)
            ELSE
C              {A} is transposed:
               CALL FMSOV (K, IDTYPE, M, 'LUA', LUA)
            END IF
            SIZE = DFLOAT(LUA(4)) * DFLOAT(LUA(6))
            MBA = (SIZE + 131071.)/131072.
            WRITE(6,*) '   Storage for {A} (Mbytes)=', MBA
         END IF
C
C        LUB:
         WRITE(6,*) 'Opening vector file {B}:'
         CALL FMSOV (K, IDTYPE, N, 'LUB', LUB)
         SIZE = DFLOAT(LUB(4)) * DFLOAT(LUB(6))
         MBB = (SIZE + 131071.)/131072.
         WRITE(6,*) '   Storage for {B} (Mbytes)=', MBB
C
C        LUC:
         IF(ICASE .EQ. 5) THEN
C           This is a matrix file:
            WRITE(6,*) 'Opening matrix file [C]:'
            IF(MOD.EQ.1) CALL RSDI (LOWEQ, M, 'LUC', LUC)
            IF(MOD.EQ.2) CALL RNDI (LOWEQ, M, 'LUC', LUC)
            IF(MOD.EQ.3) CALL CHDI (LOWEQ, M, 'LUC', LUC)
            IF(MOD.EQ.4) CALL CSDI (LOWEQ, M, 'LUC', LUC)
            IF(MOD.EQ.5) CALL CNDI (LOWEQ, M, 'LUC', LUC)
            SIZE = DFLOAT(LUC(5)) * DFLOAT(LUC(10))
            IF(LUC(11) .EQ. 2) SIZE = 2.0*SIZE
            MBC = (SIZE + 131071.)/131072.
            WRITE(6,*) '   Storage for [C] (Mbytes)=', MBA
         ELSE
C           This is a vector file:
            WRITE(6,*) 'Opening vector file {C}:'
            CALL FMSOV (M, IDTYPE, N, 'LUC', LUC)
            SIZE = DFLOAT(LUC(4)) * DFLOAT(LUC(6))
            MBC = (SIZE + 131071.)/131072.
            WRITE(6,*) '   Storage for {C} (Mbytes)=', MBC
         END IF
C
C (3)    Write data to FMS files:
         IF(IFILL) THEN
C
C        {A}
         CALL FMSTIM (TCPU_A1, TWALL_A1, TIO_A1)
         IOPT = 0
         IF(ICASE .EQ. 1) THEN
C           This is a matrix file:
            WRITE(6,*) 'Writing data to matrix file [A]:'
            IF(MOD.EQ.1) CALL RSDA (LU0, 0, LUA)
            IF(MOD.EQ.2) CALL RNDA (LU0, 0, LUA)
            IF(MOD.EQ.3) CALL CHDA (LU0, 0, LUA)
            IF(MOD.EQ.4) CALL CSDA (LU0, 0, LUA)
            IF(MOD.EQ.5) CALL CNDA (LU0, 0, LUA)
         ELSE
C           This is a vector file:
            WRITE(6,*) 'Writing data to vector file {A}:'
```

```
            CALL VFILL (LUA)
         END IF
         CALL FMSTIM (TCPU_A2, TWALL_A2, TIO_A2)
         TEMP = TWALL_A2 - TWALL_A1
         IF(TEMP .LT. .01) TEMP=.01
         TEMP = DFLOAT(MBA)/TEMP
         IORATE = TEMP
         WRITE(6,*) '   Transfer rate  (Mb/Sec.)=', IORATE
C
C        {B}
         WRITE(6,*) 'Writing data to vector file {B}:'
         CALL FMSTIM (TCPU_B1, TWALL_B1, TIO_B1)
         CALL VFILL (LUB)
         END IF
         CALL FMSTIM (TCPU_B2, TWALL_B2, TIO_B2)
         TEMP = TWALL_B2 - TWALL_B1
         IF(TEMP .LT. .01) TEMP=.01
         TEMP = DFLOAT(MBB)/TEMP
         IORATE = TEMP
         WRITE(6,*) '   Transfer rate  (Mb/Sec.)=', IORATE
C
C (4)    Perform matrix algebra:
         WRITE(6,*) 'Calling FMSMM'
C        Compute {C} = {A}{B}:
         CALL FMSMM (LUC, LUC, 0, TRANSA, LUA, LUB)
C        Compute {C} = {C} - {A}{B}:
         CALL FMSMM (LUC, LUC, -1, TRANSA, LUA, LUB)
C
C (5)    Read data from FMS files:
C        Check the answer; {C} should be {0}:
         IOPT  = 1
         ERROR = 0.0D0
         IF(ICASE .EQ. 5) THEN
C           [C]is a matrix file:
            WRITE(6,*) 'Checking matrix file [C]:'
            IF(MOD.EQ.1) CALL RSDAF
     1         (LUC, RALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.2) CALL RNDAF
     1         (LUC, RALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.3) CALL CHDAF
     1         (LUC, RALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.4) CALL CSDAF
     1         (LUC, CALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
            IF(MOD.EQ.5) CALL CNDAF
     1         (LUC, CALPHA, 1, LU0, 0, LU0, LU0, LU0, LU0, 0)
         ELSE
C           {C} is a vector file:
            WRITE(6,*) 'Checking vector file {C}:'
            CALL VCHECK (LUC, ERROR)
         END IF
         WRITE(6,*) 'MAXIMUM ERROR =', ERROR
C
C (6)    Close FMS files:
C
C        {A}
         IF(ICASE .EQ. 1) THEN
            CALL FMSCM (LUA)
         ELSE
            CALL FMSCV (LUA)
         END IF
C
C        {B}
         CALL FMSCV (LUB)
C
```

```
C       {C}
        IF(ICASE .EQ. 5) THEN
           CALL FMSCM (LUC)
        ELSE
           CALL FMSCV (LUC)
        END IF
        IF(ASK('Do you want another solution?')) GO TO 1
        CALL FMSPOP (MYNAME)
        CALL FMSEND
 2001   FORMAT (/'You selected case 1, MOD',I2/
     1   '   C{',I7,',',I7,'} = A[',I7,',',I7,'] * B{',I7,',',I7,'}')
 2003   FORMAT (/'You selected case 3, MOD',I2/
     1   '   C{',I7,',',I7,'} = A{',I7,',',I7,'} * B{',I7,',',I7,'}')
 2004   FORMAT (/'You selected case 4, MOD',I2/
     1   '   C{',I7,',',I7,'} = A{',I7,',',I7,'}T * B{',I7,',',I7,'}')
 2005   FORMAT (/'You selected case 5, MOD',I2/
     1   '   C[',I7,',',I7,'] = A{',I7,',',I7,'}T * B{',I7,',',I7,'}')
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END

C       ================================================================
        SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                     JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        REAL*8     A(0:*), D(*)
        COMMON /MYDATA/ERROR, IOPT
        REAL*8 ERROR
        REAL*8 EI
C
        IF(IOPT .EQ. 0) THEN
C          Populate the diagonal with test data:
           IF(IROW2 .EQ. JCOL2) THEN
C             This is a diagonal block:
              DO I = IROW1,IROW2
                 D(I) = RANAIJ(I,I)
              END DO
           END IF
C
C          Populate [AL] with random numbers:
```

```
C           The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
            DO I = IROW1,IROW2
               J1 = MAX0(LOWEQ(I),JCOL1)
               J2 = MIN0(I-1,JCOL2)
               L  = LOCEQ(I) + IJSTEP*J1
               DO J = J1,J2
                  A(L) = RANAIJ(I,J)
                  L = L + IJSTEP
               END DO
            END DO
         ELSE
C           Determine the maximum element in [AL]
            IF(IROW2 .EQ. JCOL2) THEN
C              This is a diagonal block:
               DO I = IROW1,IROW2
                  EI = ABS(D(I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END IF
            DO I = IROW1,IROW2
               J1 = MAX0(LOWEQ(I),JCOL1)
               J2 = MIN0(I-1,JCOL2)
               L  = LOCEQ(I) + IJSTEP*J1
               DO J = J1,J2
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
                  L = L + IJSTEP
               END DO
            END DO
         END IF
         RETURN
         END
C        ================================================================
         SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     A(0:*), D(*)
         COMMON /MYDATA/ERROR, IOPT
         REAL*8     ERROR
         REAL*8     EI
C
         IF(IOPT .EQ. 0) THEN
C           Populate [AU] with test data:
C           The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
            DO J = JCOL1,JCOL2
               I1 = MAX0(LOWEQ(J),IROW1)
               I2 = MIN0(J-1,IROW2)
               L  = LOCEQ(J) + IJSTEP*I1
               DO I = I1,I2
                  A(L) = RANAIJ(I,J)
                  L = L + IJSTEP
               END DO
            END DO
         ELSE
C           Find the largest element in [AU]:
            DO J = JCOL1,JCOL2
               I1 = MAX0(LOWEQ(J),IROW1)
               I2 = MIN0(J-1,IROW2)
               L  = LOCEQ(J) + IJSTEP*I1
               DO I = I1,I2
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
                  L = L + IJSTEP
```

```
               END DO
            END DO
         END IF
         RETURN
         END
C        ================================================================
         SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C        ================================================================
         INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER    LOWEQ(*), LOCEQ(*)
         REAL*8     D(*)
         COMPLEX*16 A(0:*)
         COMMON /MYDATA/ERROR, IOPT
         REAL*8     ERROR
         REAL*8     EI
         REAL*8     AIJ
C
         IF(IOPT .EQ. 0) THEN
C           Populate the diagonal with test data:
            IF(IROW2 .EQ. JCOL2) THEN
C              This is a diagonal block:
               DO I = IROW1,IROW2
                  D(I) = RANAIJ(I,I)
               END DO
            END IF
C
C           Populate profile of [AL] with test data:
C           The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
            DO I = IROW1,IROW2
               J1 = MAX0(LOWEQ(I),JCOL1)
               J2 = MIN0(I-1,JCOL2)
               L  = LOCEQ(I) + IJSTEP*J1
               DO J = J1,J2
                  AIJ  = RANAIJ(I,J)
                  A(L) = DCMPLX(AIJ,AIJ)
                  L = L + IJSTEP
               END DO
            END DO
         ELSE
C           Determine the maximum element in [AL]:
            IF(IROW2 .EQ. JCOL2) THEN
C              This is a diagonal block:
               DO I = IROW1,IROW2
                  EI = ABS(D(I))
                  IF(EI .GT. ERROR) ERROR = EI
               END DO
            END IF
C
            DO I = IROW1,IROW2
               J1 = MAX0(LOWEQ(I),JCOL1)
               J2 = MIN0(I-1,JCOL2)
               L  = LOCEQ(I) + IJSTEP*J1
               DO J = J1,J2
                  EI = ABS(A(L))
                  IF(EI .GT. ERROR) ERROR = EI
                  L = L + IJSTEP
               END DO
            END DO
         END IF
         RETURN
         END
C        ================================================================
         SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
```

```
     1                    JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*)
        COMMON /MYDATA/ERROR, IOPT
        REAL*8     ERROR
        REAL*8     EI
        REAL*8     AIJ
C
        IF(IOPT .EQ. 0) THEN
C          Populate the diagonal with test data:
           IF(IROW2 .EQ. JCOL2) THEN
C             This is a diagonal block:
              DO I = IROW1,IROW2
                 AIJ  = RANAIJ(I,I)
                 D(I) = DCMPLX(AIJ,AIJ)
              END DO
           END IF
C
C          Populate profile of [AL] with test data:
C          The term A(I,J) is addressed as A(LOCEQ(I)+IJSTEP*J)
           DO I = IROW1,IROW2
              J1 = MAX0(LOWEQ(I),JCOL1)
              J2 = MIN0(I-1,JCOL2)
              L  = LOCEQ(I) + IJSTEP*J1
              DO J = J1,J2
                 AIJ  = RANAIJ(I,J)
                 A(L) = DCMPLX(AIJ,AIJ)
                 L = L + IJSTEP
              END DO
           END DO
        ELSE
C          Find the maximum element in [AL]:
           IF(IROW2 .EQ. JCOL2) THEN
C             This is a diagonal block:
              DO I = IROW1,IROW2
                 EI = ABS(D(I))
                 IF(EI .GT. ERROR) ERROR = EI
              END DO
           END IF
C
           DO I = IROW1,IROW2
              J1 = MAX0(LOWEQ(I),JCOL1)
              J2 = MIN0(I-1,JCOL2)
              L  = LOCEQ(I) + IJSTEP*J1
              DO J = J1,J2
                 EI = ABS(A(L))
                 IF(EI .GT. ERROR) ERROR = EI
                 L = L + IJSTEP
              END DO
           END DO
        END IF
        RETURN
        END
C       ================================================================
        SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                    JCOL1, JCOL2, IJSTEP)
C       ================================================================
        INTEGER    IROW1, IROW2, JCOL1, JCOL2, IJSTEP
        INTEGER    LOWEQ(*), LOCEQ(*)
        COMPLEX*16 A(0:*), D(*)
        COMMON /MYDATA/ERROR, IOPT
        REAL*8     EI
        REAL*8     ERROR
```

```
      REAL*8     AIJ
C
      IF(IOPT .EQ. 0) THEN
C        Populate profile of [AL] with test data:
C        The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
         DO J = JCOL1,JCOL2
            I1 = MAX0(LOWEQ(J),IROW1)
            I2 = MIN0(J-1,IROW2)
            L  = LOCEQ(J) + IJSTEP*I1
            DO I = I1,I2
               AIJ  = RANAIJ(I,J)
               A(L) = DCMPLX(AIJ,AIJ)
               L = L + IJSTEP
            END DO
         END DO
      ELSE
C        Determine the maximum element in [AU]:
C        The term A(I,J) is addressed as A(LOCEQ(J)+IJSTEP*I)
         DO J = JCOL1,JCOL2
            I1 = MAX0(LOWEQ(J),IROW1)
            I2 = MIN0(J-1,IROW2)
            L  = LOCEQ(J) + IJSTEP*I1
            DO I = I1,I2
               EI = ABS(A(L))
               IF(EI .GT. ERROR) ERROR = EI
               L = L + IJSTEP
            END DO
         END DO
      END IF
      RETURN
      END
C     ================================================================
      SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C     ================================================================
      INTEGER    JEQN1, JEQN2, NUMEQ
      INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
      REAL*8     A(0:*)
      COMMON /MYDATA/ERROR, IOPT
      REAL*8     ERROR
      REAL*8     EI
C
      IF(IOPT .EQ. 0) THEN
C        Fill [A] with random numbers:
         DO J = JEQN1,JEQN2
            DO I = 1,NUMEQ
               L = LOCI(I) + LOCJ(J,LUFLAG(I))
               A(L) = RANAIJ(I,J)
            END DO
         END DO
      ELSE
C        Find the maximum element in [A]:
         DO J = JEQN1,JEQN2
            DO I = 1,NUMEQ
               L = LOCI(I) + LOCJ(J,LUFLAG(I))
               EI = ABS(A(L))
               IF(EI .GT. ERROR) ERROR = EI
            END DO
         END DO
      END IF
      RETURN
      END
C     ================================================================
      SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JEQN1, JEQN2, NUMEQ)
C     ================================================================
      INTEGER    JEQN1, JEQN2, NUMEQ
```

```
      INTEGER    LOCI(NUMEQ), LOCJ(JEQN1:JEQN2,2), LUFLAG(NUMEQ)
      COMPLEX*16 A(0:*), DIA, OFFDIA
      PARAMETER  (DIA   =( 1.0D0, 1.0D0))
      PARAMETER  (OFFDIA=(-1.0D0,-1.0D0))
      COMMON /MYDATA/ERROR, IOPT
      REAL*8     ERROR
      REAL*8     EI
      REAL*8     AIJ
C
      IF(IOPT .EQ. 0) THEN
C        Fill [A] with random numbers:
         DO J = JEQN1,JEQN2
            DO I = 1,NUMEQ
               L = LOCI(I) + LOCJ(J,LUFLAG(I))
               AIJ  = RANAIJ(I,J)
               A(L) = DCMPLX(AIJ,AIJ)
            END DO
         END DO
      ELSE
C        Find the maximum element in [A]:
         DO J = JEQN1,JEQN2
            DO I = 1,NUMEQ
               L = LOCI(I) + LOCJ(J,LUFLAG(I))
               EI = ABS(A(L))
               IF(EI .GT. ERROR) ERROR = EI
            END DO
         END DO
      END IF
      RETURN
      END
C=======================================================================
      SUBROUTINE VFILL (LUX)
C=======================================================================
C     Fill vector file with random data
      INTEGER LUX(25)
      CHARACTER*5 MYNAME
      PARAMETER (MYNAME='VFILL')
      INTEGER MAXMD
      INTEGER MDUSED
      INTEGER MDAVIL
      REAL*8  RAVIL
      REAL*8  RREQD
C
C     FMS memory management requires the following arrays:
      POINTER (RMD_PTR, RMD)
      REAL*8     RMD(0:1)
      CALL FMSPSH (MYNAME)
      CALL FMSIGT ('MEMPTR', RMD_PTR)
C
      NUMEQ  = LUX(3)
      LENVEC = LUX(4)
      IDTYPE = LUX(5)
      NUMVEC = LUX(6)
C
C     Allocate storage for a block of vectors:
C
C     How much memory is available?
      CALL FMSIGT ('MAXMD', MAXMD)
      CALL FMSIGT ('MDUSED', MDUSED)
      MDAVIL = MAXMD - MDUSED
      RAVIL  = DFLOAT(MDAVIL)
      RREQD  = DFLOAT(LENVEC)*DFLOAT(NUMVEC)
      IF(RAVIL .GT. RREQD) THEN
C        The memory available is larger than the vector file.
```

```
            MDAVIL = LENVEC*NUMVEC
         END IF
C
C        Allocate MDAVIL words of memory:
         CALL FMSRMG (RMD, LV1, MDAVIL)
C
C        Fill blocks of vectors with random numbers and write to LUX:
         NVINC  = MDAVIL/LENVEC
         LDISK  = 1
         DO NV1 = 1,NUMVEC,NVINC
            NV2 = MIN0(NV1+NVINC-1,NUMVEC)
            NVEC = NV2 - NV1 + 1
            LV  = LV1
            DO NV = NV1,NV2
               CALL XRAN (RMD(LV), IDTYPE, NUMEQ, NV)
               LV = LV + LENVEC
            END DO
            NWRITE = NVEC*LENVEC
            CALL FMSWRT (LUX(1), LDISK, RMD(LV1), NWRITE)
            LDISK = LDISK + NWRITE
         END DO
C
C        Return the memory
         CALL FMSRMR (RMD, LV1, MDAVIL)
         CALL FMSPOP (MYNAME)
         RETURN
         END
C=======================================================================
         SUBROUTINE VCHECK (LUX, ERROR)
C=======================================================================
C        Find the largest absolute value in file LUX
         INTEGER LUX(25)
         REAL*8  ERROR
         REAL*8  EI
         CHARACTER*6 MYNAME
         PARAMETER (MYNAME='VCHECK')
         INTEGER MAXMD
         INTEGER MDUSED
         INTEGER MDAVIL
         REAL*8  RAVIL
         REAL*8  RREQD
C
C        FMS memory management requires the following arrays:
         POINTER (RMD_PTR, RMD)
         REAL*8     RMD(0:1)
         CALL FMSPSH (MYNAME)
         CALL FMSIGT ('MEMPTR', RMD_PTR)
C
         NUMEQ  = LUX(3)
         LENVEC = LUX(4)
         IDTYPE = LUX(5)
         NUMVEC = LUX(6)
C
C        Allocate storage for a block of vectors:
C
C        How much memory is available?
         CALL FMSIGT ('MAXMD', MAXMD)
         CALL FMSIGT ('MDUSED', MDUSED)
         MDAVIL = MAXMD - MDUSED
         RAVIL  = DFLOAT(MDAVIL)
         RREQD  = DFLOAT(LENVEC)*DFLOAT(NUMVEC)
         IF(RAVIL .GT. RREQD) THEN
C           The memory available is larger than the vector file.
            MDAVIL = LENVEC*NUMVEC
         END IF
```

```
C
C       Allocate MDAVIL words of memory:
        CALL FMSRMG (RMD, LV1, MDAVIL)
C
C       Read blocks of vectors:
        NVINC  = MDAVIL/LENVEC
        LDISK  = 1
        DO NV1 = 1,NUMVEC,NVINC
           NV2 = MIN0(NV1+NVINC-1,NUMVEC)
           NVEC = NV2 - NV1 + 1
           NREAD= NVEC*LENVEC
           CALL FMSRED (LUX(1), LDISK, RMD(LV1), NREAD)
           LDISK = LDISK + NREAD
           LV  = LV1
           DO NV = NV1,NV2
              LVI = LV
              IMAX = IDTYPE*NUMEQ
              DO I = 1,IMAX
                 EI = ABS(RMD(LVI))
                 IF(EI .GT. ERROR) ERROR = EI
                 LVI = LVI + 1
              END DO
              LV = LV + LENVEC
           END DO
        END DO
C
C       Return the memory
        CALL FMSRMR (RMD, LV1, MDAVIL)
        CALL FMSPOP (MYNAME)
        RETURN
        END
C=======================================================================
        FUNCTION RANAIJ (I,J)
C=======================================================================
        REAL*8    RANAIJ
        INTEGER   I,J
        INTEGER   IY
        INTEGER   ISEEDR, ISEEDC
        PARAMETER (ISEEDR=11111111)
        PARAMETER (ISEEDC=12345678)
        IY     = I*ISEEDR + J*ISEEDC
        IY     = MOD(IY,65536)
        IY     = IY * 11111 + 1
        RANAIJ = DFLOAT((IY/256)+1) / 2844374.0D0
        RANAIJ = RANAIJ - 0.5D0
        RETURN
        END
C=======================================================================
        SUBROUTINE XRAN (X, IDTYPE, NROWS, NCOL)
C=======================================================================
        REAL*8    X(IDTYPE, NROWS)
        INTEGER   IDTYPE, NROWS, NCOL
        REAL*8    AIJ
        INTEGER   ISEEDR, ISEEDC
        PARAMETER (ISEEDR=11111111)
        PARAMETER (ISEEDC=12345678)
        DO IROW = 1,NROWS
           IY     = IROW*ISEEDR + NCOL*ISEEDC
           IY     = MOD(IY,65536)
           IY     = IY * 11111 + 1
           AIJ    = DFLOAT((IY/256)+1) / 2844374.0D0
           AIJ    = AIJ - 0.5D0
           X(1,IROW) = AIJ
           IF(IDTYPE .EQ. 2) X(2,IROW) = AIJ
        END DO
```

```
        RETURN
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 22. IO PERFORMANCE TEST

This program measures the performance of the I/O system used for FMS files. The input parameters are:

- Size of transfer (Mbytes)
- Number of times to transfer
- Any FMS Parameter

```
C       E X A M P L E   22
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_22')
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       FMS vector file attributes:
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (RMD_PTR, RMD)
        REAL*8     RMD(0:1)
C
C       Local variables:
        INTEGER     LUPR
        INTEGER     MBYTES, NTIMES
        INTEGER     IDTYPE, I, LX, LDX, LDISK
        REAL*8      TCPU1,  TCPU2
        REAL*8      TWALL1, TWALL2
        REAL*8      TIO1,   TIO2
        REAL*8      TWRITE, TREAD, TMBYTE
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
```

```
    1   CONTINUE
        MBYTES = ASK_I('Enter the transfer length (Mbytes)')
        NTIMES = ASK_I('Enter the number of times to transfer')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', LUPR)
C
C (2)   Open FMS files:
        N      = 1024*1024*MBYTES/8
        IDTYPE = 1
        CALL FMSOV (N, IDTYPE, NTIMES, 'LUX', LUX)
        LDX    = LUX(4)
        TMBYTE = DFLOAT(8*LDX/(1024*1024)) * DFLOAT(NTIMES)
C
C       Allocate memory:
        CALL FMSRMG (RMD, LX, LDX)
C
C       Initialize the memory:
        DO I = 1,LDX
           RMD(LX+I-1) = DFLOAT(I)
        END DO
C
C (3)   Write data to FMS files:
C
        LDISK = 1
        WRITE(LUPR,2001) TMBYTE
        CALL FMSTIM (TCPU1, TWALL1, TIO1)
        DO I=1,NTIMES
           CALL FMSWRT (LUX(1), LDISK, RMD(LX), LDX)
           LDISK = LDISK + LDX
        END DO
        CALL FMSTIM (TCPU2, TWALL2, TIO2)
        TWRITE = TWALL2 - TWALL1
        IF(TWRITE .NE. 0) THEN
           TWRITE = TMBYTE/TWRITE
        ELSE
           TWRITE = 0.0D0
```

```
        END IF
C
C (4)   Perform matrix algebra:
C       None required.
C
C
C (5)   Read data from FMS files:
C
        LDISK = 1
        WRITE(LUPR,2002), TMBYTE
        CALL FMSTIM (TCPU1, TWALL1, TIO1)
        DO I=1,NTIMES
           CALL FMSRED (LUX(1), LDISK, RMD(LX), LDX)
           LDISK = LDISK + LDX
        END DO
        CALL FMSTIM (TCPU2, TWALL2, TIO2)
        TREAD  = TWALL2 - TWALL1
        IF(TREAD  .NE. 0) THEN
           TREAD  = TMBYTE/TREAD
        ELSE
           TREAD  = 0.0D0
        END IF
C
C       Print the results:
        WRITE(LUPR,2000) MBYTES, NTIMES, TWRITE, TREAD
C
C (6)   Close FMS files:
C
C       Return the work array:
        CALL FMSRMR (RMD, LX, LDX)
        CALL FMSCV (LUX)
        IF(ASK('Do you want another solution?')) GO TO 1
        CALL FMSPOP(MYNAME)
        CALL FMSEND
 2000   FORMAT (/
     1  ' <--Transfers-->    <--Rate(Mb/Sec)-->'/
     2  ' Mbytes    Times      Write       Read'/
     3  ' ======    =====    =======    ======='/
     4  I7,I9,2F11.3/)
```

```
 2001   FORMAT (/
     1  ' Writing ',F10.0,' Mbytes')
 2002   FORMAT (
     1  ' Reading ',F10.0,' Mbytes')
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        GO TO 20
   10   CONTINUE
        WRITE(6,2001)
   20   CONTINUE
        WRITE(6,2000) QUESTION
        READ (5,1000,ERR=10) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
 2001   FORMAT (1X,'Bad Input.  Try again.')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        GO TO 20
   10   CONTINUE
        WRITE(6,2001)
   20   CONTINUE
        WRITE(6,2000,ERR=10) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
 2001   FORMAT (1X,'Bad Input.  Try again.')
```

```
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   22
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_22')
C
C       Input Data:
        LOGICAL ASK
        INTEGER ASK_I
C
C       FMS vector file attributes:
        INTEGER     LUX(25)
C
C       FMS memory management requires the following arrays:
        POINTER (RMD_PTR, RMD)
        REAL*8     RMD(0:1)
C
C       Local variables:
        INTEGER     LUPR
        INTEGER     MBYTES, NTIMES
        INTEGER     IDTYPE, I, LX, LDX, LDISK
        REAL*8      TCPU1,  TCPU2
        REAL*8      TWALL1, TWALL2
        REAL*8      TIO1,   TIO2
        REAL*8      TWRITE, TREAD, TMBYTE
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH (MYNAME)
    1   CONTINUE
        MBYTES = ASK_I('Enter the transfer length (Mbytes)')
        NTIMES = ASK_I('Enter the number of times to transfer')
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('LUPR', LUPR)
C
C (2)   Open FMS files:
        N      = 1024*1024*MBYTES/8
        IDTYPE = 1
        CALL FMSOV (N, IDTYPE, NTIMES, 'LUX', LUX)
        LDX    = LUX(4)
        TMBYTE = DFLOAT(8*LDX/(1024*1024)) * DFLOAT(NTIMES)
C
C       Allocate memory:
        CALL FMSRMG (RMD, LX, LDX)
C
C       Initialize the memory:
        DO I = 1,LDX
           RMD(LX+I-1) = DFLOAT(I)
        END DO
C
C (3)   Write data to FMS files:
C
        LDISK = 1
        WRITE(LUPR,2001) TMBYTE
        CALL FMSTIM (TCPU1, TWALL1, TIO1)
        DO I=1,NTIMES
           CALL FMSWRT (LUX(1), LDISK, RMD(LX), LDX)
           LDISK = LDISK + LDX
        END DO
        CALL FMSTIM (TCPU2, TWALL2, TIO2)
```

```
         TWRITE = TWALL2 - TWALL1
         IF(TWRITE .NE. 0) THEN
            TWRITE = TMBYTE/TWRITE
         ELSE
            TWRITE = 0.0D0
         END IF
C
C (4)    Perform matrix algebra:
C        None required.
C
C
C (5)    Read data from FMS files:
C
         LDISK = 1
         WRITE(LUPR,2002), TMBYTE
         CALL FMSTIM (TCPU1, TWALL1, TIO1)
         DO I=1,NTIMES
            CALL FMSRED (LUX(1), LDISK, RMD(LX), LDX)
            LDISK = LDISK + LDX
         END DO
         CALL FMSTIM (TCPU2, TWALL2, TIO2)
         TREAD  = TWALL2 - TWALL1
         IF(TREAD  .NE. 0) THEN
            TREAD  = TMBYTE/TREAD
         ELSE
            TREAD  = 0.0D0
         END IF
C
C        Print the results:
         WRITE(LUPR,2000) MBYTES, NTIMES, TWRITE, TREAD
C
C (6)    Close FMS files:
C
C        Return the work array:
         CALL FMSRMR (RMD, LX, LDX)
         CALL FMSCV (LUX)
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP (MYNAME)
         CALL FMSEND
 2000    FORMAT (/
     1   ' <--Transfers-->    <--Rate(Mb/Sec)-->'/
     2   ' Mbytes    Times      Write       Read'/
     3   ' ======    =====    =======    ======='/
     4   I7,I9,2F11.3/)
 2001    FORMAT (/
     1   ' Writing ',F10.0,' Mbytes')
 2002    FORMAT (
     1   ' Reading ',F10.0,' Mbytes')
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         GO TO 20
   10    CONTINUE
         WRITE(6,2001)
   20    CONTINUE
         WRITE(6,2000) QUESTION
         READ (5,1000,ERR=10) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
         END IF
```

```
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
 2001   FORMAT (1X,'Bad Input.  Try again.')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        GO TO 20
   10   CONTINUE
        WRITE(6,2001)
   20   CONTINUE
        WRITE(6,2000,ERR=10) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
 2001   FORMAT (1X,'Bad Input.  Try again.')
        END
```

Copyright © Multipath Corporation

FMS EXAMPLE 23. POPULATING TWO SYSTEMS CONCURRENTLY USING OPENMP

This example illustrates how to use ***FMS*** to build two systems of equations concurrently. The matrix and vector data is generated in parallel using FMS or OpenMP.

```
C-----------------------------------------------------------------------
C       DECLARATIONS
C-----------------------------------------------------------------------
C       E X A M P L E   23
C
C       FMS Parameters:
        INTEGER LUPR
        INTEGER MAXCPU
        INTEGER MAXMD
        INTEGER MDUSED
C
C       Subroutines called in parallel:
        EXTERNAL COLUMN
        EXTERNAL RHS
C
C       Name of this program:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_23')
C
C       Input Data functions:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Problem size parameters:
        INTEGER NUMEQ1, NUMEQ2, NRHS
C
C       Use OpenMP to fill matrix:
        LOGICAL OMP_FILL
C
C       Use OpenMP MUTEX locks:
```

```
      COMMON/MYDATA/OMP_MUTEX
      LOGICAL       OMP_MUTEX
C
C     Scale factor for input matrices to CNDAF:
      COMPLEX*16 ALPHA(1)
      DATA ALPHA(1)/(1.0D0,0.0D0)/
C
C     Number of initialization matrices:
      INTEGER NUMAI
      DATA NUMAI/1/
C
C     Number of submatrices:
      INTEGER NUMSF
      DATA NUMSF/0/
C
C     Number of vectors to reduce during factoring:
      INTEGER NUMRED
      DATA NUMRED/0/
C
C     Skip operations during solving (no):
      INTEGER ISKIP
      DATA ISKIP/0/
C
C     Dummy Complex argument to FMSCOL:
      COMPLEX*16 CDUMMY
      DATA CDUMMY/(0.0D0,0.0D0)/
C
C     FMS matrix and vector file attributes:
C     Matrix file:
      INTEGER LUA(25,2)
C     Vector file:
      INTEGER     LUX(25,2)
C     Dummy submatrix file:
      INTEGER     LUS(25)
C     Dummy output matrix file:\
      INTEGER     LUA0(25)
      DATA LUA0(1)/0/
C
C     Data type:
```

```
        INTEGER IDTYPE
C       Complex*16:
        PARAMETER (IDTYPE=2)
C
C       FMS profile vector:
C       LOWEQ(1)=-1 flags a full matrix:
        INTEGER LOWEQ(1)
        DATA LOWEQ/-1/
C
C       Constants used for this test matrix:
        COMPLEX*16 CZERO, CONE, ANSWER
        PARAMETER (CZERO  = (0.0D0, 0.0D0))
        PARAMETER (ANSWER = (0.5D0,-0.5D0))
C
C       Variables used to check answer:
        REAL*8 ERROR, ETEST
C
C       Local variables:
        INTEGER L_X, LENX
        INTEGER MDLEFT, MDINC
        INTEGER LOCD
        INTEGER LENVEC1, LENVEC2
C
C       Work queue variables (shared):
        INTEGER NXCOL, NXRHS
C
C       FMS memory management requires the following arrays:
        POINTER (CMD_PTR, CMD)
        POINTER (RMD_PTR, RMD)
        POINTER (IMD_PTR, IMD)
        COMPLEX*16 CMD(0:1)
        REAL*8     RMD(0:1)
        INTEGER    IMD(0:1)
C-----------------------------------------------------------------------
C (1)   Initialize FMS:
C-----------------------------------------------------------------------
        CALL FMSINI
        CALL FMSPSH (MYNAME)
```

```
        CALL FMSIGT ('LOGTIM', LOGTIM)
        IF(LOGTIM .LT. 3) CALL FMSIST ('LOGTIM',    3)
        CALL FMSIST ('IPRF'  , 1026)
C       Loop back to here to do next problem:
  100   CONTINUE
        NUMEQ1 = ASK_I('Enter the number of equations in matrix 1')
        NUMEQ2 = ASK_I('Enter the number of equations in matrix 2')
        NRHS   = ASK_I('Enter the number of solution vectors')
        OMP_FILL = ASK('Do you want to use OpenMP to generate data')
        IF(OMP_FILL) THEN
C          Can use OpenMP or FMS locks:
           OMP_MUTEX= ASK('Do you want to use OpenMP MUTEX locks')
        ELSE
C          Must use FMS locks:
           OMP_MUTEX = .FALSE.
        END IF
        WRITE (6,*) 'You may now alter any FMS parameter.'
        WRITE (6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        CALL FMSIGT ('LUPR', LUPR)
        CALL FMSIGT ('MAXCPU', MAXCPU)
C-----------------------------------------------------------------------
C (2)   Open FMS files:
C-----------------------------------------------------------------------
        CALL CNDI  (LOWEQ, NUMEQ1, 'Matrix1', LUA(1,1))
        CALL CNDI  (LOWEQ, NUMEQ2, 'Matrix2', LUA(1,2))
        PRINT *,'Opening Vector file 1'
        CALL FMSOV (NUMEQ1, IDTYPE, NRHS, 'Vectors1', LUX(1,1))
        PRINT *,'Opening Vector file 2'
        CALL FMSOV (NUMEQ2, IDTYPE, NRHS, 'Vectors2', LUX(1,2))
C-----------------------------------------------------------------------
C (3)   Write data to FMS files:
C-----------------------------------------------------------------------
C
C       Initialize FMSCOL for two matrices built simultaneously:
C
C       Divide the remaining memory between the two matrix files:
        CALL FMSIGT ('MAXMD', MAXMD)
```

```
         CALL FMSIGT ('MDUSED', MDUSED)
         MDLEFT = MAXMD - MDUSED
         MDINC  = MDLEFT/2
C        Save the existing value of MAXMD:
         MAXMD_S = MAXMD
C
C        Initialize FMSCOL for the first file:
         MAXMD = MDUSED + MDINC
         CALL FMSIST ('MAXMD', MAXMD)
         PRINT *,'Initializing FMSCOL for Matrix 1'
         CALL FMSCOL (-1, CDUMMY, LUA(1,1))
C
C        Initialize FMSCOL for the  second file:
         PRINT *,'Initializing FMSCOL for Matrix 2'
         CALL FMSIST ('MAXMD', MAXMD_S)
         CALL FMSCOL (-1, CDUMMY, LUA(1,2))
C
C        Generate matrix elements in parallel
C        ====================================
         WRITE(LUPR,2000)
         NXCOL = 0
         IF(OMP_FILL) THEN
C           Use OpenMP to fill in parallel:
!$OMP PARALLEL DO DEFAULT(SHARED) NUM_THREADS(MAXCPU)
            DO ICPU = 1,MAXCPU
               CALL COLUMN(LUA,NXCOL)
            END DO
!$OMP END PARALLEL DO
         ELSE
C           Use FMS to fill in parallel:
C           Loop over children processes:
            DO ICPU = 2,MAXCPU
               CALL FMSPAR (2, COLUMN, LUA, NXCOL)
            END DO
C
C           Start the children running:
            IF(MAXCPU .GT. 1) CALL FMSRUN
C
C           Do parent's part:
```

```
            CALL COLUMN(LUA,NXCOL)
C
C           Wait for the children to complete:
            IF(MAXCPU .GT. 1) CALL FMSYNC
         END IF
C
C        End FMSCOL:
         PRINT *,'Ending FMSCOL for Matrix 1'
         CALL FMSCOL (NUMEQ1+1, CDUMMY, LUA(1,1))
         PRINT *,'Ending FMSCOL for Matrix 2'
         CALL FMSCOL (NUMEQ2+1, CDUMMY, LUA(1,2))
C
C        Generate the RHS vectors in parallel
C        ====================================
         WRITE(LUPR,2001)
         NXRHS = 0
         IF(OMP_FILL) THEN
C           Use OpenMP to fill in parallel:
!$OMP PARALLEL DO DEFAULT(SHARED) NUM_THREADS(MAXCPU)
            DO ICPU = 1,MAXCPU
               CALL RHS(LUX,NXRHS)
            END DO
!$OMP END PARALLEL DO
         ELSE
C           Use FMS to fill in parallel:
C           Loop over children processes:
            DO ICPU = 2,MAXCPU
               CALL FMSPAR (2, RHS, LUX, NXRHS)
            END DO
C
C           Start the children running:
            IF(MAXCPU .GT. 1) CALL FMSRUN
C
C           Do parent's part:
            CALL RHS (LUX,NXRHS)
C
C           Wait for the children to complete:
            IF(MAXCPU .GT. 1) CALL FMSYNC
         END IF
```

```
C-----------------------------------------------------------------------
C (4)   Perform matrix algebra:
C-----------------------------------------------------------------------
        CALL CNDF (LUA(1,1), ALPHA, NUMAI, LUS, NUMSF, LUA0,
     1   LUA(1,1), LUX, LUX, NUMRED)
        CALL CNDF (LUA(1,2), ALPHA, NUMAI, LUS, NUMSF, LUA0,
     1   LUA(1,2), LUX, LUX, NUMRED)
        CALL CNDS (LUA(1,1), LUX(1,1), LUX(1,1), NRHS, ISKIP)
        CALL CNDS (LUA(1,2), LUX(1,2), LUX(1,2), NRHS, ISKIP)
C-----------------------------------------------------------------------
C (5)   Read data from FMS files:
C-----------------------------------------------------------------------
        LENVEC1 = LUX(4,1)
        LENVEC2 = LUX(4,2)
        IF(LENVEC1 .GT. LOENVEC2) THEN
           LENX = LENVEC1/2
        ELSE
           LENX = LENVEC2/2
        END IF
        CALL FMSCMG (CMD, L_X, LENX)
C
C       Check system 1:
        LOCD = 1
        ERROR = 0.0D0
        DO IVEC = 1,NRHS
           CALL FMSRED (LUX(1,1), LOCD, CMD(L_X), LENVEC1)
           LOCD  = LOCD + LUX(4,1)
           DO I=1,NUMEQ1
              ETEST = ABS( CMD(L_X + I - 1) - ANSWER )
              IF(ETEST .GT. ERROR) ERROR = ETEST
           END DO
        END DO
        PRINT *,'MAXIMUM ERROR IN SYSTEM 1=', ERROR
C
C       Check system 2:
        LOCD = 1
        ERROR = 0.0D0
        DO IVEC = 1,NRHS
           CALL FMSRED (LUX(1,2), LOCD, CMD(L_X), LENVEC2)
```

```
            LOCD  = LOCD + LUX(4,2)
            DO I=1,NUMEQ2
               ETEST = ABS( CMD(L_X + I - 1) - ANSWER )
               IF(ETEST .GT. ERROR) ERROR = ETEST
            END DO
         END DO
         PRINT *,'MAXIMUM ERROR IN SYTSTEM 2=', ERROR
         CALL FMSCMR (CMD, L_X, LENX)
C-----------------------------------------------------------------------
C (6)    End FMS:
C-----------------------------------------------------------------------
C        Do the next problem.
         CALL FMSCV (LUX(1,1))
         CALL FMSCV (LUX(1,2))
         CALL FMSCM (LUA(1,1))
         CALL FMSCM (LUA(1,2))
         IF(ASK('Do you want another solution?')) GO TO 100
         CALL FMSPOP (MYNAME)
         CALL FMSEND
C-----------------------------------------------------------------------
C        FORMAT STATEMENTS
C-----------------------------------------------------------------------
 2000    FORMAT (/
     1   ' Writing the Columns in parallel'/
     2   ' ==============================')
 2001    FORMAT (/
     1   ' Writing the RHS vectors in parallel'/
     2   ' ==================================')
         END
C=======================================================================
         SUBROUTINE COLUMN (LUA, NXCOL)
C-----------------------------------------------------------------------
C        DESCRIPTION:
C           This subroutine computes the matrix elements.
C           It is designed to be run in parallel.
C
C        FORMAL PARAMETERS:
C           (R ) LUA(25,2) = Matrix file attribute lists
C
```

```
C          (RW) NXCOL = Next column to process (shared)
C-----------------------------------------------------------------------
C       Formal Parameters
C-----------------------------------------------------------------------
        INTEGER    LUA(25,2)
        INTEGER    NXCOL
C-----------------------------------------------------------------------
C       Local Variables
C-----------------------------------------------------------------------
        INTEGER    MYCOL
        INTEGER    NUMEQ1, NUMEQ2
        INTEGER    L_A1, L_A2
        LOGICAL    IDO_1, IDO_2
        INTEGER    MY_TOTAL
        INTEGER    MYNODE, LUPR
        COMPLEX*16 CZERO, DIA, OFFDIA
        COMMON/MYDATA/OMP_MUTEX
        LOGICAL       OMP_MUTEX
        DATA CZERO /( 0.0D0, 0.0D0)/
        DATA DIA   /( 1.0D0, 1.0D0)/
        DATA OFFDIA/(-1.0D0,-1.0D0)/
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='COLUMN')
        COMPLEX*16 CMD(0:1)
        POINTER (CMD_PTR,CMD)
C-----------------------------------------------------------------------
        CALL FMSPSH (MYNAME)
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('LUPR', LUPR)
        IDO_1 = .TRUE.
        IDO_2 = .TRUE.
        MY_TOTAL = 0
        NUMEQ1 = LUA(8,1)
        NUMEQ2 = LUA(8,2)
C
C       Get temporary storage to hold a column:
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        CALL FMSCMG (CMD, L_A1, NUMEQ1)
```

```
         CALL FMSCMG (CMD, L_A2, NUMEQ2)
C
C        Loop over columns:
  100    CONTINUE
C           Get your next column number:
C           Critical Section
C           ================
            IF(OMP_MUTEX) THEN
!$OMP CRITICAL
               MYCOL = INTINC(NXCOL)
!$OMP END CRITICAL
            ELSE
               CALL FMSONE
               MYCOL = INTINC(NXCOL)
               CALL FMSALL
            END IF
C           End of Critical Section
C           ======================
            IF(MYCOL .GT. NUMEQ1) IDO_1 = .FALSE.
            IF(MYCOL .GT. NUMEQ2) IDO_2 = .FALSE.
            IF((.NOT.IDO_1) .AND.
     1         (.NOT.IDO_2) ) THEN
C              This process is done.
               CALL FMSCMR (CMD, L_A1, NUMEQ1)
               CALL FMSCMR (CMD, L_A2, NUMEQ2)
C              Report your total work:
               IF(OMP_MUTEX) THEN
!$OMP CRITICAL
                  WRITE(LUPR,2000) MYNODE, MY_TOTAL
!$OMP END CRITICAL
               ELSE
                  CALL FMSONE
                  WRITE(LUPR,2000) MYNODE, MY_TOTAL
                  CALL FMSALL
               END IF
               CALL FMSPOP (MYNAME)
               RETURN
            ELSE
               MY_TOTAL = MY_TOTAL + 1
```

```
            END IF
C
            IF(IDO_1) THEN
               IF(MYCOL .EQ. 1) THEN
                  CMD(L_A1) = DCMPLX(NUMEQ1,NUMEQ1)
                  DO I=2,NUMEQ1
                     CMD(L_A1+I-1) = OFFDIA
                  END DO
               ELSE
                  CMD(L_A1) = OFFDIA
                  DO I=2,NUMEQ1
                     CMD(L_A1+I-1) = CZERO
                  END DO
                  CMD(L_A1+MYCOL-1) = DIA
               END IF
               CALL FMSCOL (MYCOL, CMD(L_A1), LUA(1,1))
            END IF
C
            IF(IDO_2) THEN
               IF(MYCOL .EQ. 1) THEN
                  CMD(L_A2) = DCMPLX(NUMEQ2,NUMEQ2)
                  DO I=2,NUMEQ2
                     CMD(L_A2+I-1) = OFFDIA
                  END DO
               ELSE
                  CMD(L_A2) = OFFDIA
                  DO I=2,NUMEQ2
                     CMD(L_A2+I-1) = CZERO
                  END DO
                  CMD(L_A2+MYCOL-1) = DIA
               END IF
               CALL FMSCOL (MYCOL, CMD(L_A2), LUA(1,2))
            END IF
C
C           Do the next column:
         GO TO 100
 2000    FORMAT (' Process',I3,' computed',I5,' Columns.')
         END
C=======================================================================
```

```
        SUBROUTINE RHS (LUX, NXRHS)
C=======================================================================
C
C       DESCRIPTION:
C          This subroutine computes the RHS vectors.
C          It is designed to be run in parallel.
C
C       FORMAL PARAMETERS:
C          (R ) LUX(25,2) = FMS vector file attributes
C          (RW) NXRHS     = Next RHS to process (shared)
C-----------------------------------------------------------------------
C       Formal Parameters:
C-----------------------------------------------------------------------
        INTEGER    LUX(25,2)
        INTEGER    NXRHS
C-----------------------------------------------------------------------
C       Local Variables
C-----------------------------------------------------------------------
        INTEGER    MYRHS
        LOGICAL    IDO_1, IDO_2
        INTEGER    MY_TOTAL
        INTEGER    MYNODE, LUPR
        CHARACTER*3 MYNAME
        PARAMETER  (MYNAME='RHS')
        INTEGER    NUMEQ1, NUMEQ2
        INTEGER    NUMVEC1, NUMVEC2
        COMMON/MYDATA/OMP_MUTEX
        LOGICAL       OMP_MUTEX
        COMPLEX*16 CZERO, CONE
        DATA CZERO/(0.0D0,0.0D0)/
        DATA CONE /(1.0D0,0.0D0)/
        COMPLEX*16 CMD(0:1)
        POINTER    (CMD_PTR,CMD)
C
        CALL FMSPSH (MYNAME)
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('LUPR', LUPR)
        IDO_1   = .TRUE.
        IDO_2   = .TRUE.
```

```
         MY_TOTAL = 0
         NUMEQ1  = LUX( 3,1)
         LENVEC1 = LUX( 4,1)
         NUMVEC1 = LUX( 6,1)
         NUMEQ2  = LUX( 3,2)
         LENVEC2 = LUX( 4,2)
         NUMVEC2 = LUX( 6,2)
C
C        Get temporary storage to hold the longest vector record:
         CALL FMSIGT ('MEMPTR', CMD_PTR)
         IF(LENVEC1 .GT. LENVEC2) THEN
            LENX = LENVEC1/2
         ELSE
            LENX = LENVEC2/2
         END IF
         CALL FMSCMG (CMD, L_X, LENX)
C
C        Populate the vector with test data:
         DO I=2,LENX
            CMD(L_X+I-1) = CZERO
         END DO
         CMD(L_X) = CONE
C
C        Loop over the RHS vectors:
  100    CONTINUE
C           Get your next RHS vector number:
C           Critical Section
C           ================
            IF(OMP_MUTEX) THEN
!$OMP CRITICAL
               MYRHS = INTINC(NXRHS)
!$OMP END CRITICAL
            ELSE
               CALL FMSONE
               MYRHS = INTINC(NXRHS)
               CALL FMSALL
            END IF
C           End of Critical Section
C           ======================
```

```
            IF(MYRHS .GT. NUMVEC1) IDO_1 = .FALSE.
            IF(MYRHS .GT. NUMVEC2) IDO_2 = .FALSE.
            IF((.NOT.IDO_1) .AND.
     1         (.NOT.IDO_2) ) THEN
C              This process is done.
               CALL FMSCMR (CMD, L_X, LENX)
C              Report your total work:
               IF(OMP_MUTEX) THEN
!$OMP CRITICAL
                  WRITE(LUPR,2000) MYNODE, MY_TOTAL
!$OMP END CRITICAL
               ELSE
                  CALL FMSONE
                  WRITE(LUPR,2000) MYNODE, MY_TOTAL
                  CALL FMSALL
               END IF
               CALL FMSPOP (MYNAME)
               RETURN
            ELSE
               MY_TOTAL = MY_TOTAL + 1
            END IF
C
            IF(IDO_1) THEN
               LDISK = 1 + LENVEC1*(MYRHS-1)
               CALL FMSWRT (LUX(1,1), LDISK, CMD(L_X), LENVEC1)
            END IF
C
            IF(IDO_2) THEN
               LDISK = 1 + LENVEC2*(MYRHS-1)
               CALL FMSWRT (LUX(1,1), LDISK, CMD(L_X), LENVEC2)
            END IF
         GO TO 100
 2000    FORMAT (' Process',I3,' computed',I5,' RHS vectors.')
         END
C=======================================================================
         INTEGER FUNCTION INTINC (I)
C-----------------------------------------------------------------------
C        This function increments a volatile shared variable.  It is
C        placed in a subroutine to prevent some compilers from storing
```

```
C       the value in a register and not updating it.
        INTEGER I
        I = I + 1
        INTINC = I
        RETURN
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

```
C-----------------------------------------------------------------------
C       DECLARATIONS
C-----------------------------------------------------------------------
C       E X A M P L E   23
C
C       FMS Parameters:
        INTEGER LUPR
        INTEGER MAXCPU
        INTEGER MAXMD
        INTEGER MDUSED
C
C       Subroutines called in parallel:
        EXTERNAL COLUMN
        EXTERNAL RHS
C
C       Name of this program:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_23')
C
C       Input Data functions:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Problem size parameters:
        INTEGER NUMEQ1, NUMEQ2, NRHS
C
C       Use OpenMP to fill matrix:
        LOGICAL OMP_FILL
C
C       Use OpenMP MUTEX locks:
        COMMON/MYDATA/OMP_MUTEX
        LOGICAL       OMP_MUTEX
C
C       Scale factor for input matrices to CNDAF:
        COMPLEX*16 ALPHA(1)
        DATA ALPHA(1)/(1.0D0,0.0D0)/
C
C       Number of initialization matrices:
        INTEGER NUMAI
        DATA NUMAI/1/
C
C       Number of submatrices:
        INTEGER NUMSF
        DATA NUMSF/0/
C
C       Number of vectors to reduce during factoring:
        INTEGER NUMRED
        DATA NUMRED/0/
C
C       Skip operations during solving (no):
        INTEGER ISKIP
        DATA ISKIP/0/
C
C       Dummy Complex argument to FMSCOL:
        COMPLEX*16 CDUMMY
        DATA CDUMMY/(0.0D0,0.0D0)/
C
C       FMS matrix and vector file attributes:
C       Matrix file:
        INTEGER LUA(25,2)
C       Vector file:
        INTEGER     LUX(25,2)
C       Dummy submatrix file:
```

```
      INTEGER     LUS(25)
C     Dummy output matrix file:\
      INTEGER     LUA0(25)
      DATA LUA0(1)/0/
C
C     Data type:
      INTEGER IDTYPE
C     Complex*16:
      PARAMETER (IDTYPE=2)
C
C     FMS profile vector:
C     LOWEQ(1)=-1 flags a full matrix:
      INTEGER LOWEQ(1)
      DATA LOWEQ/-1/
C
C     Constants used for this test matrix:
      COMPLEX*16 CZERO, CONE, ANSWER
      PARAMETER (CZERO  = (0.0D0, 0.0D0))
      PARAMETER (ANSWER = (0.5D0,-0.5D0))
C
C     Variables used to check answer:
      REAL*8 ERROR, ETEST
C
C     Local variables:
      INTEGER L_X, LENX
      INTEGER MDLEFT, MDINC
      INTEGER LOCD
      INTEGER LENVEC1, LENVEC2
C
C     Work queue variables (shared):
      INTEGER NXCOL, NXRHS
C
C     FMS memory management requires the following arrays:
      POINTER (CMD_PTR, CMD)
      POINTER (RMD_PTR, RMD)
      POINTER (IMD_PTR, IMD)
      COMPLEX*16 CMD(0:1)
      REAL*8     RMD(0:1)
      INTEGER    IMD(0:1)
C-----------------------------------------------------------------------
C (1)   Initialize FMS:
C-----------------------------------------------------------------------
      CALL FMSINI
      CALL FMSPSH (MYNAME)
      CALL FMSIGT ('LOGTIM', LOGTIM)
      IF(LOGTIM .LT. 3) CALL FMSIST ('LOGTIM',    3)
      CALL FMSIST ('IPRF'  , 1026)
C     Loop back to here to do next problem:
  100 CONTINUE
      NUMEQ1 = ASK_I('Enter the number of equations in matrix 1')
      NUMEQ2 = ASK_I('Enter the number of equations in matrix 2')
      NRHS   = ASK_I('Enter the number of solution vectors')
      OMP_FILL = ASK('Do you want to use OpenMP to generate data')
      IF(OMP_FILL) THEN
C        Can use OpenMP or FMS locks:
         OMP_MUTEX= ASK('Do you want to use OpenMP MUTEX locks')
      ELSE
C        Must use FMS locks:
         OMP_MUTEX = .FALSE.
      END IF
      WRITE (6,*) 'You may now alter any FMS parameter.'
      WRITE (6,*) 'When you are finished, type the letters RETURN'
      CALL FMSSET
      CALL FMSIGT ('MEMPTR', CMD_PTR)
      CALL FMSIGT ('LUPR', LUPR)
```

```
        CALL FMSIGT ('MAXCPU', MAXCPU)
C-----------------------------------------------------------------------
C (2)   Open FMS files:
C-----------------------------------------------------------------------
        CALL CNDI  (LOWEQ, NUMEQ1, 'Matrix1', LUA(1,1))
        CALL CNDI  (LOWEQ, NUMEQ2, 'Matrix2', LUA(1,2))
        PRINT *,'Opening Vector file 1'
        CALL FMSOV (NUMEQ1, IDTYPE, NRHS, 'Vectors1', LUX(1,1))
        PRINT *,'Opening Vector file 2'
        CALL FMSOV (NUMEQ2, IDTYPE, NRHS, 'Vectors2', LUX(1,2))
C-----------------------------------------------------------------------
C (3)   Write data to FMS files:
C-----------------------------------------------------------------------
C
C       Initialize FMSCOL for two matrices built simultaneously:
C
C       Divide the remaining memory between the two matrix files:
        CALL FMSIGT ('MAXMD', MAXMD)
        CALL FMSIGT ('MDUSED', MDUSED)
        MDLEFT = MAXMD - MDUSED
        MDINC  = MDLEFT/2
C       Save the existing value of MAXMD:
        MAXMD_S = MAXMD
C
C       Initialize FMSCOL for the first file:
        MAXMD = MDUSED + MDINC
        CALL FMSIST ('MAXMD',MAXMD)
        PRINT *,'Initializing FMSCOL for Matrix 1'
        CALL FMSCOL (-1, CDUMMY, LUA(1,1))
C
C       Initialize FMSCOL for the  second file:
        PRINT *,'Initializing FMSCOL for Matrix 2'
        CALL FMSIST ('MAXMD', MAXMD_S)
        CALL FMSCOL (-1, CDUMMY, LUA(1,2))
C
C       Generate matrix elements in parallel
C       ====================================
        WRITE(LUPR,2000)
        NXCOL = 0
        IF(OMP_FILL) THEN
C          Use OpenMP to fill in parallel:
!$OMP PARALLEL DO DEFAULT(SHARED) NUM_THREADS(MAXCPU)
           DO ICPU = 1,MAXCPU
              CALL COLUMN(LUA,NXCOL)
           END DO
!$OMP END PARALLEL DO
        ELSE
C          Use FMS to fill in parallel:
C          Loop over children processes:
           DO ICPU = 2,MAXCPU
              CALL FMSPAR (2, COLUMN, LUA, NXCOL)
           END DO
C
C          Start the children running:
           IF(MAXCPU .GT. 1) CALL FMSRUN
C
C          Do parent's part:
           CALL COLUMN(LUA,NXCOL)
C
C          Wait for the children to complete:
           IF(MAXCPU .GT. 1) CALL FMSYNC
        END IF
C
C       End FMSCOL:
        PRINT *,'Ending FMSCOL for Matrix 1'
```

```
         CALL FMSCOL (NUMEQ1+1, CDUMMY, LUA(1,1))
         PRINT *,'Ending FMSCOL for Matrix 2'
         CALL FMSCOL (NUMEQ2+1, CDUMMY, LUA(1,2))
C
C        Generate the RHS vectors in parallel
C        ====================================
         WRITE(LUPR,2001)
         NXRHS = 0
         IF(OMP_FILL) THEN
C           Use OpenMP to fill in parallel:
!$OMP PARALLEL DO DEFAULT(SHARED) NUM_THREADS(MAXCPU)
            DO ICPU = 1,MAXCPU
               CALL RHS(LUX,NXRHS)
            END DO
!$OMP END PARALLEL DO
         ELSE
C           Use FMS to fill in parallel:
C           Loop over children processes:
            DO ICPU = 2,MAXCPU
               CALL FMSPAR (2, RHS, LUX, NXRHS)
            END DO
C
C           Start the children running:
            IF(MAXCPU .GT. 1) CALL FMSRUN
C
C           Do parent's part:
            CALL RHS (LUX,NXRHS)
C
C           Wait for the children to complete:
            IF(MAXCPU .GT. 1) CALL FMSYNC
         END IF
C-----------------------------------------------------------------------
C (4)    Perform matrix algebra:
C-----------------------------------------------------------------------
         CALL CNDAF (LUA(1,1), ALPHA, NUMAI, LUS, NUMSF, LUA0,
     1    LUA(1,1), LUX, LUX, NUMRED)
         CALL CNDAF (LUA(1,2), ALPHA, NUMAI, LUS, NUMSF, LUA0,
     1    LUA(1,2), LUX, LUX, NUMRED)
         CALL CNDS (LUA(1,1), LUX(1,1), LUX(1,1), NRHS, ISKIP)
         CALL CNDS (LUA(1,2), LUX(1,2), LUX(1,2), NRHS, ISKIP)
C-----------------------------------------------------------------------
C (5)    Read data from FMS files:
C-----------------------------------------------------------------------
         LENVEC1 = LUX(4,1)
         LENVEC2 = LUX(4,2)
         IF(LENVEC1 .GT. LOENVEC2) THEN
            LENX = LENVEC1/2
         ELSE
            LENX = LENVEC2/2
         END IF
         CALL FMSCMG (CMD, L_X, LENX)
C
C        Check system 1:
         LOCD = 1
         ERROR = 0.0D0
         DO IVEC = 1,NRHS
            CALL FMSRED (LUX(1,1), LOCD, CMD(L_X), LENVEC1)
            LOCD  = LOCD + LUX(4,1)
            DO I=1,NUMEQ1
               ETEST = ABS( CMD(L_X + I - 1) - ANSWER )
               IF(ETEST .GT. ERROR) ERROR = ETEST
            END DO
         END DO
         PRINT *,'MAXIMUM ERROR IN SYSTEM 1=', ERROR
```

```
C
C       Check system 2:
        LOCD = 1
        ERROR = 0.0D0
        DO IVEC = 1,NRHS
           CALL FMSRED (LUX(1,2), LOCD, CMD(L_X), LENVEC2)
           LOCD  = LOCD + LUX(4,2)
           DO I=1,NUMEQ2
              ETEST = ABS( CMD(L_X + I - 1) - ANSWER )
              IF(ETEST .GT. ERROR) ERROR = ETEST
           END DO
        END DO
        PRINT *,'MAXIMUM ERROR IN SYTSTEM 2=', ERROR
        CALL FMSCMR (CMD, L_X, LENX)
C-----------------------------------------------------------------------
C (6)   End FMS:
C-----------------------------------------------------------------------
C       Do the next problem.
        CALL FMSCV (LUX(1,1))
        CALL FMSCV (LUX(1,2))
        CALL FMSCM (LUA(1,1))
        CALL FMSCM (LUA(1,2))
        IF(ASK('Do you want another solution?')) GO TO 100
        CALL FMSPOP (MYNAME)
        CALL FMSEND
C-----------------------------------------------------------------------
C       FORMAT STATEMENTS
C-----------------------------------------------------------------------
 2000   FORMAT (/
     1  ' Writing the Columns in parallel'/
     2  ' ================================')
 2001   FORMAT (/
     1  ' Writing the RHS vectors in parallel'/
     2  ' ===================================')
        END
C=======================================================================
        SUBROUTINE COLUMN (LUA, NXCOL)
C-----------------------------------------------------------------------
C       DESCRIPTION:
C          This subroutine computes the matrix elements.
C          It is designed to be run in parallel.
C
C       FORMAL PARAMETERS:
C          (R ) LUA(25,2) = Matrix file attribute lists
C
C          (RW) NXCOL = Next column to process (shared)
C-----------------------------------------------------------------------
C       Formal Parameters
C-----------------------------------------------------------------------
        INTEGER    LUA(25,2)
        INTEGER    NXCOL
C-----------------------------------------------------------------------
C       Local Variables
C-----------------------------------------------------------------------
        INTEGER    MYCOL
        INTEGER    NUMEQ1, NUMEQ2
        INTEGER    L_A1, L_A2
        LOGICAL    IDO_1, IDO_2
        INTEGER    MY_TOTAL
        INTEGER    MYNODE, LUPR
        COMPLEX*16 CZERO, DIA, OFFDIA
        COMMON/MYDATA/OMP_MUTEX
        LOGICAL       OMP_MUTEX
        DATA CZERO /( 0.0D0, 0.0D0)/
        DATA DIA   /( 1.0D0, 1.0D0)/
```

```
        DATA OFFDIA/(-1.0D0,-1.0D0)/
        CHARACTER*6 MYNAME
        PARAMETER (MYNAME='COLUMN')
        COMPLEX*16 CMD(0:1)
        POINTER (CMD_PTR,CMD)
C-------------------------------------------------------------------------
        CALL FMSPSH (MYNAME)
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('LUPR', LUPR)
        IDO_1 = .TRUE.
        IDO_2 = .TRUE.
        MY_TOTAL = 0
        NUMEQ1 = LUA(8,1)
        NUMEQ2 = LUA(8,2)
C
C       Get temporary storage to hold a column:
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        CALL FMSCMG (CMD, L_A1, NUMEQ1)
        CALL FMSCMG (CMD, L_A2, NUMEQ2)
C
C       Loop over columns:
  100   CONTINUE
C          Get your next column number:
C          Critical Section
C          ================
           IF(OMP_MUTEX) THEN
!$OMP CRITICAL
              MYCOL = INTINC(NXCOL)
!$OMP END CRITICAL
           ELSE
              CALL FMSONE
              MYCOL = INTINC(NXCOL)
              CALL FMSALL
           END IF
C          End of Critical Section
C          =======================
           IF(MYCOL .GT. NUMEQ1) IDO_1 = .FALSE.
           IF(MYCOL .GT. NUMEQ2) IDO_2 = .FALSE.
           IF((.NOT.IDO_1) .AND.
     1        (.NOT.IDO_2) ) THEN
C             This process is done.
              CALL FMSCMR (CMD, L_A1, NUMEQ1)
              CALL FMSCMR (CMD, L_A2, NUMEQ2)
C             Report your total work:
              IF(OMP_MUTEX) THEN
!$OMP CRITICAL
                 WRITE(LUPR,2000) MYNODE, MY_TOTAL
!$OMP END CRITICAL
              ELSE
                 CALL FMSONE
                 WRITE(LUPR,2000) MYNODE, MY_TOTAL
                 CALL FMSALL
              END IF
              CALL FMSPOP (MYNAME)
              RETURN
           ELSE
              MY_TOTAL = MY_TOTAL + 1
           END IF
C
           IF(IDO_1) THEN
              IF(MYCOL .EQ. 1) THEN
                 CMD(L_A1) = DCMPLX(NUMEQ1,NUMEQ1)
                 DO I=2,NUMEQ1
                    CMD(L_A1+I-1) = OFFDIA
                 END DO
```

```
               ELSE
                  CMD(L_A1) = OFFDIA
                  DO I=2,NUMEQ1
                     CMD(L_A1+I-1) = CZERO
                  END DO
                  CMD(L_A1+MYCOL-1) = DIA
               END IF
               CALL FMSCOL (MYCOL, CMD(L_A1), LUA(1,1))
            END IF
C
            IF(IDO_2) THEN
               IF(MYCOL .EQ. 1) THEN
                  CMD(L_A2) = DCMPLX(NUMEQ2,NUMEQ2)
                  DO I=2,NUMEQ2
                     CMD(L_A2+I-1) = OFFDIA
                  END DO
               ELSE
                  CMD(L_A2) = OFFDIA
                  DO I=2,NUMEQ2
                     CMD(L_A2+I-1) = CZERO
                  END DO
                  CMD(L_A2+MYCOL-1) = DIA
               END IF
               CALL FMSCOL (MYCOL, CMD(L_A2), LUA(1,2))
            END IF
C
C           Do the next column:
         GO TO 100
 2000    FORMAT (' Process',I3,' computed',I5,' Columns.')
         END
C=======================================================================
         SUBROUTINE RHS (LUX, NXRHS)
C=======================================================================
C
C        DESCRIPTION:
C           This subroutine computes the RHS vectors.
C           It is designed to be run in parallel.
C
C        FORMAL PARAMETERS:
C           (R ) LUX(25,2) = FMS vector file attributes
C           (RW) NXRHS     = Next RHS to process (shared)
C-----------------------------------------------------------------------
C        Formal Parameters:
C-----------------------------------------------------------------------
         INTEGER    LUX(25,2)
         INTEGER    NXRHS
C-----------------------------------------------------------------------
C        Local Variables
C-----------------------------------------------------------------------
         INTEGER    MYRHS
         LOGICAL    IDO_1, IDO_2
         INTEGER    MY_TOTAL
         INTEGER    MYNODE, LUPR
         CHARACTER*3 MYNAME
         PARAMETER  (MYNAME='RHS')
         INTEGER    NUMEQ1, NUMEQ2
         INTEGER    NUMVEC1, NUMVEC2
         COMMON/MYDATA/OMP_MUTEX
         LOGICAL       OMP_MUTEX
         COMPLEX*16 CZERO, CONE
         DATA CZERO/(0.0D0,0.0D0)/
         DATA CONE /(1.0D0,0.0D0)/
         COMPLEX*16 CMD(0:1)
         POINTER    (CMD_PTR,CMD)
```

```
C
        CALL FMSPSH (MYNAME)
        CALL FMSIGT ('MYNODE', MYNODE)
        CALL FMSIGT ('LUPR', LUPR)
        IDO_1   = .TRUE.
        IDO_2   = .TRUE.
        MY_TOTAL = 0
        NUMEQ1  = LUX( 3,1)
        LENVEC1 = LUX( 4,1)
        NUMVEC1 = LUX( 6,1)
        NUMEQ2  = LUX( 3,2)
        LENVEC2 = LUX( 4,2)
        NUMVEC2 = LUX( 6,2)
C
C       Get temporary storage to hold the longest vector record:
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        IF(LENVEC1 .GT. LENVEC2) THEN
           LENX = LENVEC1/2
        ELSE
           LENX = LENVEC2/2
        END IF
        CALL FMSCMG (CMD, L_X, LENX)
C
C       Populate the vector with test data:
        DO I=2,LENX
           CMD(L_X+I-1) = CZERO
        END DO
        CMD(L_X) = CONE
C
C       Loop over the RHS vectors:
  100   CONTINUE
C          Get your next RHS vector number:
C          Critical Section
C          ================
           IF(OMP_MUTEX) THEN
!$OMP CRITICAL
              MYRHS = INTINC(NXRHS)
!$OMP END CRITICAL
           ELSE
              CALL FMSONE
              MYRHS = INTINC(NXRHS)
              CALL FMSALL
           END IF
C          End of Critical Section
C          =======================
           IF(MYRHS .GT. NUMVEC1) IDO_1 = .FALSE.
           IF(MYRHS .GT. NUMVEC2) IDO_2 = .FALSE.
           IF((.NOT.IDO_1) .AND.
     1        (.NOT.IDO_2) ) THEN
C             This process is done.
              CALL FMSCMR (CMD, L_X, LENX)
C             Report your total work:
              IF(OMP_MUTEX) THEN
!$OMP CRITICAL
                 WRITE(LUPR,2000) MYNODE, MY_TOTAL
!$OMP END CRITICAL
              ELSE
                 CALL FMSONE
                 WRITE(LUPR,2000) MYNODE, MY_TOTAL
                 CALL FMSALL
              END IF
              CALL FMSPOP (MYNAME)
              RETURN
           ELSE
              MY_TOTAL = MY_TOTAL + 1
```

```
            END IF
C
            IF(IDO_1) THEN
               LDISK = 1 + LENVEC1*(MYRHS-1)
               CALL FMSWRT (LUX(1,1), LDISK, CMD(L_X), LENVEC1)
            END IF
C
            IF(IDO_2) THEN
               LDISK = 1 + LENVEC2*(MYRHS-1)
               CALL FMSWRT (LUX(1,2), LDISK, CMD(L_X), LENVEC2)
            END IF
         GO TO 100
 2000    FORMAT (' Process',I3,' computed',I5,' RHS vectors.')
         END
C=======================================================================
         INTEGER FUNCTION INTINC (I)
C-----------------------------------------------------------------------
C        This function increments a volatile shared variable.  It is
C        placed in a subroutine to prevent some compilers from storing
C        the value in a register and not updating it.
         INTEGER I
         I = I + 1
         INTINC = I
         RETURN
         END
C=======================================================================
         LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
         CHARACTER* (*) QUESTION
         CHARACTER*1 IYN
         WRITE(6,2000) QUESTION
         READ (5,1000) IYN
         IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
            ASK = .TRUE.
         ELSE
            ASK = .FALSE.
         END IF
         RETURN
 1000    FORMAT (A)
 2000    FORMAT (1X,A,' (y,n)>')
         END
C=======================================================================
         INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
         CHARACTER* (*) STRING
         WRITE(6,2000) STRING
         READ (5,*) ASK_I
         RETURN
 2000    FORMAT (1X,A,'>')
         END
```

Copyright © Multipath Corporation

FMS EXAMPLE 24. 8-NODE BRICK FINITE ELEMENT EXAMPLE

This example generates a finite element mesh using 8-node bricks. The mesh is NE_THICK by NE_WIDE by NE_LONG. FMSlib is then used to solve the problem for a specified unit force on one end.

```
C       E X A M P L E   2 4
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_24')
C
C       Input data functions:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER   LUA(25)
        INTEGER   LUX(25), LUB(25), LUAX(25)
        INTEGER   LUS(25)
C
```

```
C       FMS memory management requires the following arrays:
        POINTER (IMD_PTR, IMD)
        POINTER (RMD_PTR, RMD)
        POINTER (CMD_PTR, CMD)
        INTEGER    IMD(0:1)
        REAL*8     RMD(0:1)
        COMPLEX*16 CMD(0:1)
C
C       Allocated array starting values:
        INTEGER NEWNUM
        INTEGER LB
        INTEGER LX
        INTEGER IREC
        INTEGER LS
        INTEGER LOWEQ
        INTEGER LAX
        INTEGER LIFIX
C
C       FMS Parameter values;
        INTEGER MPOSDF
        DATA    MPOSDF/1/
        INTEGER IPRI
        DATA    IPRI/1/
        INTEGER IPRF
        DATA    IPRF/1026/
        INTEGER MDATAU
        DATA    MDATAU/2/
        INTEGER LOGTIM
        DATA    LOGTIM/3/
        INTEGER INSIDE
        DATA    INSIDE/-1/
        INTEGER NEQBLK
        DATA    NEQBLK/96/
        INTEGER INCORE
        DATA    INCORE/2/
C
C       Big diagonal value to impose constraints:
        REAL*8     BIGNUM
        DATA BIGNUM/1.0D30/
```

```
C
C       Read in problem data:
        INTEGER   NETHICK
        INTEGER   NEWIDE
        INTEGER   NELONG
        INTEGER   NDOF
C
C       Local variables:
        INTEGER   MAXEQ, NELDOF, NUMEQ
        INTEGER   LENR, LENI, LENV
        INTEGER   INEW, I
        INTEGER   LENX, LDISK, NV
        REAL*8    EI, ERROR
C
C       Dummy arguments:
        INTEGER LUDUMMY(25)
        DATA LUDUMMY(1)/0/
        REAL*8  ALPHA(1)
        DATA ALPHA(1)/0.0D0/
C
C       Common block to communicate with user supplied subroutine.
        COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
        INTEGER   NPTHICK, NPWIDE, NPLONG
C
C (1)   Initialize FMS:
        CALL FMSINI
        CALL FMSPSH(MYNAME)
C
C       Require matrix to be positive definite:
        CALL FMSIST ('MPOSDF',MPOSDF)
C
C       Call RSUBLK during factoring:
        CALL FMSIST ('MDATAU',MDATAU)
C
C       Get the memory pointers:
C
C       Turn on timings:
        CALL FMSIGT ('LOGTIM', LOGTIM)
```

```
      IF(LOGTIM .LT. 3) CALL FMSIST ('LOGTIM',    3)
C
C     Set matrix initialization print level:
      CALL FMSIST ('IPRI'  , IPRI)
C
C     Select outside blocking of matrix profile:
      CALL FMSIST ('INSIDE'  , INSIDE)
C
C     Select block size:
      CALL FMSIST ('NEQBLK'  , NEQBLK)
C
C     Set matrix factoring print level:
      CALL FMSIGT ('IPRF', IPRF)
      IF(IAND(IPRF,   2) .EQ. 0) IPRF = IPRF + 2
      IF(IAND(IPRF,1024) .EQ. 0) IPRF = IPRF + 1024
      CALL FMSIST ('IPRF'  , IPRF)
C
C     Store data in memory if it will fit:
      CALL FMSIST ('INCORE'  , INCORE)
C
C     Read in problem data:
    1 CONTINUE
      NETHICK = ASK_I('Enter the number of elements thick')
      NEWIDE  = ASK_I('Enter the number of elements wide')
      NELONG  = ASK_I('Enter the number of elements long')
      NUMEL   = NETHICK * NEWIDE * NELONG
      IF(NUMEL .LE. 0) GO TO 1
    2 CONTINUE
      NDOF    = ASK_I('Enter the number of degrees of freedom per node')
      IF(NDOF  .LE. 0) GO TO 2
      WRITE(6,*) 'You may now alter any FMS parameter.'
      WRITE(6,*) 'When you are finished, type the letters RETURN'
      CALL FMSSET
      CALL FMSIGT ('MEMPTR',IMD_PTR)
      CALL FMSIGT ('MEMPTR',RMD_PTR)
      CALL FMSIGT ('MEMPTR',CMD_PTR)
      NPTHICK = NETHICK + 1
      NPWIDE  = NEWIDE  + 1
```

```
      NPLONG  = NELONG  + 1
      MAXEQ   = NPTHICK * NPWIDE * NPLONG * NDOF
      NELDOF  = 8 * NDOF
C
C (2)   Open FMS files:
C
C       Submatrix File:
      LENR    = (NELDOF * (NELDOF+1))/2
      LENI    = 2 + NELDOF
      LENV    = NELDOF
      CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C
C       Allocate storage for element integer record:
      LENI = LUS(4)
      CALL FMSIMG (IMD, IREC, LENI)
C
C       Allocate storage for element real record:
      LENR = LUS(3)
      CALL FMSRMG (RMD, LS, LENR)
C
C       Generate element data:
      CALL ELGEN (NETHICK, NEWIDE, NELONG, NUMEL,
     1   LUS, IMD(IREC), RMD(LS), NELDOF)
      WRITE(6,*) 'NUMBER OF ELEMENTS          = ', NUMEL
      WRITE(6,*) 'INITIAL NUMBER OF EQUATIONS = ', MAXEQ
C
C       Free storage for element real record:
      CALL FMSRMR (RMD, LS, LENR)
C
C       Free storage for element integer record:
      CALL FMSIMR (IMD, IREC, LENI)
C
C       Allocate storage for NEWNUM vector:
      CALL FMSIMG (IMD, NEWNUM, MAXEQ)
C
C       Allocate storage for solution vector {X}:
      CALL FMSRMG (RMD, LX, MAXEQ)
C
C       Impose displacement boundary conditions:
```

```
         CALL XBC (RMD(LX), IMD(NEWNUM), MAXEQ)
C
C        Allocate storage for profile vector:
         CALL FMSIMG (IMD, LOWEQ, MAXEQ)
C
C        Compute profile vector; eliminate zero constraints:
         CALL FMSPF (LUS, NUMSF, NEWNUM, MAXEQ, LOWEQ, NUMEQ)
         WRITE(6,*) 'ACTUAL  NUMBER OF EQUATIONS = ', NUMEQ
C
C        Open the global matrix file:
         CALL RSDI   (LOWEQ, NUMEQ, 'LUA', LUA)
C
C        Free storage for profile vector:
         CALL FMSIMR (IMD, LOWEQ, MAXEQ)
C
C (3)    Write data to FMS files:
C
C        Allocate storage for IFIX vector to modify diagonals for
C        nonzero constrained displacements:
         CALL FMSIMG (IMD, LIFIX, NUMEQ)
C
C        Flag diagonals with nonzero specified displacements:
         DO I = 1,MAXEQ
            INEW = IMD(NEWNUM+I-1)
            IF(INEW .LT. 0) THEN
C              Value constrained to a nonzero value:
               INEW   = -INEW
               IMD(LIFIX+INEW-1) = 1
            ELSE
               IMD(LIFIX+INEW-1) = 0
            END IF
         END DO
C
C        Allocate storage for RHS vector {B}:
         CALL FMSRMG (RMD, LB, MAXEQ)
C
C        Open an incore file for {X}:
         CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  RMD(LX), NUMEQ, LUX)
C
```

```
C       Open an incore file for {B}:
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  RMD(LB), NUMEQ, LUB)
C
C (4)   Perform matrix algebra:
C       Build and factor the matrix:
        CALL RSDAF (LUDUMMY, ALPHA, 0, LUS, NUMSF, LUDUMMY,
     1   LUA, LUB, LUX, 0)
C
C       Build {B} and solve for {X}:
        CALL BBC(RMD(LB), IMD(NEWNUM), MAXEQ)
C
C       Solve for {X}:
        CALL RSDS (LUA, LUB, LUX, NUMRHS, 0)
C
C (5)   Read data from FMS files:
C       Not required.
C
C       Compute [A]{X} from SUM([Si]{X}) to check error:
C
C       Allocate storage for [A]{X} vector:
        CALL FMSRMG (RMD, LAX, MAXEQ)
C
C       Open an incore file for [A]{X} vector:
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, RMD(LAX), NUMEQ, LUAX)
C
C       Compute {AX} = [Si]{X}:
        CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
C
C       Compute the error {AX} - {B}:
        ERROR = 0.0D0
C       Expand solution vector to include zero constrained values:
        DO 50 I = MAXEQ,1,-1
           INEW   = IMD(NEWNUM+I-1)
           IF(INEW .EQ. 0) THEN
C             Constrained zero value:
              RMD(LB+I-1) = 0.0D0
           ELSE IF(INEW .LT. 0) THEN
C             Constrained nonzero value:
              INEW = -INEW
```

```
               RMD(LB+I-1) = RMD(LB+INEW-1)
            ELSE
C              Value which was determined:
               RMD(LB+1-I) = RMD(LB+INEW-1)
               EI = ABS(RMD(LAX+INEW-1)-RMD(LB+INEW-1))
               IF(EI .GT. ERROR) ERROR = EI
            END IF
   50    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =',ERROR
C
C (6)    Close FMS files:
         CALL FMSCM  (LUA)
         CALL FMSCV  (LUAX)
         CALL FMSRMR (RMD, LAX, MAXEQ)
         CALL FMSCV  (LUX)
         CALL FMSRMR (RMD, LX, MAXEQ)
         CALL FMSCV  (LUB)
         CALL FMSRMR (RMD, LB, MAXEQ)
         CALL FMSIMR (IMD, LIFIX, NUMEQ)
         CALL FMSCS  (LUS)
         CALL FMSIMR (IMD, NEWNUM, MAXEQ)
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP(MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER LOWEQ(*), LOCEQ(*)
         REAL*8  A(0:*), D(*)
         POINTER (IMD_PTR, IMD)
         INTEGER  IMD(0:1)
         COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
         INTEGER LIFIX
         REAL*8  BIGNUM
C
```

```
        CALL FMSIGT ('MEMPTR', IMD_PTR)
C       Modify diagonal for nonzero constraint equations:
        IF(IROW2 .EQ. JCOL2) THEN
C          This is a diagonal block:
           DO I = IROW1,IROW2
              IF(IMD(LIFIX-1+I) .NE. 0) D(I) = BIGNUM
           END DO
        END IF
        RETURN
        END
C=======================================================================
        SUBROUTINE ELGEN (
     1   NETHICK, NEWIDE, NELONG, NUMEL,
     2   LUS, IREC, S, NELDOF)
C=======================================================================
        INTEGER NETHICK, NEWIDE, NELONG, NUMEL
        INTEGER LUS(25)
        INTEGER IREC(*)
        INTEGER NELDOF
        REAL*8  S(*)
        COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
        INTEGER NPTHICK, NPWIDE, NPLONG, NDOF
        INTEGER LENI, LENR, LENV
        REAL*8  DIAG
        CHARACTER*5 MYNAME
        PARAMETER (MYNAME='ELGEN')
C
        CALL FMSPSH(MYNAME)
        LENR   = LUS(3)
        LENI   = LUS(4)
        LENV   = LUS(10)
        DIAG   = DFLOAT(NELDOF-1)
D       WRITE(LUDBUG,2000),
D    1  NETHICK, NEWIDE, NELONG, NUMEL,
D    2  LOC(LUS), LOC(IREC), LOC(S), NELDOF
C
C       Generate an 8-node finite element mesh that is NETHICK elements
C       thick, NEWIDE elements wide by NELONG elements long.
```

```
C       Number the nodes first in the thick direction, then in the
C       wide direction and then in the long direction.
C       Use the same matrix S(NELDOF,NELDOF) for all elements.
C
C       ========================
C       Element integer records:
C       ========================
C       Write the element integer file.
C       Each record contains the following:
C       IREC(1) = Number of element degrees of freedom.
C       IREC(2) = Element format type (1:5)
C       IREC(3:LENI) = Element global equation numbers
        CALL FMSEEK (LUS(2), 0)
        IREC(1) = NELDOF
        IREC(2) = 4
C       DOF increments:
        INCTHICK = NDOF
        INCWIDE  = INCTHICK * NPTHICK
        INCLONG  = INCWIDE  * NPWIDE
D       WRITE(LUDBUG,2005) INCTHICK, INCWIDE, INCLONG
C       Element numbering:
        NP1 = 2
        NP2 = NP1 + NDOF
        NP3 = NP2 + NDOF
        NP4 = NP3 + NDOF
        NP5 = NP4 + NDOF
        NP6 = NP5 + NDOF
        NP7 = NP6 + NDOF
        NP8 = NP7 + NDOF
D       WRITE(LUDBUG,2001) NP1, NP2, NP3, NP4, NP5, NP6, NP7, NP8
C       Global numbering:
        NP1G = 0
        NP2G = NP1G + NDOF
        NP3G = NP1G + INCWIDE
        NP4G = NP3G + NDOF
        NP5G = NP1G + INCLONG
        NP6G = NP5G + NDOF
        NP7G = NP5G + INCWIDE
```

```
         NP8G = NP7G + NDOF
D        WRITE(LUDBUG,2002) NP1G, NP2G, NP3G, NP4G, NP5G, NP6G, NP7G,
D     1  NP8G
         NDOFLONG1 = 0
         NE        = 0
D        WRITE(LUDBUG,2003)
         DO I = 1,NELONG
            NDOFWIDE1 = NDOFLONG1
            DO J = 1,NEWIDE
               NDOFTHICK1 = NDOFWIDE1
               DO K = 1,NETHICK
                  NDOF1 = NDOFTHICK1
                  NE   = NE + 1
                  DO IDOF = 1,NDOF
                     NDOF1 = NDOF1 + 1
                     IREC(NP1+IDOF) = NP1G + NDOF1
                     IREC(NP2+IDOF) = NP2G + NDOF1
                     IREC(NP3+IDOF) = NP3G + NDOF1
                     IREC(NP4+IDOF) = NP4G + NDOF1
                     IREC(NP5+IDOF) = NP5G + NDOF1
                     IREC(NP6+IDOF) = NP6G + NDOF1
                     IREC(NP7+IDOF) = NP7G + NDOF1
                     IREC(NP8+IDOF) = NP8G + NDOF1
                  END DO
D                 WRITE(LUDBUG,2004) NE, (IREC(II),II=1,LENI)
                  CALL FMSSWR (LUS(2), IREC, LENI)
                  NDOFTHICK1 = NDOFTHICK1 + INCTHICK
               END DO
               NDOFWIDE1 = NDOFWIDE1 + INCWIDE
            END DO
            NDOFLONG1 = NDOFLONG1 + INCLONG
         END DO
C        ====================
C        Element real records:
C        ====================
C        Use lower triangle stored by row format:
         L = 0
         DO I = 1,NELDOF
            DO J = 1,(I-1)
```

```
               L = L + 1
               S(L) = -1.0D0
            END DO
            L = L + 1
            S(L) = DIAG
         END DO
C
C        Write element real records:
         CALL FMSEEK (LUS(1), 0)
         DO NE = 1,NUMEL
            CALL FMSSWR (LUS(1), S, LENR)
         END DO

         CALL FMSPOP(MYNAME)
         RETURN
D2000    FORMAT(
D     1  'SUBROUTINE ELGEN('/
D     2  ' NETHICK=',I5,', NEWIDE=',I5,', NELONG=',I5,', NUMEL=',I5/
D     5  ' &LUS   =',I15,', &IREC=',I15,', &S=',I15,' NELDOF=',I5,'0'/)
D2001    FORMAT(' NP1 -NP8 :',I4,7I5)
D2002    FORMAT(' NP1G-NP8G:',I4,7I5)
D2003    FORMAT(/
D     1  '   NE SIZE TYPE  EQ1'/
D     2  '===== ==== ==== ====')
D2004    FORMAT(15I5/(15X,12I5))
D2005    FORMAT(' INCTHICK, INCWIDE, INCLONG:  ',3I5)
         END
C=======================================================================
         SUBROUTINE XBC (X, NEWNUM, MAXEQ)
C=======================================================================
C        Impose displacement boundary conditions:
         REAL*8  X(MAXEQ)
         INTEGER NEWNUM(MAXEQ)
         CHARACTER*3 MYNAME
         COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
         INTEGER NPTHICK, NPWIDE, NPLONG
         PARAMETER (MYNAME='XBC')
         CALL FMSPSH(MYNAME)
```

```
C
C       Initialize all values to free:
        DO I = 1,MAXEQ
           NEWNUM(I) = 1
           X(I) = 0.0D0
        END DO
C
C       Constrain all nodes at Z=0 to z=0.
        IDOF = 0
        DO I=1,NPTHICK
           DO J=1,NPWIDE
              IDOF = IDOF + NDOF
              NEWNUM(IDOF) = 0
           END DO
        END DO
C
C       Constrain node 1 to (0,0,0):
        DO I=1,NDOF
           NEWNUM(I) = 0
        END DO
C
C       Constrain node at (1,NEWIDE,1) to x=0:
        IDOF = NDOF*(NPTHICK * (NPWIDE-1)) + 1
        NEWNUM(IDOF) = 0
D       WRITE(6,2000)
D       DO I=1,MAXEQ
D          WRITE(6,2001) I, NEWNUM(I), X(I)
D       END DO
        CALL FMSPOP(MYNAME)
        RETURN
D2000   FORMAT(/
D    1  '    I   NEWNUM        X'/
D    2  '=====   ======   ======')
D2001   FORMAT(I5, 2I9)
        END
C=======================================================================
        SUBROUTINE BBC (B, NEWNUM, MAXEQ)
C=======================================================================
C       Impose force boundary conditions:
```

```
      REAL*8  B(MAXEQ)
      INTEGER NEWNUM(MAXEQ)
      COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
      INTEGER NPTHICK, NPWIDE, NPLONG
      CHARACTER*3 MYNAME
      PARAMETER (MYNAME='BBC')
      CALL FMSPSH(MYNAME)
C
C     Apply a force of 1 on all Z nodes at Z=NPLONG:
      NFDOF = NDOF * NPTHICK * NPWIDE
      DO I = (MAXEQ-NFDOF+NDOF),MAXEQ,NDOF
         IF(NEWNUM(I) .LE. 0) THEN
C           DOF already has a specified displacement:
            PRINT *,'BBC: DOF',I, 'has conflicted boundary conditions'
            STOP 'FATAL ERROR IN SUBROUTINE BBC'
         END IF
         B(I) = 1.0D0
      END DO
C
C     Reorder {B} for new numbering system:
      DO I = 1,MAXEQ
         INEW = NEWNUM(I)
         IF(INEW .GT. 0) THEN
C           Value to be determined:
            B(INEW) = B(I)
         ELSE IF(INEQ .EQ. 0) THEN
C           Value constrained to 0:
            B(INEW) = 0.0D0
         ELSE
C           Value constrained to a nonzero value:
            INEW = -INEW
            B(INEW) = BIGNUM*B(I)
         END IF
      END DO
      CALL FMSPOP(MYNAME)
      RETURN
      END
C=======================================================================
```

```
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

```
C       E X A M P L E   2 4
C
C       Program name:
        CHARACTER*10 MYNAME
        PARAMETER (MYNAME='EXAMPLE_24')
C
C       Input data functions:
        LOGICAL ASK
        INTEGER ASK_I
C
C       Data type = real:
        PARAMETER (IDTYPE = 1)
C
C       Number of RHS vectors:
        PARAMETER (NUMRHS = 1)
C
C       Number of element files:
        PARAMETER (NUMSF  = 1)
C
C       Number of element vectors:
        PARAMETER (NUMELV = 0)
C
C       Skip forward reduction during call to RSDS:
        PARAMETER (ISKIP  = 0)
C
C       FMS matrix and vector file attributes:
        INTEGER   LUA(25)
        INTEGER   LUX(25), LUB(25), LUAX(25)
        INTEGER   LUS(25)
C
C       FMS memory management requires the following arrays:
        POINTER (IMD_PTR, IMD)
        POINTER (RMD_PTR, RMD)
        POINTER (CMD_PTR, CMD)
        INTEGER    IMD(0:1)
        REAL*8     RMD(0:1)
        COMPLEX*16 CMD(0:1)
C
C       Allocated array starting values:
        INTEGER NEWNUM
        INTEGER LB
        INTEGER LX
        INTEGER IREC
        INTEGER LS
        INTEGER LOWEQ
        INTEGER LAX
        INTEGER LIFIX
C
C       FMS Parameter values;
        INTEGER MPOSDF
        DATA    MPOSDF/1/
        INTEGER IPRI
        DATA    IPRI/1/
        INTEGER IPRF
        DATA    IPRF/1026/
        INTEGER MDATAU
        DATA    MDATAU/2/
        INTEGER LOGTIM
        DATA    LOGTIM/3/
        INTEGER INSIDE
        DATA    INSIDE/-1/
        INTEGER NEQBLK
        DATA    NEQBLK/96/
```

```
      INTEGER INCORE
      DATA    INCORE/2/
C
C     Big diagonal value to impose constraints:
      REAL*8     BIGNUM
      DATA BIGNUM/1.0D30/
C
C     Read in problem data:
      INTEGER   NETHICK
      INTEGER   NEWIDE
      INTEGER   NELONG
      INTEGER   NDOF
C
C     Local variables:
      INTEGER   MAXEQ, NELDOF, NUMEQ
      INTEGER   LENR, LENI, LENV
      INTEGER   INEW, I
      INTEGER   LENX, LDISK, NV
      REAL*8    EI, ERROR
C
C     Dummy arguments:
      INTEGER LUDUMMY(25)
      DATA LUDUMMY(1)/0/
      REAL*8  ALPHA(1)
      DATA ALPHA(1)/0.0D0/
C
C     Common block to communicate with user supplied subroutine.
      COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
      INTEGER   NPTHICK, NPWIDE, NPLONG
C
C (1)   Initialize FMS:
      CALL FMSINI
      CALL FMSPSH (MYNAME)
C
C     Require matrix to be positive definite:
      CALL FMSIST ('MPOSDF',MPOSDF)
C
C     Call RSUBLK during factoring:
      CALL FMSIST ('MDATAU',MDATAU)
C
C     Get the memory pointers:
C
C     Turn on timings:
      CALL FMSIGT ('LOGTIM', LOGTIM)
      IF(LOGTIM .LT. 3) CALL FMSIST ('LOGTIM',    3)
C
C     Set matrix initialization print level:
      CALL FMSIST ('IPRI'  , IPRI)
C
C     Select outside blocking of matrix profile:
      CALL FMSIST ('INSIDE'  , INSIDE)
C
C     Select block size:
      CALL FMSIST ('NEQBLK'  , NEQBLK)
C
C     Set matrix factoring print level:
      CALL FMSIGT ('IPRF', IPRF)
      IF(IAND(IPRF,   2) .EQ. 0) IPRF = IPRF + 2
      IF(IAND(IPRF,1024) .EQ. 0) IPRF = IPRF + 1024
      CALL FMSIST ('IPRF'  , IPRF)
C
C     Store data in memory if it will fit:
      CALL FMSIST ('INCORE'  , INCORE)
C
```

```
C       Read in problem data:
    1   CONTINUE
        NETHICK = ASK_I('Enter the number of elements thick')
        NEWIDE  = ASK_I('Enter the number of elements wide')
        NELONG  = ASK_I('Enter the number of elements long')
        NUMEL   = NETHICK * NEWIDE * NELONG
        IF(NUMEL .LE. 0) GO TO 1
    2   CONTINUE
        NDOF    = ASK_I('Enter the number of degrees of freedom per node')
        IF(NDOF  .LE. 0) GO TO 2
        WRITE(6,*) 'You may now alter any FMS parameter.'
        WRITE(6,*) 'When you are finished, type the letters RETURN'
        CALL FMSSET()
        CALL FMSIGT ('MEMPTR', IMD_PTR)
        CALL FMSIGT ('MEMPTR', RMD_PTR)
        CALL FMSIGT ('MEMPTR', CMD_PTR)
        NPTHICK = NETHICK + 1
        NPWIDE  = NEWIDE  + 1
        NPLONG  = NELONG  + 1
        MAXEQ   = NPTHICK * NPWIDE * NPLONG * NDOF
        NELDOF  = 8 * NDOF
C
C (2)   Open FMS files:
C
C       Submatrix File:
        LENR    = (NELDOF * (NELDOF+1))/2
        LENI    = 2 + NELDOF
        LENV    = NELDOF
        CALL FMSOS  (LENR, LENI, LENV, NUMELV, NUMEL, 'LUS', LUS)
C
C       Allocate storage for element integer record:
        LENI = LUS(4)
        CALL FMSIMG (IMD, IREC, LENI)
C
C       Allocate storage for element real record:
        LENR = LUS(3)
        CALL FMSRMG (RMD, LS, LENR)
C
C       Generate element data:
        CALL ELGEN (NETHICK, NEWIDE, NELONG, NUMEL,
     1   LUS, IMD(IREC), RMD(LS), NELDOF)
        WRITE(6,*) 'NUMBER OF ELEMENTS          = ', NUMEL
        WRITE(6,*) 'INITIAL NUMBER OF EQUATIONS = ', MAXEQ
C
C       Free storage for element real record:
        CALL FMSRMR (RMD, LS, LENR)
C
C       Free storage for element integer record:
        CALL FMSIMR (IMD, IREC, LENI)
C
C       Allocate storage for NEWNUM vector:
        CALL FMSIMG (IMD, NEWNUM, MAXEQ)
C
C       Allocate storage for solution vector {X}:
        CALL FMSRMG (RMD, LX, MAXEQ)
C
C       Impose displacement boundary conditions:
        CALL XBC (RMD(LX), IMD(NEWNUM), MAXEQ)
C
C       Allocate storage for profile vector:
        CALL FMSIMG (IMD, LOWEQ, MAXEQ)
C
C       Compute profile vector; eliminate zero constraints:
        CALL FMSPF (LUS, NUMSF, IMD(NEWNUM), MAXEQ, IMD(LOWEQ), NUMEQ)
        WRITE(6,*) 'ACTUAL  NUMBER OF EQUATIONS = ', NUMEQ
```

```
C
C       Open the global matrix file:
        CALL RSDI   (IMD(LOWEQ), NUMEQ, 'LUA', LUA)
C
C       Free storage for profile vector:
        CALL FMSIMR (IMD, LOWEQ, MAXEQ)
C
C (3)   Write data to FMS files:
C
C       Allocate storage for IFIX vector to modify diagonals for
C       nonzero constrained displacements:
        CALL FMSIMG (IMD, LIFIX, NUMEQ)
C
C       Flag diagonals with nonzero specified displacements:
        DO I = 1,MAXEQ
           INEW = IMD(NEWNUM+I-1)
           IF(INEW .LT. 0) THEN
C             Value constrained to a nonzero value:
              INEW   = -INEW
              IMD(LIFIX+INEW-1) = 1
           ELSE
              IMD(LIFIX+INEW-1) = 0
           END IF
        END DO
C
C       Allocate storage for RHS vector {B}:
        CALL FMSRMG (RMD, LB, MAXEQ)
C
C       Open an incore file for {X}:
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  RMD(LX), MAXEQ, LUX)
C
C       Open an incore file for {B}:
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS,  RMD(LB), MAXEQ, LUB)
C
C (4)   Perform matrix algebra:
C       Build and factor the matrix:
        CALL RSDAF (LUDUMMY, ALPHA, 0, LUS, NUMSF, LUDUMMY,
     1   LUA, LUB, LUX, 0)
C
C       Build {B} and solve for {X}:
        CALL BBC(RMD(LB), IMD(NEWNUM), MAXEQ)
C
C       Solve for {X}:
        CALL RSDS (LUA, LUB, LUX, NUMRHS, 0)
C
C (5)   Read data from FMS files:
C       Not required.
C
C       Compute [A]{X} from SUM([Si]{X}) to check error:
C
C       Allocate storage for [A]{X} vector:
        CALL FMSRMG (RMD, LAX, MAXEQ)
C
C       Open an incore file for [A]{X} vector:
        CALL FMSOV2 (NUMEQ, IDTYPE, NUMRHS, RMD(LAX), NUMEQ, LUAX)
C
C       Compute {AX} = [Si]{X}:
        CALL RSDSVM (LUS, NUMSF, LUX, LUAX, NUMRHS)
C
C       Compute the error {AX} - {B}:
        ERROR = 0.0D0
C       Expand solution vector to include zero constrained values:
        DO 50 I = MAXEQ,1,-1
           INEW   = IMD(NEWNUM+I-1)
```

```
            IF(INEW .EQ. 0) THEN
C              Constrained zero value:
               RMD(LB+I-1) = 0.0D0
            ELSE IF(INEW .LT. 0) THEN
C              Constrained nonzero value:
               INEW = -INEW
               RMD(LB+I-1) = RMD(LB+INEW-1)
            ELSE
C              Value which was determined:
               RMD(LB+1-I) = RMD(LB+INEW-1)
               EI = ABS(RMD(LAX+INEW-1)-RMD(LB+INEW-1))
               IF(EI .GT. ERROR) ERROR = EI
            END IF
   50    CONTINUE
         WRITE(6,*) 'MAXIMUM ERROR =',ERROR
C
C (6)    Close FMS files:
         CALL FMSCM  (LUA)
         CALL FMSCV  (LUAX)
         CALL FMSRMR (RMD, LAX, MAXEQ)
         CALL FMSCV  (LUX)
         CALL FMSRMR (RMD, LX, MAXEQ)
         CALL FMSCV  (LUB)
         CALL FMSRMR (RMD, LB, MAXEQ)
         CALL FMSIMR (IMD, LIFIX, NUMEQ)
         CALL FMSCS  (LUS)
         CALL FMSIMR (IMD, NEWNUM, MAXEQ)
         IF(ASK('Do you want another solution?')) GO TO 1
         CALL FMSPOP (MYNAME)
         CALL FMSEND
         END
C=======================================================================
         SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2,
     1                      JCOL1, JCOL2, IJSTEP)
C=======================================================================
         INTEGER IROW1, IROW2, JCOL1, JCOL2, IJSTEP
         INTEGER LOWEQ(*), LOCEQ(*)
         REAL*8  A(0:*), D(*)
         POINTER (IMD_PTR, IMD)
         INTEGER  IMD(0:1)
         COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
         INTEGER LIFIX
         REAL*8  BIGNUM
C
         CALL FMSIGT ('MEMPTR', IMD_PTR)
C        Modify diagonal for nonzero constraint equations:
         IF(IROW2 .EQ. JCOL2) THEN
C           This is a diagonal block:
            DO I = IROW1,IROW2
               IF(IMD(LIFIX-1+I) .NE. 0) D(I) = BIGNUM
            END DO
         END IF
         RETURN
         END
C=======================================================================
         SUBROUTINE ELGEN (
     1   NETHICK, NEWIDE, NELONG, NUMEL,
     2   LUS, IREC, S, NELDOF)
C=======================================================================
         INTEGER NETHICK, NEWIDE, NELONG, NUMEL
         INTEGER LUS(25)
         INTEGER IREC(*)
         INTEGER NELDOF
         REAL*8  S(*)
```

```
      COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1   BIGNUM, LIFIX
      INTEGER NPTHICK, NPWIDE, NPLONG, NDOF
      INTEGER LENI, LENR, LENV
      REAL*8  DIAG
      CHARACTER*5 MYNAME
      PARAMETER (MYNAME='ELGEN')
C
      CALL FMSPSH (MYNAME)
      LENR   = LUS(3)
      LENI   = LUS(4)
      LENV   = LUS(10)
      DIAG   = DFLOAT(NELDOF-1)
D     WRITE(LUDBUG,2000),
D    1  NETHICK, NEWIDE, NELONG, NUMEL,
D    2  LOC(LUS), LOC(IREC), LOC(S), NELDOF
C
C     Generate an 8-node finite element mesh that is NETHICK elements
C     thick, NEWIDE elements wide by NELONG elements long.
C     Number the nodes first in the thick direction, then in the
C     wide direction and then in the long direction.
C     Use the same matrix S(NELDOF,NELDOF) for all elements.
C
C     ========================
C     Element integer records:
C     ========================
C     Write the element integer file.
C     Each record contains the following:
C     IREC(1) = Number of element degrees of freedom.
C     IREC(2) = Element format type (1:5)
C     IREC(3:LENI) = Element global equation numbers
      CALL FMSEEK (LUS(2), 0)
      IREC(1) = NELDOF
      IREC(2) = 4
C     DOF increments:
      INCTHICK = NDOF
      INCWIDE  = INCTHICK * NPTHICK
      INCLONG  = INCWIDE  * NPWIDE
D     WRITE(LUDBUG,2005) INCTHICK, INCWIDE, INCLONG
C     Element numbering:
      NP1 = 2
      NP2 = NP1 + NDOF
      NP3 = NP2 + NDOF
      NP4 = NP3 + NDOF
      NP5 = NP4 + NDOF
      NP6 = NP5 + NDOF
      NP7 = NP6 + NDOF
      NP8 = NP7 + NDOF
D     WRITE(LUDBUG,2001) NP1, NP2, NP3, NP4, NP5, NP6, NP7, NP8
C     Global numbering:
      NP1G = 0
      NP2G = NP1G + NDOF
      NP3G = NP1G + INCWIDE
      NP4G = NP3G + NDOF
      NP5G = NP1G + INCLONG
      NP6G = NP5G + NDOF
      NP7G = NP5G + INCWIDE
      NP8G = NP7G + NDOF
D     WRITE(LUDBUG,2002) NP1G, NP2G, NP3G, NP4G, NP5G, NP6G, NP7G,
D    1  NP8G
      NDOFLONG1 = 0
      NE        = 0
D     WRITE(LUDBUG,2003)
      DO I = 1,NELONG
         NDOFWIDE1 = NDOFLONG1
```

```
            DO J = 1,NEWIDE
               NDOFTHICK1 = NDOFWIDE1
               DO K = 1,NETHICK
                  NDOF1 = NDOFTHICK1
                  NE   = NE + 1
                  DO IDOF = 1,NDOF
                     NDOF1 = NDOF1 + 1
                     IREC(NP1+IDOF) = NP1G + NDOF1
                     IREC(NP2+IDOF) = NP2G + NDOF1
                     IREC(NP3+IDOF) = NP3G + NDOF1
                     IREC(NP4+IDOF) = NP4G + NDOF1
                     IREC(NP5+IDOF) = NP5G + NDOF1
                     IREC(NP6+IDOF) = NP6G + NDOF1
                     IREC(NP7+IDOF) = NP7G + NDOF1
                     IREC(NP8+IDOF) = NP8G + NDOF1
                  END DO
D                 WRITE(LUDBUG,2004) NE, (IREC(II),II=1,LENI)
                  CALL FMSSWR (LUS(2), IREC, LENI)
                  NDOFTHICK1 = NDOFTHICK1 + INCTHICK
               END DO
               NDOFWIDE1 = NDOFWIDE1 + INCWIDE
            END DO
            NDOFLONG1 = NDOFLONG1 + INCLONG
         END DO
C        =====================
C        Element real records:
C        =====================
C        Use lower triangle stored by row format:
         L = 0
         DO I = 1,NELDOF
            DO J = 1,(I-1)
               L = L + 1
               S(L) = -1.0D0
            END DO
            L = L + 1
            S(L) = DIAG
         END DO
C
C        Write element real records:
         CALL FMSEEK (LUS(1), 0)
         DO NE = 1,NUMEL
            CALL FMSSWR (LUS(1), S, LENR)
         END DO

         CALL FMSPOP (MYNAME)
         RETURN
D2000    FORMAT(
D    1   'SUBROUTINE ELGEN('/
D    2   ' NETHICK=',I5,', NEWIDE=',I5,', NELONG=',I5,', NUMEL=',I5/
D    5   ' &LUS   =',I15,', &IREC=',I15,', &S=',I15,' NELDOF=',I5,'0'/)
D2001    FORMAT(' NP1 -NP8 :',I4,7I5)
D2002    FORMAT(' NP1G-NP8G:',I4,7I5)
D2003    FORMAT(/
D    1   '   NE SIZE TYPE  EQ1'/
D    2   '===== ==== ==== ====')
D2004    FORMAT(15I5/(15X,12I5))
D2005    FORMAT(' INCTHICK, INCWIDE, INCLONG:  ',3I5)
         END
C=======================================================================
         SUBROUTINE XBC (X, NEWNUM, MAXEQ)
C=======================================================================
C        Impose displacement boundary conditions:
         REAL*8  X(MAXEQ)
         INTEGER NEWNUM(MAXEQ)
```

```
         CHARACTER*3 MYNAME
         COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1    BIGNUM, LIFIX
         INTEGER NPTHICK, NPWIDE, NPLONG
         PARAMETER (MYNAME='XBC')
         CALL FMSPSH (MYNAME)
C
C        Initialize all values to free:
         DO I = 1,MAXEQ
            NEWNUM(I) = 1
            X(I) = 0.0D0
         END DO
C
C        Constrain all nodes at Z=0 to z=0.
         IDOF = 0
         DO I=1,NPTHICK
            DO J=1,NPWIDE
               IDOF = IDOF + NDOF
               NEWNUM(IDOF) = 0
            END DO
         END DO
C
C        Constrain node 1 to (0,0,0):
         DO I=1,NDOF
            NEWNUM(I) = 0
         END DO
C
C        Constrain node at (1,NEWIDE,1) to x=0:
         IDOF = NDOF*(NPTHICK * (NPWIDE-1)) + 1
         NEWNUM(IDOF) = 0
D        WRITE(6,2000)
D        DO I=1,MAXEQ
D           WRITE(6,2001) I, NEWNUM(I), X(I)
D        END DO
         CALL FMSPOP (MYNAME)
         RETURN
D2000    FORMAT(/
D    1   '    I   NEWNUM        X'/
D    2   '=====   ======   ======')
D2001    FORMAT(I5, 2I9)
         END
C=======================================================================
         SUBROUTINE BBC (B, NEWNUM, MAXEQ)
C=======================================================================
C        Impose force boundary conditions:
         REAL*8  B(MAXEQ)
         INTEGER NEWNUM(MAXEQ)
         COMMON/MYDATA/NPTHICK, NPWIDE, NPLONG, NDOF,
     1    BIGNUM, LIFIX
         INTEGER NPTHICK, NPWIDE, NPLONG
         CHARACTER*3 MYNAME
         PARAMETER (MYNAME='BBC')
         CALL FMSPSH (MYNAME)
C
C        Apply a force of 1 on all Z nodes at Z=NPLONG:
         NFDOF = NDOF * NPTHICK * NPWIDE
         DO I = (MAXEQ-NFDOF+NDOF),MAXEQ,NDOF
            IF(NEWNUM(I) .LE. 0) THEN
C              DOF already has a specified displacement:
               PRINT *,'BBC: DOF',I, 'has conflicted boundary conditions'
               STOP 'FATAL ERROR IN SUBROUTINE BBC'
            END IF
            B(I) = 1.0D0
         END DO
C
```

```
C       Reorder {B} for new numbering system:
        DO I = 1,MAXEQ
           INEW = NEWNUM(I)
           IF(INEW .GT. 0) THEN
C             Value to be determined:
              B(INEW) = B(I)
           ELSE IF(INEQ .EQ. 0) THEN
C             Value constrained to 0:
              B(INEW) = 0.0D0
           ELSE
C             Value constrained to a nonzero value:
              INEW = -INEW
              B(INEW) = BIGNUM*B(I)
           END IF
        END DO
        CALL FMSPOP (MYNAME)
        RETURN
        END
C=======================================================================
        LOGICAL FUNCTION ASK(QUESTION)
C=======================================================================
        CHARACTER* (*) QUESTION
        CHARACTER*1 IYN
        WRITE(6,2000) QUESTION
        READ (5,1000) IYN
        IF( (IYN .EQ. 'Y') .OR. (IYN .EQ. 'y') ) THEN
           ASK = .TRUE.
        ELSE
           ASK = .FALSE.
        END IF
        RETURN
 1000   FORMAT (A)
 2000   FORMAT (1X,A,' (y,n)>')
        END
C=======================================================================
        INTEGER FUNCTION ASK_I(STRING)
C=======================================================================
        CHARACTER* (*) STRING
        WRITE(6,2000) STRING
        READ (5,*) ASK_I
        RETURN
 2000   FORMAT (1X,A,'>')
        END
```

Copyright © Multipath Corporation

6. ACHIEVING MAXIMUM PERFORMANCE WITH FMS

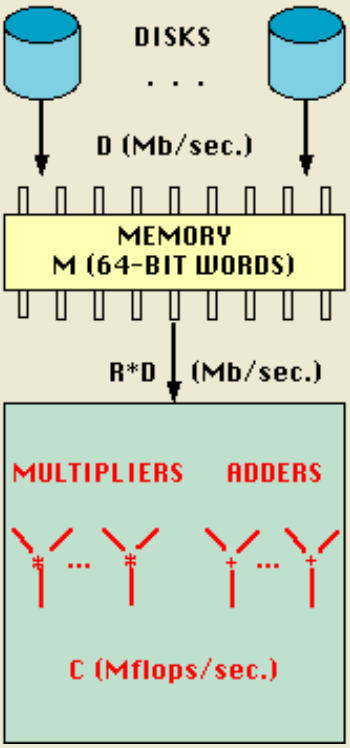
6.1 Balancing Processing and Disk Speeds with Memory Size
6.2 Time and Storage Requirements
6.3 Data Alignment
6.4 Memory Management
6.5 File Striping
6.6 Non-Uniform Memory Access (NUMA) Machines
6.7 Direct I/O
6.8 Large Pages

Copyright © Multipath Corporation

6.1. BALANCING PROCESSING AND DISK SPEEDS WITH MEMORY SIZE

In a typical computer, data, which is stored on disk, flows into memory, where it is held until required by the processors. Data then flows from memory into the processors, which perform multiplications and additions.

The following parameters are used to characterize the computer:



D = Disk transfer rate

(Megabytes per second).

D is the flow rate from disk to memory. If several disks are performing transfers in parallel (file striping), D is the aggregate transfer rate of all disks operating together when a single file record is transferred.

M = Memory size

(64-bit words).

M is the free memory available for buffer space in 64-bit words. M does not include memory required for the operating system, program instructions, or other program data.

C = Processor speed

(Millions of floating point operations per second, Mflops).

When several processors are operated in parallel, C is the aggregate sum of all processors operating together.

R = Reuse

(operations per word).

R is the number of times, on the average, the processors use each memory word for a multiply or add operation. R is determined by the algorithm.

In a properly balanced system, the data flowrate from the disks, times the reuse from memory, equals the computational rate as shown by the following:

```
DR = 8C
```

If DR < 8C, the process is I/O bound.

If DR > 8C, the process is CPU bound.

The reuse, R, is determined by the algorithm being used. For factoring PROFILE matrices, R is 2B, where B is the half bandwidth of the matrix. The factor 2 accounts for one multiplication and one addition per term.

During vector solution, R is 2(NUMRHS), where NUMRHS is the number of right-hand side vectors. For most machines, the ratio 4C/D exceeds the number of right-hand side vectors and the solution is I/O bound. The solution subroutines in ***FMS*** are designed to process multiple right-hand side vectors simultaneously to minimize I/O time. If the number of right-hand side vectors is small, you may also store them in memory to eliminate I/O.

The reuse R is different for matrices stored in PROFILE format and BLOCK format.

- **PROFILE format matrices**

  The number of vectors of length L which can be stored in M words of memory is N = M/L. The reuse is 2*N, allowing for additions and multiplications. The amount of memory to balance I/O and processing becomes the following:

  ```
  M = (4CL)/D
  ```

  The above equation shows that if the matrix bandwidth or vector length (L) doubles, the amount of memory (M) should double. If the computational speed (C) doubles, the amount of memory should double. If the disk transfer rate (D) doubles, only half as much memory is required.

- **BLOCK format matrices**

  The memory required to store one block of dimension L is M=L**2. where **2 means squared. The reuse is 2*L, allowing for additions and multiplications. The amount of memory per block to balance I/O and processing becomes:

  ```
  M = (4C/D)**2
  ```

  The above equation shows that if the computational speed (C) doubles, the amount of memory should increase by a factor of 4. If the disk transfer rate (D) doubles, only one fourth as much memory is required.

The most cost effective method of increasing performance is to increase the disk transfer rate, D. The ***FMS*** File Striping System was designed to do that task.

These equations can be used to estimate machine configurations. Any specific machine will be used for a variety of applications and problem sizes. These equations should be considered only as a guideline to prevent configurations which are extremely I/O bound or CPU bound.

Copyright © Multipath Corporation

6.2. TIME AND STORAGE REQUIREMENTS

The processing time and storage requirements for ***FMS*** depend on the number of equations and vectors, matrix profile, matrix symmetry, data type and operation being performed.

The following table lists the number of words required to store matrices and vectors.

Storage Requirements (8-byte words)

Data Type		Real		Complex	
Matrix Symmetry		**Symmetric**	**Nonsymmetric**	**Symmetric**	**Nonsymmetric**
Matrix	**Profile**	NB	N(2B-1)	2NB	2N(2B-1)
	Full	N(N+1)/2	NN	N(N+1)	2NN
Vector		NX		2NX	

Values specified are in units of words (8-bytes).

N = Number of equations.
B = Average half bandwidth (from FMSBW).
X = Number of vectors.

Disk storage requirements are usually specified in megabytes (Mbytes). To convert the values in the above table to Mbytes, multiply by 8 and divide by 1,000,000. For example, a full complex nonsymmetric matrix with 20,000 equations would require 6,400 Mbytes of disk storage.

The following table lists the number of floating point operations required to factor and multiply a matrix. The number of operations listed includes addition and multiplication. The operation count for solution includes forward reduction and back substitution.

Floating Point Operations (F)

Data Type		Real		Complex	
Matrix Symmetry		**Symmetric**	**Nonsymmetric**	**Symmetric**	**Nonsymmetric**
Factor	**Profile**	NEE	2NEE	4NEE	8NEE
	Full	NNN/3	2NNN/3	4NNN/3	8NNN/3
	Profile	2XN(2B-1)		8XN(2B-1)	

Solve or Multiply	**Full**	2XNN	8XNN

Values specified are the number of floating point operations (add and multiply).

N = Number of equations.
B = Average half bandwidth (from FMSBW).
E = Effective factoring bandwidth (from FMSBW).
X = Number of vectors.

You may use the operation counts in the table above to estimate processing time. The performance of scientific computers is usually rated in units of Mflops (Millions of floating point operations per second). To estimate processing time, you may use the following equation:

T = F/(1,000,000 C)
T = processing time (seconds)
F = number of floating point operations from the table above.
C = processing speed (Mflops).

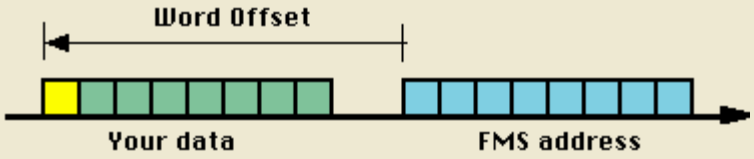
Copyright © Multipath Corporation

6.3. DATA ALIGNMENT

The arrays passed to ***FMS*** must be aligned on natural boundaries, which are:

- Integer, 4 or 8 bytes, depending on the machine,
- Real, 8 bytes,
- Complex, 16 bytes.

In order to achieve maximum performance on memory resident problems, ***FMS*** maps directly onto the data dimensioned in your program. ***FMS*** accomplishes this by reaching into the address space of your program using arrays located in ***FMS***.



The figure above shows a line representing increasing address space. At the lower end of address space is your program containing data to be referenced by ***FMS***. At a higher point in memory is ***FMS***. By using negative subscripts in an array, ***FMS*** may directly address your data.

In order for ***FMS*** to address data in your program, your data must begin on an address boundary that is an integer multiple of the word length. For example, if you are processing 64-bit data on a byte-addressable machine, your data must begin on an address that is divisible by 8.

You may control the address alignment of your data by the following methods:

- Allocate the memory from the ***FMS*** memory pool by calling one of the memory management routines FMSIGT, FMSRGT, FMSCGT to allocate integer, real or complex data. ***FMS*** automatically aligns data in its memory pool on the appropriate boundary.
- Place your data in a properly aligned common block.
  When addresses are generated for your program during linking, your computer may be directed to locate the common block on a specified address boundary.
  The order that you place data in a common block is also important. If possible, place integers, reals and complex data in separate common blocks as shown:

```
COMMON/IDATA/IARRAY
INTEGER IARRAY
COMMON/RDATA/RARRAY
REAL*8 RARRAY
COMMON/CDATA/CARRAY
```

```
COMPLEX*16 CARRAY
```

If you must mix data types within a common block, the longest word data types should be first. For example, all complex data must be at the beginning of the common block, followed by real data then integer data.

```
COMMON/DATA/CARRAY,RARRAY,IARRAY
COMPLEX*16 CARRAY
REAL*8 RARRAY
INTEGER IARRAY
```

This will prevent an odd number of short words from destroying the alignment of longer word data.

FMS automatically checks the alignment of your data on each appropriate subroutine call. If an error is found, fatal error condition, DATA DOES NOT START ON AN APPROPRIATE ADDRESS, is printed along with the array address information.

Copyright © Multipath Corporation

6.4. MEMORY MANAGEMENT

FMS contains a memory management system that you can use for scratch data storage. When FMSINI is called, a pool of memory is created. ***FMS*** allocates memory from this pool as required.

The size of the memory pool is determined by the ***FMS*** parameter MAXMD. ***FMS*** keeps track of the memory used out of the pool with the Parameter MDUSED. At any point you may determine how much memory is left in the pool as follows:

```
CALL FMSIGT('MAXMD', MAXMD)
CALL FMSIGT('MDUSED', MDUSED)
MDLEFT = MAXMD - MDUSED
```

You may allocate memory from this pool for scratch data storage. This has the following advantages for your application:

- **Less Programming Errors**

  Dynamically allocating storage that is the exact size produces less programming errors than arrays having fixed dimensions. As the problem size changes, the storage automatically changes also. If the code works on a small problem, it will also work on a large one. By contrast, fixed dimension arrays provide no warning when they are exceeded, resulting in programming errors that are hard to trace.

- **Portability**

  All ***FMS*** subroutines have the same application program interface on all platforms. This means that you do not need to change your application as you move between platforms. ***FMS*** handles the details of implementing memory management correctly on each platform.

- **More Efficient Code**

  By allowing arrays to adjust dynamically, your data for each problem will be packed into just the right amount of memory. This makes better use of the cache than data that is more distributed.

- **Shared Memory**

  The memory in the ***FMS*** memory pool is automatically shared among all processors. If you are using the ***FMS*** parallel processing utilities to run your code in parallel, the shared problem data must be in the ***FMS*** memory pool.

The first step in using memory in the ***FMS*** memory pool is to declare an array in your program that you will use to reference the allocated memory. There are currently two mechanisms used to do this.

1. **Machines supporting FORTRAN pointers**

   If your machine supports the FORTRAN POINTER statement, the reference arrays are obtained by including the POINTER statements for the arrays and obtaining the pointer values by calling FMSIGT, as follows:

```
POINTER (CMD PTR, CMD)
```

```
POINTER (RMD_PTR, RMD)
POINTER (IMD_PTR, IMD)
COMPLEX*16 CMD(0:1)
REAL*8     RMD(0:1)
INTEGER    IMD(0:1)
  ...
CALL FMSIGT ('MEMPTR',CMD_PTR)
CALL FMSRGT ('MEMPTR',RMD_PTR)
CALL FMSCGT ('MEMPTR',IMD_PTR)
```

2. **Machines not supporting FORTRAN pointers**

If your machine does not support the FORTRAN POINTER statement, the reference arrays are obtained as follows:

```
COMMON CMD
EQUIVALENCE (IMD, RMD, CMD)
COMPLEX*16 CMD(0:1)
REAL*8     RMD(0:1)
INTEGER    IMD(0:1)
```

On most machines, ***FMS*** automatically aligns BLANK COMMON on a 16-byte boundary.

You should not be concerned if you use BLANK COMMON for other purposes. ***FMS*** does not store any data in CMD(0:1). It simply uses the arrays IMD, RMD and CMD as reference points for the allocated memory.

Once the reference arrays are established in your subroutine, you call FMSIMG to allocate LEN integer words starting at IMD(LOC), FMSRMG to allocate LEN REAL*8 words starting at RMD(LOC), or FMSCMG to allocate LEN COMPLEX*16 words starting at CMD(LOC).

The value of LOC returned is relative to the array value you provide on the call. For example, if you make the call:

```
CALL FMSIMG (IMD(10), LOC, LEN)
```

the allocated memory will start at IMD(10+LOC).

You may make the value of LOC be an actual subscript of the array by dimensioning the array to start at 0,

```
INTEGER IMD (0:1)
```

Then when you call the memory allocation subroutine

```
CALL FMSIMG (IMD, LOC, LEN)
```

the allocated memory will start at IMD(LOC).

When you no longer require the storage, you can return it to the pool by calling

```
CALL FMSIMR(IMD,LOC,LEN)
CALL FMSRMR(RMD,LOC,LEN)
CALL FMSCMR(CMD,LOC,LEN)
```

If you want to list the contents of the ***FMS*** memory pool, you can include the statement

```
CALL FMSCST('SHOW','MEMORY')
```

You should return the memory you allocated to the memory pool before calling the ***FMS*** matrix algebra subroutines. This will prevent memory from being idle during ***FMS*** processing.

If you use the ***FMS*** parallel processing utilities to run part of your application in parallel, you may also call the ***FMS*** memory management routines from within your application. Each process will allocate a private region from the memory pool. For example, if you have 8 processes running and each calls the ***FMS*** memory management routines to allocate a region of memory, a total of 8 regions will be allocated.

Copyright © Multipath Corporation

6.5. FILE STRIPING

FMS includes an extensive file striping system that provides the following benefits:

- **Increased Bandwidth.**

  Multiple disks are operated in parallel to transfer data between disk and memory more rapidly (file striping).

- **Increased Capacity.**

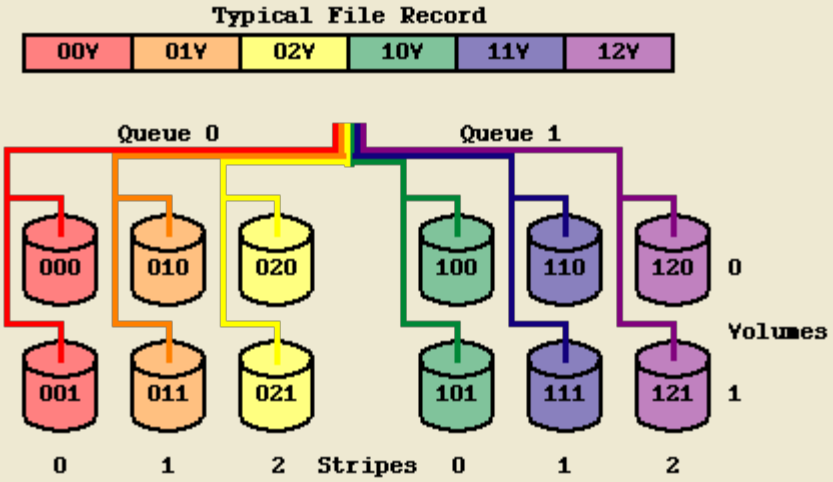
  Multiple disks are bound together to create files much larger than the physical drives (volume binding).

- **Distributed Memory.**

  Data transfers between disk and memory are distributed throughout the memory system, even if the memory is physically partitioned.

- **Asynchronous I/O**

  Transfers between memory and disk occur in parallel with processing to reduce run time.



FMS accomplishes these benefits by using a multi-level storage scheme for each file record. At the highest level, data is divided among I/O "queues". Each queue is responsible for managing the data in its memory region. Disks may physically be attached directly to the memory region or be at a central location.

Within each queue, the data is further divided into "stripes". Running in parallel, the stripes transfer their part of the record to their individual disk.

Each stripe may consist of multiple "volumes", which are used to increase capacity.

The figure above shows a ***FMS*** file system having 2 queues, 3 stripes per queue and 2 volumes per stripe.

Only symmetrical configurations are supported. The capacity of each volume must be identical.

The total speedup is the number of queues times the number of stripes per queue. For this example, data would be transferred at a rate of (2)(3) = 6 times faster than each individual disk.

The total capacity is the total number of volumes times the capacity per volume. For this example, the total storage would be (2)(3)(2)=12 times the data that could be stored on an individual disk.

When a file is opened for your application, ***FMS*** physically opens a file on each queue and stripe to hold the data. As you write each to the file, ***FMS*** divides your records up equally among the queues and stripes. When the volume becomes full, ***FMS*** automatically switches to the next volume. Because the configuration is symmetrical, all volumes will be switched at the same time.

FMS numbers the queues, stripes and volumes from 0 through 9. The 3 digit queue/stripe/volume number uniquely identifies each disk. For example, disk 021 in the above figure is in queue 0, stripe 2 and volume 1. The maximum number of queues, stripes per queue and volumes per stripe is 10 each, for a total of 1000 volumes.

Using the ***FMS*** File Striping System

The ***FMS*** File Striping System is easy to use. Simple follow these steps:

1. Set the ***FMS*** Parameters NUMIOQ, NUMSTP, NUMVOL and MBYTES.
   These parameters specify the number of queues, number of stripes per queue, number of volumes per stripe and the size of each volume in megabytes. If you are using only one volume per stripe, the parameter MBYTES is not required. ***FMS*** assumes that the files you create will fit on a single volume in each stripe.
   Usually these parameters are set in the license file so they may be easily changed if the hardware configuration is modified.

Example

```
FMSSET
NUMIOQ=2
NUMSTP=3
NUMVOL=2
MBYTES=1024
RETURN
```

2. Create directories FMSDISKqsv
   FMS accesses the file system through directories you provide. The directories are named
   `FMSDISKqsv`

   where qsv is the queue/stripe/volume number. One directory is required for each volume.

   The performance and capacity you obtain depends on where you place these directories. It is important to distinguish between these directories used by ***FMS*** and physical disks. You may place the directories anywhere in the file system. One or more directories can be placed on a single physical disk. In an extreme example, it is possible to have all ***FMS*** directories reside on a single disk.

To obtain the best performance, you should balance the disk stripes among the physical disks to achieve a uniform transfer rate. For example, if you have one disk that transfers data twice as fast as another, place two stripes on the faster disk.

The capacity is limited by the space available in the smallest directory. You may want to rearrange the files on your system to provide ***FMS*** the maximum space.

Example

In the above figure, you would create the following 12 directories (UNIX notation used):

Directory Created	Queue	Stripe	Volume
mkdir /disk0/FMSDISK000	0	0	0
mkdir /disk1/FMSDISK001	0	0	1
mkdir /disk2/FMSDISK010	0	1	0
mkdir /disk3/FMSDISK011	0	1	1
mkdir /disk4/FMSDISK020	0	2	0
mkdir /disk5/FMSDISK021	0	2	1
mkdir /disk6/FMSDISK100	1	0	0
mkdir /disk7/FMSDISK101	1	0	1
mkdir /disk8/FMSDISK110	1	1	0
mkdir /disk9/FMSDISK111	1	1	1
mkdir /disk10/FMSDISK120	1	2	0
mkdir /disk11/FMSDISK121	1	2	1

3. Create links pointing to the directories.

FMS finds the directories in the file system by links you provide in the current working directory. One link is required for each directory.

Example

In the above figure, you would create the following 12 links (UNIX notation used):

```
ln -s /disk0/FMSDISK000 FMSDISK000
ln -s /disk1/FMSDISK001 FMSDISK001
```

```
ln -s /disk2/FMSDISK010 FMSDISK010
ln -s /disk3/FMSDISK011 FMSDISK011
ln -s /disk4/FMSDISK020 FMSDISK020
ln -s /disk5/FMSDISK021 FMSDISK021
ln -s /disk6/FMSDISK100 FMSDISK100
ln -s /disk7/FMSDISK101 FMSDISK101
ln -s /disk8/FMSDISK110 FMSDISK110
ln -s /disk9/FMSDISK111 FMSDISK111
ln -s /disk10/FMSDISK120 FMSDISK120
ln -s /disk11/FMSDISK121 FMSDISK121
```

One final tuning note. Operating too many disks in parallel on a single SCSI bus may exceed the bandwidth of the bus or the controller. On most systems, this is between 3 and 4 disks. The best solution is to divide the stripes evenly among the I/O busses.

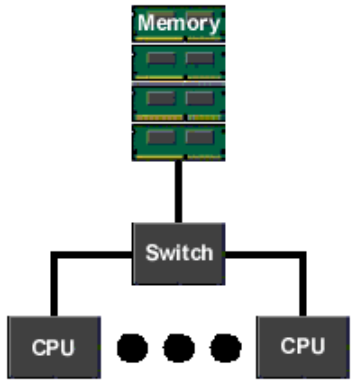
Copyright © Multipath Corporation

6.6. NON-UNIFORM MEMORY ACCESS MACHINES (NUMA)

Architectural Background

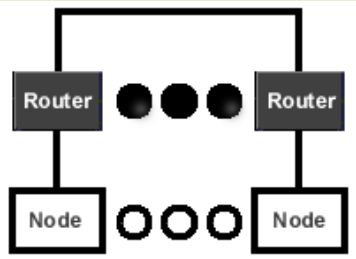
NUMA machines provide a linear address space, allowing all processors to directly address all memory. This feature exploits the 64-bit addressing available in modern scientific computers. The advantages over distributed memory machines include faster movement of data, less replication of data and easier programming. The disadvantages include the cost of hardware routers and the lack of programming standards for large configurations. ***FMS*** solves the programming standard issue by providing a set of programming tools that are portable across all NUMA architectures, as described below.

Node of a NUMA machine



The fundamental building block of a NUMA machine is a Uniform Memory Access (UMA) region that we will call a "node". Within this region, the CPUs share a common physical memory. This local memory provides the fastest memory access for each of the CPUs on the node. Modern processors contain many CPUs within the processor itself. Multiple processors each having many CPUs can be configured as a node.

NUMA machine



For larger configurations, multiple nodes are combined to form a NUMA machine. When a processor on one node references data that is stored on another node, hardware routers automatically send the data from the node where it is stored to the node where it is being requested. This extra step in memory access results in delays, which can degrade performance. Small to medium NUMA machines have only one level of memory hierarchy; data is either local or remote. Larger NUMA machines use a routing topology, where delays are greater for nodes further away.
One design goal of a NUMA machine is to make the routers as fast as possible to minimize the difference between local and remote memory references.

The performance of an individual application depends on the number of nodes used. If only two nodes are used and the memory is placed randomly, there will be a 50% chance that memory references will be local. As the

number of nodes increases, this probability decreases. The ***FMS*** Programming Tools described in the next section overcome the scaling issues associated with large NUMA architectures.

Programming

The goal for optimal programming of NUMA machines is to maximize references to local memory on the node while minimizing references to remote memory. ***FMS*** contains the following unique "hooks" into the operating system that provide the control necessary to achieve this goal:

- **Thread Binding.**

  The compute and I/O threads managed by ***FMS*** may be physically bound to specific processors or nodes. This is the first step necessary in establishing the affinity between executing threads and the physical memory they reference.

- **Memory Placement.**

  When ***FMS*** allocates memory, it may be explicitly placed on the processor's local node.

FMS automatically distributes each matrix and vector record uniformally among the nodes. Each processor is then assigned the portion of the work that corresponds to the data on its local node. The computational sequences are ordered to minimize the references to remote data.
The Parallel Programming Tools available with ***FMS*** provide these same hooks for you to achieve optimal NUMA performance on the non-***FMS*** part of your application. These tools provide a common portable interface across all NUMA machines.

Using FMS on NUMA machines

FMS contains a single Parameter NUMAFL which turns on or off all special NUMA features. By default, this Parameter is set off, leaving the placement of threads and memory up to the operating system. This provides the best setting when the machine is used in a multi-user environment to run a large number of jobs of different priorities.
When NUMAFL is set, your application will explicitly control which CPUs are used and where allocated memory is placed. This setting provides the best performance when all or part of the machine may be dedicated to your application.

When operated in NUMA mode, the ***FMS*** Parameters MYCPU1 and MAXCPU control the starting and number of CPUs used. When the ***FMS*** Memory Management routines are called to allocate memory by a child thread, the memory is automatically placed on the node where the child thread is executing. When memory is allocated by the parent, it is distributed among all the nodes in a round-robin fashon using a stride of MAXLMD. Specifying a value of 0 for MAXLMD will result in ***FMS*** allocating all the memory on the parent's node.

These ***FMS*** Parameters provide all the control necessary for optimal NUMA performance, either within ***FMS*** or your application.

There are several reasons why a NUMA operating system will not automatically do the right thing all of the time.

First, the operating system cannot anticipate the memory access patterns of your application. By default, memory is usually placed on the node which generates the first reference (first touch placement). If your application later enters a different phase with different access patterns, almost all of the memory references will probably be remote. Some attempts have been made to dynamically move data to the nodes generating the most references (page migration). However this is after the fact and generally gets out of phase with the current reference pattern, resulting in decreased rather than increased performance.

Second, if the system is oversubscribed (more jobs than processors), processors must be shared. Under these conditions, threads are generally assigned on the basis of processor availability, not on the placement of data which they will reference (which is unknown by the operating system). As a result, once a thread changes processors, most of the memory references will probably be remote.

When a system is freshly booted, all memory is available. The first job will have its choice of where to place the memory. The remaining jobs, however, will be restricted to the amount of memory currently available. A small job which requests lots of memory will spill over to adjacent nodes. The longer the system runs, the dirtier the memory system becomes, making it increasingly impossible for optimal memory placement.

The ***FMS*** NUMA programming tools described above solve all these problems by providing you with explicit control over thread and memory placement.

Copyright © Multipath Corporation

6.7. DIRECT I/O

The ***FMS*** Parameter IOTYPE=2 provides the capability of transferring data directly between memory and disk, bypassing the intermediate buffer cache. These transfers can improve performance when the problem size exceeds the amount of physical memory.
By default on most systems, write and read operations first move the data to/from an intermediate buffer. Physical transfers to/from the disk occur as required under the control of the operating system.

If the problem size is less than the amount of physical memory, this intermediate buffer cache can reduce the number of physical transfers to disk. However, in this scenario, the ***FMS*** Parameter INCORE provides better performance by using the memory directly as an ***FMS*** incore file, which eliminates the moves to/from the buffer cache.

If the amount of data to be moved is larger than the amount of memory, the movement to the intermediate buffer is an unnecessary step, degrading performance. This direct I/O feature performs the write or read operations directly between memory and the physical disk device.

The ***FMS*** interface for direct I/O is at the most efficient point in the operating system. Because ***FMS*** communicates directly with the disk controller, the following restrictions are placed on the transfers:

1. The transfer must start on an aligned memory address.
   This is usually a page boundary.
2. The transfer must start on an aligned disk address.
   This is usually a disk sector, although the effective sector size may be increased when RAID controllers are used to stripe several physical disks.
3. The transfer length must be an integer multiple of disk units.
   This is usually a disk sector.
   FMS automatically increases the transfer length if required to satisfy this condition.
   You should check for undesirable side effects as discussed below.
4. The transfer length must not exceed a maximum length.
   FMS automatically breaks up transfers that exceed the maximum length.

When a ***FMS*** file is opened for Direct I/O, ***FMS*** determines the physical alignment and length requirements for the transfers. If the default sector size LSECTR does not satisfy these requirements, a fatal error is generated and the correct value of LSECTR is printed. This value is dependent on the specific disk configuration of your system and cannot be determined until the file is opened. If you encounter this error, set the correct value of LSECTR in the license file and rerun the job. This value of LSECTR will be valid for all files opened on that file system.

FMS automatically satisfied all the conditions for Direct I/O on transfers performed internally within ***FMS***.

When you perform transfers to/from ***FMS*** files using FMSWRT or FMSRED with direct I/O, you must assure that your transfers satisfy the above (4) conditions. The easiest way to satisfy the memory alignment condition is to allocate the memory from the FMS Memory Management system. All allocated memory will automatically start on an aligned memory address. If you use this address as the start of an array, then condition (1) above will be satisfied. In general, if you use an array dimensioned in your application, this condition will NOT be satisfied. You will need to compute an offset into your array where the alignment is satisfied and start the transfers at that address.

The second condition, aligned disk address, will be satisfied if you read/write whole disk records. ***FMS*** automatically starts each record on an aligned disk address. If you are reading part of a record, then you probably will need to read/write more than the required data to satisfy this alignment requirement.

The third condition, transfer length, is automatically handled by ***FMS***. By default, ***FMS*** automatically increases the length of transfers to satisfy this condition. However, this can have undesirable side effects if your application has not allocated sufficient memory corresponding to the increased length. For FMSWRT operations, you may overwrite useful data on disk if you are transferring part of a record. It is also possible, though unlikely, that the added transfer length will fall outside your address space, causing a segment violation.

For FMSRED operations, you may overwrite data in your application which lies beyond the array you are reading.

The best way to avoid these undesirable side effects is to allocate sufficient memory to hold a copy of the disk image size of the record. The record length for each file type is computed by ***FMS*** when the file is opened. This record length has been rounded up from the minimum required to satisfy the length and alignment requirements. The table below lists the file attribute which stores this record length for each file type where direct I/O is available:

File Record Length Computed By FMS

FILE TYPE	RECORD LENGTH
Vector File	LUX(4)
Matrix File [L] or [U]	LUA(5)

Note that relatively small files, including the matrix diagonals, are never transferred using direct I/O.

You may also direct ***FMS*** NOT to automatically increase the transfer length by specifying IOWARN=2.

The fourth condition, maximum transfer length, is automatically satisfied by ***FMS*** without any undesirable side effects.

If you use FMSROW, FMSCOL or FMSPUT to write data to a ***FMS*** matrix file, ***FMS*** automatically satisfies the conditions for direct I/O.

If you do not satisfy the conditions, the result is machine dependent. On some systems, a fatal error will occur. On

others, the I/O operations on the file will degrade to buffered I/O.

You may use the ***FMS*** Parameter IOWARN to check your transfers for direct I/O. To check all the conditions, include

```
IOTYPE=2
IOWARN=28
```

in the license file. All transfers which do not satisfy the conditions will print a warning message containing DIO.

It is normal for LENGTH warning messages to occur when ***FMS*** automatically increases the length of the transfer. You should assure that these increases do not producce undesirable side effects as discussed above. By letting ***FMS*** round up the transfer lengths, you avoid transferring extra data when buffered I/O is used.

Direct I/O is determined by the value of IOTYPE at the time that the file is opened. While IOTYPE is usually set in the license file to control all appropriate files, you may set it by calling FMSIST within your application before opening an individual file.

This is a high performance option worth including. The code changes required are usually minimal.

EXAMPLE

The following example shows how to write a ***FMS*** vector file containing NUMVEC vectors of COMPLEX*16 data.

```
C       Initialization for FMS memory management:
        POINTER (CMD_PTR, CMD)
        COMPLEX*16 CMD(0:1)
        CALL FMSIGT ('MEMPTR', CMD_PTR)

C       Open the file:
        CALL FMSOV (NUMEQ, 2, NUMVEC, 'LUX', LUX)
C       Find the record length:
        LENREC_R8  = LUX(4)
        LENREC_C16 = LENREC_R8/2

C       Allocate space for 1 record in memory:
        CALL FMSCMG (CMD, LOCREC, LENREC_C16)
C       Note that CMD(LOCREC) starts on an aligned address:

C       Write the vector file:
        LDISK      = 1
        DO NV = 1,NUMVEC
C          Fill CMD(LOCREC:LOCREC+LENREC_C16-1) with data:
           CALL FMSWRT (LUX(1), LDISK, CMD(LOCREC), LENREC_R8)
```

```
            LDISK = LDISK + LENREC_R8
         END DO

C        Return the memory:
         CALL FMSCMR (CMD, LOCREC, LENREC_C16)
```

Copyright © Multipath Corporation

6.8. LARGE PAGES

FMS includes support for large pages on systems where they are available. Large pages improve performance of the virtual memory system. Large pages are selected using the ***FMS*** Parameters LPAGE and MDHOW.
Virtual memory systems must translate the virtual addresses of your code and data into physical memory addresses before the memory can be accessed. These translations must occur for each instruction and data element referenced.

Virtual addresses are divided into two parts: a page number and an offset within the page.

The low order part of the address, which is the offset within the page, is the same for both virtual and physical addresses. The high order part, which is the page number, is different for each page.

In order to perform page number translations at machine speed, a table, called the Translation Lookaside Buffer (TLB), is maintained of the most recently referenced pages. If the page requested is within this table, processing continues at machine speed. However, if the page is not found, a TLB miss occurs. This results in a significant interruption of the processing.

When a TLB miss occurs, the following actions are performed:

1. The page table is scanned to determine if the requested page is in memory. If it is, its page number replaces one of the entries in the TLB and processing resumes.
2. If the requested page is not in memory, a page fault occurs and the requested page is read in from disk. During this process, execution of the task is suspended.

The performance impact of frequent TLB misses can be significant, typically 5% or more on well-tuned applications.

The number of entries in the TLB is relatively small, usually about 100. Keep in mind that this table must be scanned within the machine cycle for every referenced instruction and data element. The address space spanned by all entries in the TLB is also relatively small. For example, if there are 100 entries and the page size is 4 Kbytes, the total address range that can be used without a TLB miss is 400 Kbytes. This address space includes your instructions, data and occasionally parts of the operating system. If your application repeatedly runs through an address range larger than that amount, TLB thrashing will occur.

On some systems it is possible to change the page size of the allocated memory to significantly reduce the number of TLB misses. For example, if the page size is increased to 16 Mbytes and there are 100 TLB entries, then the total address range becomes 1.5625 Gbytes. Small pages are efficient for editing or transaction processing. However they are inefficient for large scientific applications.

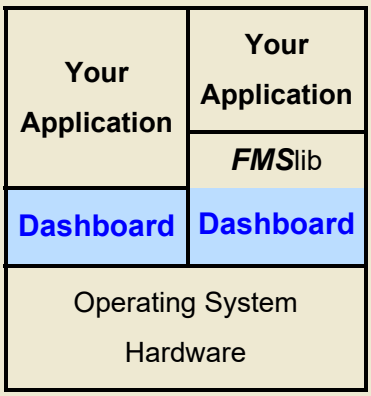
Copyright © Multipath Corporation

7. DASHBOARD PERFORMANCE MONITOR

What is **Dashboard**?

Dashboard is a performance monitoring library that creates and updates a website dynamically at runtime. Separate pages report the performance of key hardware and software components. The website may be viewed by any device supporting a web browser, including desktops, tablets and smartphones.
To get an overview of the capabilities of **Dashboard** you may run **MatrixWarrior**.

How is **Dashboard** used?



There are two ways to use **Dashboard**:

1. Instrument your application with calls to **Dashboard** routines, or

2. Use one of the ***FMS***lib utilities already instrumented with **Dashboard**. These utilities include:

- Matrix algebra,
- Parallel processing,
- GPU accelerators,
- Memory management, and
- Disk file striping.

When any of these ***FMS***lib utilities are used, **Dashboard** reports are automatically generated.

What reports does **Dashboard** generate?

Website pages generated by **Dashboard** are divided into three categories:

1. Hardware Reports:

Dashboard interfaces directly with hooks built into the operating system (Linux and Windows) to obtain configuration and performance information. **Dashboard** also uses functions provided by hardware manufacturers to obtain device-specific information. These reports are initialized by a single call at the beginning of your application. As your application is running, time varying information is automatically updated at a specified interval.

2. Application Reports:

Dashboard also provides functions which monitor the performance of your application. These include calls you can make at the beginning and end of subroutines. **Dashboard** uses the information you provide to maintain a call stack, listing the currently active routines and the order in which they were called. A call history is also maintained,

which lists the number of times each routine was called, the calling thread (parent or child) and the time used (CPU and Wall).

You may also provide **Dashboard** with the amount of ***Useful Work*** the routine is performing. Typical Useful Work items include floating point operations performed or bytes transferred. **Dashboard** combines this information with the routine timing to compute the rate of performing useful work. This information is then displayed as compute rate (flops/sec) or transfer rate (bytes/sec).

3. Custom Reports:

Dashboard can also generate application-specific pages to display information in the context of an application. As an example, ***FMS***lib includes a Performance page which provides job status and a summary of compute and transfer rates. This page also includes a picture of the matrices and vectors being processed.

If your application uses ***FMS***lib, **Dashboard** is already included. As an option, you can supplement the ***FMS***lib reports by including calls to your routines.

Where do I get **Dashboard**?

Dashboard is distributed as part of the ***FMS***lib library. You may obtain **Dashboard** by downloading ***FMS***lib from Multipath's website at www.fmslib.com.

Installing **Dashboard** in your application

The first step is to add a nonzero value of IWATCH to the license file between the `FMSSET` and `RETURN` lines as follows:

```
FMSSET
IWATCH=99
RETURN
```

This will instruct the ***FMS*** library to generate **Dashboard** reports. If ***FMS***lib is already installed in your application, this is all that is required.

1. Hardware Reports

To generate reports on the installed hardware and its performance, make the following two calls in your application:

```
CALL FMSINI()
```

During this call, **Dashboard** performs a series of initialization steps. A list of these steps is included in the subroutine description for FMSINI. This call should be at the beginning of your application so the overall timing reports will be accurate.

Next,

```
CALL FMSEND()
```

The call to FMSEND should be at the end of your application.

These two calls are all that is required to generate the **Dashboard** hardware reports. If you are using ***FMS***lib, these calls are already in your application.

2. Application Reports

You can instrument your application to provide performance and timing reports by placing your routines on the **Dashboard** call stack as follows:

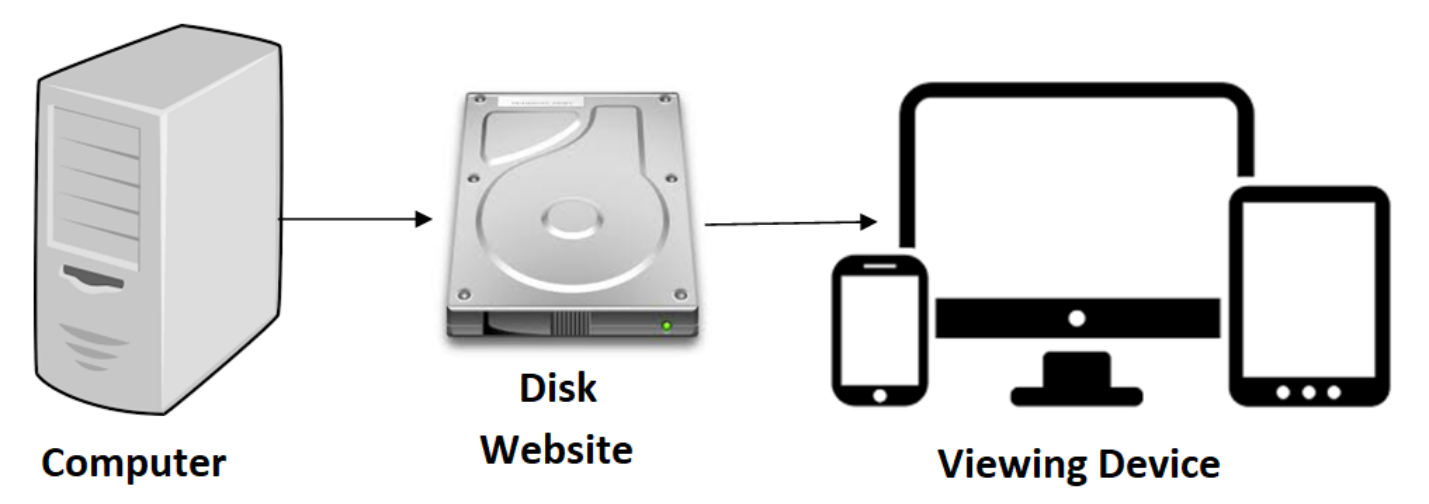
```
SUBROUTINE MYSUB(...)
CALL FMSPSH("MYSUB")
...
CALL FMSPOP("MYSUB")
END
```

The first call places the subroutine on the stack and the second call removes it. **Dashboard** creates a time stamp on both calls to compute the net timing (CPU, WALL) spent in the routine. You may instrument all or part of your application. Subroutines can be nested. **Dashboard** also records which thread made the call, parent or child. The first call to FMSPSH is assumed to be your application name. This name is included on the header of each page.

3. Custom Reports

All timing data collected by **Dashboard** is available to generate an application-specific web page. As an example, the Performance page was designed to display information important to ***FMS***lib. If you have an application that could benefit from a custom **Dashboard** page, please contact Multipath for more information.

How **Dashboard** works



The display of information involves the following 3 steps:

1. **Dashboard** and the instrumented application run on a computer. This is the machine and application that are monitored. It can be any machine running the Linux or Windows operating system, from a laptop to high

performance server containing GPU accelerators.

2. **Dashboard** creates a website as files on a disk. This disk may be in the instrumented computer or on any remote fileserver accessible from the instrumented computer. By default, webpages are stored in the current working directory used to run the application. You may change this location by specifying the directory in the environment variable FMSHTML.

3. You view the website from any viewing device that contains a web browser, including a pc, tablet or smartphone.

The flow of information is in one direction: from the computer, to the disk, to your viewing device. You do not interact directly with` **Dashboard**. This design allows you to monitor the performance of a computer which may be located in a restricted area.

How to control **Dashboard** reports

Controlling Report Generation

Parameter	Default	Description
IWATCH	0	Which reports are generated A value of 115 will generate a movie.
NSUPD	5 sec.	How frequently **Dashboard** updates the pages
NSREF	3 sec.	How frequently the browser reloads the pages
NSMOVE	2 sec.	The pause between movie frames
MAXMOV	1000 frames	Maximum number of movie frames

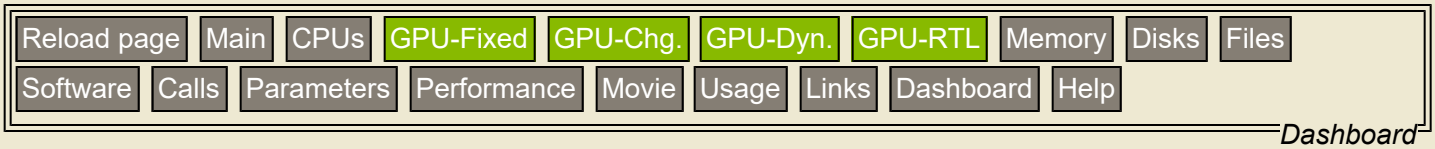
The following Parameters control what reports **Dashboard** generates and when they are generated:

- IWATCH determines ***what*** reports are generated. Several separate options are available, including making a movie of the Performance page. To enable report generation, set the ***FMS*** Parameter IWATCH to a nonzero value.
- NSUPD determines ***when*** pages are updated. For most applications it may not be necessary to generate new pages for all intermediate results. **Dashboard** records when the contents of a page have changed. When NSUPD seconds have elapsed since the last update, and the page has changed, a new version is generated. Increasing NSUPD will reduce the number of intermediate pages. In all cases, the first and last pages are generated.
- NSREF determines how often your browser reloads pages. You may force a reload at any time by clicking on the **Reload page** tab or typing control-r.
- NSMOVE determines interval between movie frames.
- MAXMOV is the maximum number of movie frames to generate. ***FMS*** divides data into blocks which fit in

memory. A separate movie frame is generated as each block is processed. When a large problem stored on disk is run on a machine with a small amount of memory, the number of operations on the blocks may become very large. Under these contitions this Parameter prevents an excessive number of movie frame files from being generated.

Dashboard Reports

The pages displayed by **Dashboard** depend on what functions have been implemented and what hardware is available. This section presents the results of implementing **Dashboard** in ***FMS***lib.



Dashboard

Main

This page lists the network node name of the machine.
You can include the name of your application in one of two ways:

1. Explicitly setting it,

```
CALL FMSCST(ANAME, 'My_Application')
```

or

2. **Dashboard** will use the name of the first routine put on the stack:

```
CALL FMSPSH('My_Application')
```

If your application includes calls to ***FMS***lib, the ***FMS*** logo is also included.

CPUs

This page lists the number and type of CPUs installed in the system. Note that for modern processors, the Clock Speed listed may change with utilization.

GPU-Fixed

This page lists information about the GPUs that does not change during the run. If multiple GPUs of different types are found, only the one(s) with the highest compute capability are listed. If no GPUs are available, pages GPU1 and GPU2 are not generated.
Nvidia uses two numbering systems: one by the Run Time Library (RTL) and another by the Nvidia Management Library (NVML). These may not be the same and can change on reboot. Correlation between these numbering systems is established by the PCI bus location of the device. This page lists the devices in the RTL order and shows the corresponding NVML number for reference. If you use the nvidia-smi utility to change the properties of a device, you should use the NVLM number.

Dashboard needs to know where the Nvidia NVML utilities are located in your file system. By default, the following directories are used:

```
(Linux)   /usr/bin
(Windows) "C:\Program Files\NVIDIA Corporation\NVSMI"
```

The nvidia-smi utility is also in this directory. If this utility is in a different directory, then the ***FMS*** Parameter CUPATH must be included in the license file to provide the name of this directory. If **Dashboard** cannot find this directory, the application will still run but some of the hardware information about the GPUs will be missing.

GPU-Chg.

This page lists GPU information which may be changed, either by the application program or using the nvidia-smi utility. This includes limits on temperature, power and clock speed.

GPU-Dyn.

This page lists GPU information that changes with time. This includes temperature, fan speed (for workstations), utilization rates, ECC errors and power management. This page is automatically updated during the run.

GPU-RTL

This page lists GPU information from the Run Time Libaray. This may include some of the information obtained from NVML, where available.

Memory

This page lists the total amount of memory in the system. For applications which use the FMS Memory Management utilities, it also shows how that memory is allocated. On systems with GPUs, it lists the memory for each GPU and its address range.

Disks

This page lists the following tables:

- Disk Properties by Mount Point (Linux) or Drive Letter (Windows)
  This table provides useful information on the disks that are available for storing data.
- Disks used by ***FMS***
  For applications which use the ***FMS*** file system, this table lists information for each file stripe, including the space used and available, number of accesses, average transfer size and transfer rate.
- Disk Performance by File Type
  This table lists the overall performance of the disks for each type of file used by ***FMS***.
- Overall Disk Performance
  This table provides a summary of disk performance for synchronous and asynchronous transfers for both read and write operations.

Files

For applications using the ***FMS*** File Striping System, information on open ***FMS*** files, their stripe configuration and

performance are listed.

Software

This page lists the following information about the software being used by the application and ***FMS***:

- Program File Name

  The name of the file for the current running program.

- Files Used

  This list includes shared-object files used by the application that were referenced when the application was linked.

- Software

  Where appropriate, this table lists the version number of the ***FMS*** library, Intel and Nvidia libraries and Fortran and C compilers.

- Environment Variables

  This table lists the environment variables which are defined.

Calls

This page lists the subroutines which have been called. The order in each table is determined by the order of first call. The following tables are included:

- Application Subroutines

  Subroutines that you have instrumented with calls to FMSPSH and FMSPOP.

- ***FMS*** Main Subroutines

  FMS subroutines called directly by your application.

- ***FMS*** Callback Subroutines

  Subroutines you provided to define matrix data or special processing of diagonal pivots.

- ***FMS*** Internal Subroutines

  Subroutines called internally by ***FMS*** to measure the performance of specific utilities.

- ***FMS*** GPU Subroutines

  Kernel subroutines run on the GPUs.

- **Dashboard** subroutines used to generate the WEB site

Routines highlighted in yellow are currently active on the stack.

Parameters

This page lists the current values of ***FMS*** parameters which have been changed from default values. In some cases, changing one parameter will cause others to be changed also. Some of the parameters are output only and are computed by ***FMS***.

When ***FMS*** determines that a different value of a Parameter would improve performance, they are listed in the

Suggested Tuning Values table.

Performance

This is the application-specific page for monitoring ***FMS*** performance. It first lists the ***FMS*** routine currently executing with a link to its manual page and the total wall time used. All information on this page pertains to that routine. When a new routine is called, this page is reset.
This page contains one or more of the following sections:

- Status Bar

  The first bar shows the percent completion for the current routine, based on the total number of operations required and the number completed. The second bar shows the percent contribution of major tasks, based on the amount of wall time used.

- Times and Problem

  This section lists information about the current running job, including times, problem type and size, and performance of processors and disks. It includes an estimate of when the routine will complete, based on work to be performed and the current performance.

- Overall Performance

  This is the bottom line number. It lists the overall performance of all processors, not including any time spent waiting for I/O. The units are floating point operations per second. The heading changes as the performance increases.

- Component Performance

  This lists the individual performance of major subroutines and the contribution made by the group of CPUs and each of the GPUs. Performance of the individual hardware components is based on the amount work performed and the wall time used. Overall performance of the routine is based on the total amount of work done by all hardware components and the total wall time used. The total performance of a routine will always be less than the sum of the individual components, due to load balancing. This section also lists the model of the GPUs, if appropriate.

- Matrix Picture

  Where appropriate, a picture of the matrix and vector(s) are shown as they are blocked for processing. The status of each block changes as the computation proceeds. The area currently being updated is shown in red, the data being referenced is shown in light red and other colors are used to show the state of the data. When the computation is complete, this picture is removed.

- Time Table

  This section lists the time (sec) used by each of the components for each of the major routines. In a perfectly balanced system, each component would use the same amount of time. For systems with several powerful GPUs, the CPUs are often not used for computation.
  This table also lists other times, including the time **Dashboard** spends generating the WEB reports.

Movie

When IWATCH is set to generate a movie, this link provides a way to playback the movie. This link only appears after all the movie frames have been generated.

Usage

On systems running Linux, this page lists the overall usage of system resources since the job started. Memory performance information, including resident set size and page faults are listed both in absolute values as well as ratio to ***FMS*** memory.

Links

This page lists links to Multipath's home page and the documentation for ***FMS***lib, **MatrixWarrior** and **Dashboard**.

Dashboard

This page lists the number of times the contents of each page changed, the number of times the page was updated and the time (CPU, Wall) used. You may use this information to adjust NSUPD to increase or decrease how often pages get updated.

Help

For **MatrixWarrior**, this links to the Quick Start Guide. For ***FMS*** EXAMPLE problems, this links to documentation on how to run the example as well as a listing of the program code. For other applications, this links to the **Dashboard** documentation.

Copyright © Multipath Corporation

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Dashboard Performance Monitor

Running on Node dgx05

MatrixWarrior

The Battle Over Matrix Size

Powered by FMSlib

World's fastest solutions for

[A] {X} = {B}

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

CPU Description

Number of processing cores	40
Cores used by FMS	40
Cores per processor	20
Minimum Clock Speed (Ghz)	1.356
Maximum Clock Speed (Ghz)	3.570
Processor Model	Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Static Properties for GPUs 0 to 3 Used by FMS

GPU Number	RTL	0	1	2	3
	NVML	0	1	2	3
Device Type (RTL)		Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB
Product Name (NVLM)		Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB
Capability(major,minor)		(6.0)	(6.0)	(6.0)	(6.0)
Multiprocessors		56	56	56	56
Cores per Multiprocessor		64	64	64	64
64-bit flops/SM/Clock		64	64	64	64
Warp size		32	32	32	32
Clock Rate (GHz)		1.33	1.33	1.33	1.33
Serial Number		0324616230464	0324616230567	0324716027161	0324616230502
CPUs Used		1 - 8	9 - 16	17 - 24	25 - 32
PCI Bus ID (hex)		2	82	85	86
PCI Device ID (hex)		0	0	0	0
PCI	Cur.Gen.	3	3	3	3
	Max.Gen.	3	3	3	3
	Cur.Width	10	10	10	10
	Max.Width	10	10	10	10
	Bus	2	82	85	86
	Device	0	0	0	0
	Domain	0	0	0	0
	Device Id	15F810DE	15F810DE	15F810DE	15F810DE
	Bus Id	0000:02:00.0	0000:82:00.0	0000:85:00.0	0000:86:00.0
	Sub System Info	118F10DE	118F10DE	118F10DE	118F10DE
Global Memory (bytes)		17066885120	17066885120	17066885120	17066885120
Display Active		Enabled	Enabled	Enabled	Enabled
Display Mode		Disabled	Disabled	Disabled	Disabled
Persistence Mode		Disabled	Disabled	Disabled	Disabled
GPU Operating Mode	Current	unknown	unknown	unknown	unknown
	Pending	unknown	unknown	unknown	unknown
VBIOS Version		86.00.3A.00.03	86.00.3A.00.03	86.00.3A.00.03	86.00.3A.00.03

Inforom Version	**OEM**	1.1	1.1	1.1	1.1
	ECC	4.1	4.1	4.1	4.1
	Power	N/A	N/A	N/A	N/A
Supported Clock Throttle Reasons		N/A	N/A	N/A	N/A

 Copyright © Multipath Corporation

Dynamic Properties for GPUs 0 to 3 Used by FMS

GPU Number	**RTL**	**0**	**1**	**2**	**3**
	NVML	**0**	**1**	**2**	**3**
GPU Temperature		55 C	59 C	55 C	69 C
Fan Speed		N/A	N/A	N/A	N/A
Utilization	**Gpu**	0 %	7 %	69 %	29 %
	Memory	0 %	1 %	36 %	13 %
Memory Usage (Mbytes)	**Total**	17066	17066	17066	17066
	Used	16908	16908	16908	16908
	Free	158	158	158	158
Clocks (Cur./Max) (MHz)	**Graphics**	1189/1328	405/1328	405/1328	405/1328
	SM	1189/1328	405/1328	405/1328	405/1328
	Memory	715/715	715/715	715/715	715/715
Power (milliwatts)	**State**	P0	P0	P0	P0
	Draw	42469	50436	42571	54360
Clocks Throttle Reasons		None	None	None	None
ECC Enabled		Yes	Yes	Yes	Yes
ECC Errors Volatile	**Single Bit**	0	0	0	0
	Double Bit	0	0	0	0
ECC Errors Aggregate	**Single Bit**	0	0	0	0
	Double Bit	0	0	0	0
Compute Mode		NonExclusive	NonExclusive	NonExclusive	NonExclusive
Compute Mode		Default	Default	Default	Default
Power Management		Enabled	Enabled	Enabled	Enabled
Power Limits (milliwatts)	**Default**	250000	250000	250000	250000
	Current	250000	250000	250000	250000
	Minimum	125000	125000	125000	125000
	Maximum	N/A	N/A	N/A	N/A

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

FMS Memory Allocation: Total memory = 504 Gbytes.

Region	Starting Address			Length				Status	Thread	Contents
	Relative		Absolute	(Gbytes)	(8-byte words)	(bytes)	%			
	(8-byte words)	(bytes)	(bytes)							
1	0	0	46913122471936	428.233	57476451328	459811610624	85.0	Free		
2	Operating system			75.571	10142902785	81143222280	15.0	Not used by FMS		
GPU 0	Access by driver only			1.539	192374866	1538998928	9.1	GPU system memory		
	0	0	47373422166016	15.398	1924732846	15397862768	90.9	Alloc.	GPU 0	FMS Data
GPU 1	Access by driver only			1.539	192374866	1538998928	9.1	GPU system memory		
	0	0	47389595402240	15.398	1924732846	15397862768	90.9	Alloc.	GPU 1	FMS Data
GPU 2	Access by driver only			1.539	192374866	1538998928	9.1	GPU system memory		
	0	0	47405768638464	15.398	1924732846	15397862768	90.9	Alloc.	GPU 2	FMS Data
GPU 3	Access by driver only			1.539	192374866	1538998928	9.1	GPU system memory		
	0	0	47421874765824	15.398	1924732846	15397862768	90.9	Alloc.	GPU 3	FMS Data
GPU 4	Access by driver only			1.539	192374866	1538998928	9.1	GPU system memory		
	0	0	47438048002048	15.398	1924732846	15397862768	90.9	Alloc.	GPU 4	FMS Data
GPU 5	Access by driver only			1.539	192374866	1538998928	9.1	GPU system memory		
	0	0	47454221238272	15.398	1924732846	15397862768	90.9	Alloc.	GPU 5	FMS Data
GPU 6	Access by driver only			1.539	192382976	1539063808	9.1	GPU system memory		
	0	0	47470327365632	15.398	1924724736	15397797888	90.9	Alloc.	GPU 6	FMS Data
GPU 7	Access by driver only			1.539	192374866	1538998928	9.1	GPU system memory		
	0	0	47486500601856	15.398	1924732846	15397862768	90.9	Alloc.	GPU 7	FMS Data

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

File System on dgx05

Device	Mount Point	File Sys.	Total GB	Free GB	% Free
/dev/sda2	/local	ext4	439.019	438.948	100.0

Disks used by FMS

FMS Directory Link	Free GBytes Total GBytes	% Available	Access Type	Direction	Number Accesses	Avg. Transfer (MBytes)	Performance (MB/Sec.)
FMSDISK /raid/FMSDISK	7090.528 7095.401	100	Async.	Read	248	61256	2082
				Write	74	59842	1669

Disk Performance by File Type

File Type			Number Accesses	Avg. Transfer (MBytes)	Performance (MB/Sec.)
[AL]	Async.	Read	129	61504	2082
		Write	36	61504	1667
[D]	Sync.	Write	1	0	1386
[AU]	Async.	Read	118	61504	2081
		Write	36	61504	1670
SEG.	Sync.	Read	1	8	6439
		Write	1	8	1480

Overall Disk Performance

Transfer Type		Total		Overall MBytes/Sec
		MBytes	Time(Sec.)	
Async.	Read	15191488	7298	2082
	Write	4428288	2653	1669
Sync.	Read	8	0	6439
	Write	19	0	1441

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

No FMS files opened

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Program or File Name=MatrixWarrior

Files Used

File
/home/ryoung/Mike_8xv100-500k/MatrixWarrior
/usr/lib64/ld-2.17.so
/usr/lib64/libstdc++.so.6.0.19
/usr/lib64/librt-2.17.so
/usr/lib64/libc-2.17.so
/usr/lib64/libm-2.17.so
/usr/lib64/libpthread-2.17.so
/usr/lib64/libdl-2.17.so
/usr/lib64/libgcc_s-4.8.5-20150702.so.1
/usr/lib64/libcuda.so.387.26
/usr/lib64/libnvidia-fatbinaryloader.so.387.26
/usr/lib64/libnuma.so.1
/usr/lib64/libnvidia-ml.so.387.26

Software

Software	Version
Operating System Version Release	Linux #1 SMP Thu May 25 17:04:51 UTC 2017 3.10.0-514.21.1.el7.x86_64
FMS	Fast Matrix Solver FMS(R) version 70 built on 4/10/2018 for 64-bit Linux
MKL	Major:2018, Minor:0, Update:2, Status:Product, Built:20180127
CUDA Driver	9.1
CUDA Runtime	9.1
CUBLAS Version	9.1
NVML Driver Version	387.26

NVML Version	9.387.26
nvidia-smi	/usr/local/bin/nvidia-smi
ifort	which: no ifort in (/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/ibutils/bin:/sbin:/usr/sbin:/cm/extra/apps/Modules/
	sh: ifort: command not found
icc	which: no icc in (/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/ibutils/bin:/sbin:/usr/sbin:/cm/extra/apps/Modules/3.
	sh: icc: command not found

Environment Variables

Variable	Value
FMSHTML	/home/ryoung/Mike_8xv100-500k
FMSMAN	http://www.fmslib.com/fmsman
FMSDISK	/raid/FMSDISK

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Currently active highlight

Application Program and Subroutines

Name	TimesCalled		CPU		Wall		Ratio
	Parent	Child	Sec.	%	Sec.	%	
MatrixWarrior	1	0	59872.57	99.7	10574.28	98.3	5.7
CNUBLK	36	0	0.00	0.0	0.01	0.0	
CSUBLK	36	0	0.01	0.0	0.02	0.0	
TOTAL	73	0	59872.58	99.7	10574.31	98.3	5.7
OTHER	6002	2144	178.35	0.3	186.85	1.7	1.0
OVERALL	6075	2144	60050.93	100.0%	10761.16	100%	5.6

FMS Subroutines Called from Appplication

Name	TimesCalled		CPU		Wall		Ratio
	Parent	Child	Sec.	%	Sec.	%	
FMSINI	1	0	178.40	0.3	186.90	1.7	
FMSSET	2	0	0.01	0.0	0.02	0.0	
FMSIGT	10	0	0.00	0.0	0.00	0.0	
FMSIST	5	0	0.00	0.0	0.00	0.0	
CNDI	2	0	0.02	0.0	0.12	0.0	
FMSCM	2	0	0.15	0.0	0.38	0.0	
CNDF	1	0	59872.35	99.7	10573.74	98.3	5.7
FMSEND	1	0	0.00	0.0	0.00	0.0	
TOTAL	24	0	60050.93	100.0	10761.16	100.0	5.6
OTHER	6051	2144	0.00	0.0	0.00	0.0	0.0
OVERALL	6075	2144	60050.93	100.0%	10761.16	100%	5.6

FMS Callback Subroutines

Name	TimesCalled		CPU		Wall		Ratio
	Parent	Child	Sec.	%	Sec.	%	
cnublk	36	0	0.00	0.0	0.02	0.0	
csublk	36	0	0.01	0.0	0.02	0.0	

TOTAL	72	0	0.01	0.0	0.03	0.0	0.3
OTHER	6003	2144	60050.92	100.0	10761.13	100.0	5.6
OVERALL	6075	2144	60050.93	100.0%	10761.16	100%	5.6

FMS Internal Subroutines

Name	TimesCalled		CPU		Wall		Ratio
	Parent	Child	Sec.	%	Sec.	%	
fmsaut	1	0	0.49	0.0	0.90	0.0	
iowait	137	0	671.81	1.1	3239.29	30.1	
cncdat	36	0	2176.45	3.6	72.58	0.7	30.0
par	3432	0	0.00	0.0	0.00	0.0	
run	88	0	0.00	0.0	0.00	0.0	
sync	88	0	0.00	0.0	0.00	0.0	
cscdat	36	0	2228.25	3.7	74.21	0.7	30.0
cnbbf	8	0	2113.68	3.5	219.09	2.0	9.6
	Teraops				5234.80		
	Gigaflops		2476.63		23892.94		9.6
cstrsm	56	0	9199.70	15.3	1141.94	10.6	8.1
	Teraops				55538.99		
	Gigaflops		6037.04		48635.60		8.1
csgemm	140	0	42860.87	71.4	5268.95	49.0	8.1
	Teraops				272560.41		
	Gigaflops		6359.19		51729.54		8.1
TOTAL	4022	0	59251.25	98.7	10016.96	93.1	5.9
OTHER	2053	2144	799.68	1.3	744.20	6.9	1.1
OVERALL	6075	2144	60050.93	100.0%	10761.16	100%	5.6

FMS GPU Subroutines

Name	TimesCalled		CPU		Wall		Ratio
	Parent	Child	Sec.	%	Sec.	%	
creg	1	0	139.66	0.2	141.43	1.3	
cncubf	0	192	0.00	0.0	462.32	4.3	
	Teraops				607.86		
	Gigaflops/device				1314.80		
cscuts	0	704	0.00	0.0	9841.94	91.5	
	Teraops				58537.17		
	Gigaflops/device				5947.73		

cscumm	0	1248	0.00	0.0	42256.81	392.7	
	Teraops				274189.30		
	Gigaflops/device				6488.64		
TOTAL	1	2144	139.66	0.2	52702.49	489.7	0.0
OTHER	6074	0	59911.27	99.8	-41941.34	-389.7	0.0
OVERALL	6075	2144	60050.93	100.0%	10761.16	100%	5.6

DASHBOARD HTML Subroutines

Name	TimesCalled		CPU		Wall		Ratio
	Parent	Child	Sec.	%	Sec.	%	
hmain	2	0	0.00	0.0	0.02	0.0	
hcpu	2	0	0.00	0.0	0.02	0.0	
hgpu1	2	0	0.72	0.0	0.77	0.0	
hgpu2	330	0	3.98	0.0	8.45	0.1	
hgpu3	2	0	0.19	0.0	1.15	0.0	
hmem	6	0	0.00	0.0	0.02	0.0	
hdisks	268	0	0.22	0.0	0.53	0.0	
hfiles	268	0	0.13	0.0	0.47	0.0	
hsoft	4	0	0.00	0.0	0.02	0.0	
hcalls	6	0	0.00	0.0	0.01	0.0	
hpar	4	0	0.00	0.0	0.01	0.0	
hperf	329	0	8.69	0.0	8.43	0.1	1.0
husage	1	0	0.01	0.0	0.00	0.0	3.4
hlinks	1	0	0.00	0.0	0.00	0.0	
hdash	329	0	0.08	0.0	0.55	0.0	
hmovie	329	0	9.24	0.0	9.76	0.1	
TOTAL	1883	0	23.26	0.0	30.21	0.3	0.8
OTHER	4192	2144	60027.67	100.0	10730.95	99.7	5.6
OVERALL	6075	2144	60050.93	100.0%	10761.16	100%	5.6

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Changed FMS Parameters

Parameter	Default Value	Current Value	Changed In
MAXMD	33809677312	57476451328	License File
IOTYPE	0	2	Application
MEMGPU	0	15397797888	Application
IMMPCT	4	0	License File
ITRPCT	4	0	License File
IPRI	2	258	License File
IWATCH	0	115	License File
MAINC	0	1	Application
MFMAT	0	2	Application
NEQGPU	2240	1316	Application
MDATAU	0	2	Application
NUMDLK	0	9420	Application
DIAMIN	0	1.414213562373095e-06	Application
DIAMAX	0	7.071067811865475e+05	Application
PIVMIN	0	1.414213562373095e-06	Application
PIVMAX	0	1.414213557981077e+00	Application
RMMPCT	4.000000000000000e+00	0.000000000000000e+00	License File
RTRPCT	4.000000000000000e+00	0.000000000000000e+00	License File
ATERMS	0	2.500005000000000e+11	Application
FREEDU	0	7.090527713775635e+03	Application

Suggested Tuning Values

Parameter	Current Value	Suggested Value

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Performance Analysis for Subroutine CNDF ; Time used=2:56:13

COMPLETED				
31% I/O		50% MM	11% TR	

Total Processor Performance
(64-bit floating point operations per second)

Tera	Giga	Mega	Kilo	Flops
50	276	744	698	247

Component Performance (Gflops)

Routine	All	40 CPUs	GPU 1 33 °C 1312 MHz 57 Watts	GPU 2 34 °C 1312 MHz 55 Watts	GPU 3 34 °C 1312 MHz 56 Watts	GPU 4 32 °C 1312 MHz 54 Watts	GPU 5 31 °C 1312 MHz 55 Watts	GPU 6 33 °C 1312 MHz 56 Watts	GPU 7 35 °C 1312 MHz 56 Watts	GPU 8 31 °C 1312 MHz 54 Watts
Matrix Multiply CPU(0%)	51730	0	6502	6519	6530	6515	6493	6494	6504	6503
Triangle Solve CPU(0%)	48636	0	6173	6229	6221	6203	6100	6080	6161	6250
Diagonal Factor	23893	9	3123	3111	3130	3122	3133	3129	3129	3124

GPU model = Tesla V100-SXM2-16GB

Time (Sec.)

Routine	All	CPUs	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7	GPU 8
Matrix Multiply	5269	0	5240	5226	5217	5230	5247	5246	5238	5239
Triangle Solve	1142	0	1125	1114	1116	1119	1138	1142	1127	1111
Diagonal										

Factor	219	10	207	208	208	209	209	210	211	213
Total Compute	6630	10	6572	6549	6541	6558	6594	6598	6576	6563
Assembly	55									
I/O wait	3239									
HTML	28									
Other	677									
Overall	10574									

Times and Problem

Job Started	0	09:03:38 Thu Apr 12 2018	**Equations**	500000	**Compute Rate (Gflops)**	50277
Routine Started	186	09:06:44 Thu Apr 12 2018	**Vectors**	0	**Read Rate (MBytes/Sec.)**	2082
Current Time	10761	12:03:00 Thu Apr 12 2018	**Block Size**	63488 X 63488	**Disk->Memory Reuse**	126976
Estimated Completion	10761	12:03:00 Thu Apr 12 2018	**Data Type**	Complex	**Disk Speed / Compute Speed**	0.66

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

A movie showing the history of the Performance page is available online at Multipath's WEB site. Click HERE for viewing.

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

System Usage Statistics

User mode time (Sec.)	53290.195
System mode time (Sec.)	6762.819
Maximum FMS memory (Kbytes)	449034776
Maximum Resident Set Size (Kbytes) Ratio to FMS Memory	451288876 1.01
Page Faults without I/O Ratio to FMS Memory	9963901 0.09
Page Faults with I/O Ratio to FMS Memory	42 0.00
Number of times swapped	0
Number of file system reads	31112265744
Number of file system writes	9069212872

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Useful Links

FMSlib WEB site
FMS Manual
Dashboard Manual
MatrixWarror Manual

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Dashboard Performance

Page	Contents Changed	Page Updated	Time Used (Sec.)	
			CPU	Wall
Main	1	2	0.00	0.02
CPUs	1	2	0.00	0.02
GPU-Fixed	1	2	0.72	0.77
GPU-Chg.	1136	330	3.98	8.45
GPU-Dyn.	1	2	0.19	1.15
Memory	16	6	0.00	0.02
Disks	343	268	0.22	0.53
Files	343	268	0.13	0.47
Software	11	4	0.00	0.02
Calls	46	6	0.00	0.01
Parameters	8	5	0.00	0.03
Performance	717	330	8.70	8.44
Movie	329	329	9.24	9.76
Usage	1	2	0.01	0.02
Links	1	2	0.00	0.01
Dashboard	1136	330	0.08	0.55
Help	2	2	0.00	0.00

Dashboard Parameters

Parameter	Value
MINDB	500 pixels
MAXDB	1000 pixels
MTSIZE	500 pixels
IWATCH	115
NSUPD	5 sec.
NSREF	3 sec.
NSMOVE	2 sec.
MAXMOV	1000 frames

NOTE:

Difference between *Contents Changed* and *Page Updated*.

During the solution, it may not be necessary to redraw a page every time its contents change. The Parameter NSUPD lets the changes accumulate for NSUPD seconds before the page is redrawn. Increasing NSUPD reduces the time **Dashboard** spends rendering pages. All pages are drawn at the beginning and end of the job.

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Welcome to MatrixWarrior

A performance measuring and tuning tool for scientific computers.

This is the Quick Start Guide. For a detailed description of **MatrixWarrior**, refer to the **MatrixWarrior** section of the ***FMS*** Manual.
If you are already familiar with **MatrixWarrior**, click Start to begin.

What can MatrixWarrior do for you?

- Measure the actual performance and tune your machine(s);
- Evaluate machines and options prior to purchase;
- Tune and burn-in your machines after purchase;
- Validate new machine designs before production;
- Check possible hardware issues in production using a known application; and
- Learn about the ***FMS*** and **Dashboard** libraries, which can be incorporated into your application to provide performance benefits and useful displays of information.

<table>
<tr><td rowspan="2">Your Application</td><td>Your Application</td><td rowspan="2">MatrixWarrior</td></tr>
<tr><td>FMSlib
[A]{X}={B}</td><td>FMSlib
[A]{X}={B}</td></tr>
<tr><td>Dashboard</td><td>Dashboard</td><td>Dashboard</td></tr>
<tr><td colspan="3">Hardware</td></tr>
</table>

MatrixWarrior is an application layered on ***FMS***lib, the World's leading software library for solving large matrix algebra problems in production applications. ***FMS***lib is highly optimized for each hardware configuration, keeping all components operating at maximum capacity throughout the entire computation. This provides an ideal test environment for measuring machine performance.
Dashboard is a performance display library that generates a WEB site during program execution. **MatrixWarrior** used **Dashboard** to display information on the hardware and its performance.

What problem does MatrixWarrior use for testing your machine?

Problem Types for [A]{X}={B}

Module	Data Type	Matrix Symmetry
1	Real	Symmetric
2	Real	Nonsymmetric
3	Complex	Hermitian

MatrixWarrior uses ***FMS***lib to solve the linear system of equations **[A]{X}={B}**, where **[A]** is an NxN coefficient matrix, **{B}** is 1 or more known vectors and **{X}** are the solution vectors to be computed. **MatrixWarrior** can solve problems with the data types and matrix symmetry listed in the table.
The solution proceeds in two steps:

4	Complex	Symmetric
5	Complex	Nonsymmetric

1. Matrix Factoring, which involves only the matrix data in **[A]**
2. Vector Solution, which uses the matrix factors computed in (1) and the known vectors **{B}** to solve for the vectors **{X}**.

During the test, you may specify the size of the matrix, data type, symmetry and number of vectors. Default values are provided, based on the machine being tested.

How can I solve a really large problem?

For large problems, ***FMS***lib can store the data on disk. You control where the data is stored by creating a directory on your file system and setting the **FMSDISK** environment variable to that directory name. **MatrixWarrior** will create the directory and set FMSDISK for you if none is specified. For more advanced use of disks, ***FMS***lib contains a complete file striping system which is described here.
NOTE: On a slow machine with a large disk, the default problem size can take a long time.

How can I control what reports are generated and displayed?

Dashboard is a software display package that monitors the performance of hardware and software. It creates and updates a website with separate pages for hardware, software and application performance. Hardware pages include information about the CPUs, GPUs, Memory and Disks. Software pages include information on the file system, installed software, a subroutine call-tree and usage statistics. A Performance page summarizes the progress of the application and the performance of key hardware components. These pages can be viewed with any standard WEB browser, including a PC, tablet or phone.
MatrixWarrior uses **Dashboard** to display information about the hardware it is using and its performance. When you start running MatrixWarrior, you will be asked if you want to open a WEB browser to view the results. When you answer yes, a second window pops up. This is your WEB browser. It begins by displaying this Help page.

Controlling Report Generation

Parameter	Default	Description
IWATCH	99	Which reports are generated A value of 115 will generate a movie.
NSUPD	5 sec.	How frequently **Dashboard** updates the pages
NSREF	3 sec.	How frequently the browser reloads the pages

You can control what reports **Dashboard** generates and how they are displayed with the Parameters listed in the table. IWATCH determines what reports get generated, NSUPD determines how often the WEB pages are updated and NSREF determines how often the WEB browser looks for updates.
The Performance page summarizes the results of the test, displaying a variety of information on the hardware performance. As new data becomes available, this page is updated. As an alternative, you can save the intermediate pages to create a *movie* of the performance history by setting IWATCH=115.
.

How can I select what hardware is used?

Adjusting Hardware

At startup, **Dashboard** determines the installed hardware and displays the information on the CPUs, GPU1, GPU2,

Parameter	Default	Description
MAXCPU	All	Number of CPU cores
MAXGPU	All	Number of GPUs
MEMPCT	50	Percent of memory

Memory and Disks pages. You may use less than the installed hardware by specifying Parameters. These scalar values can be specified in either of two places:

- at the **FMSSET>** prompt during problem input, or
- in file *mwlic.txt* between the **FMSSET** and **RETURN** lines

The most popular hardware Parameters to change are listed in the table. A complete list of all Parameters can be found in the Parameter section of the ***FMS*** manual..

Let's start

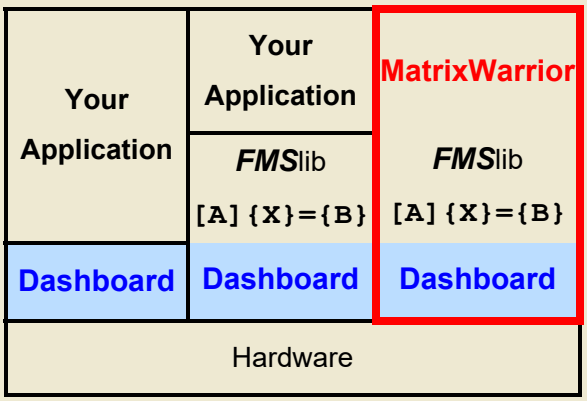
Ready to run **MatrixWarrior** ? If this is your first time, try using the default values suggested in the [*default value*] brackets.
Click Start to begin.
Please Email any questions, comments or suggestions to MatrixWarrior@fmslib.com.

8. MATRIXWARRIOR APPLICATION

What is **MatrixWarrior**?

MatrixWarrior is an application that requires no programming. It can be used to:

- Measure the actual performance and tune your machine(s);
- Evaluate machines and options prior to purchase;
- Tune and burn-in your machines after purchase;
- Validate new machine designs before production;
- Check possible hardware issues in production using a known application; and
- Learn about the ***FMS*** and **Dashboard** libraries, which can be incorporated into your application to provide performance benefits and useful displays of information.



MatrixWarrior is layered on FMSlib, the World's leading software library for solving large matrix algebra problems in production applications. ***FMS***lib is highly optimized for each hardware configuration, keeping all components operating at maximum capacity throughout the entire computation. This provides an ideal test environment for measuring machine performance.

FMSlib uses **Dashboard** to display information on the hardware and application performance. Because **Dashboard** is incorporated into ***FMS***lib, it is also part of the **MatrixWarrior** application. The chapter on Dashboard describes the reports that are generated and how you can select the information displayed.

FMSlib and **Dashboard** are libraries, which have been fully integrated into **MatrixWarrior**. All the features discussed in this manual for these libraries are available when running **MatrixWarrior**.

Where to get **MatrixWarrior**:

MatrixWarrior is available as a download from Multipath's website at www.fmslib.com.
MatrixWarrior is an application distributed as a single file, with all appropriate components of ***FMS***lib and **Dashboard** included (statically linked in). Where optional libraries are required, they are dynamically loaded at runtime. If a library is missing, **MatrixWarrior** will run without using the features provided by the optional library. The same version of **MatrixWarrior** will run on systems with and without GPUs. If all required GPU hardware and

software is detected at startup, GPUs will automatically be used (plug-and-play).

Which version of **MatrixWarrior** do I want:

Versions are available for Windows and Linux. Where appropriate, separate versions are provided for different CPU processor capabilities or NVIDIA software toolkit releases. As new hardware and software technology becomes available, it is incorporated into ***FMS***lib. However, this technology may not be compatible with earlier hardware and software. Generally speaking, older versions of **MatrixWarrior** will run on newer machines but newer versions may not run on older machines.

How to run **MatrixWarrior**

Quick Start Option - Run MatrixWarrior

When you start **MatrixWarrior**, a help page is displayed on your WEB browser which contains brief instructions on running the program. You can preview it here. This is an excellent introduction to **MatrixWarrior**, ***FMS***lib and **Dashboard**.

Detailed instructions:

When **MatrixWarrior** starts, it performs the following tasks:

- ***FMS***lib determines what hardware and software is available. A summary of this information is printed on the console window you used to start the application.
- **Dashboard** writes a unique cascading stylesheet file named `dbyyyymmddhhmm.css`. The unique date in the name assures that the correct stylesheet gets loaded for the WEB pages **Dashboard** will generate.
- The first time **MatrixWarrior** is run, a license file `mwlic.txt` is generated. This file allows you to change the machine configuration on subsequent runs by adding ***FMS*** Parameters between the following lines:

```
FMSSET
(added FMS Parameter)
RETURN
```

- The directory where **Dashboard** will generate the WEB pages is listed as follows:

`WEB pages will be created in the directory` ***Your directory***

By default, ***Your directory*** will be your current working directory. You can change where the pages will be created by defining the environment variable FMSHTML. This may be a directory on the machine you are using or on a network drive accessible by other machines. If the directory is on a machine with a WEB server, any WEB viewing device, including laptops, tablets or phones can be used to view the **Dashboard** output.

After this initialization phase, **MatrixWarrior** will ask you a series of questions. How you answer these questions will determine the problem that is run and the hardware that is used.
Each question ends with a default value enclosed in square brackets [*default value*]. To select the default value, type the *Return* or *Enter* key. If you do not want to accept the default value, you may enter a value of your choice. If **MatrixWarrior** determines that the value entered is invalid, the question is repeated.

When running **MatrixWarrior** as a batch job with an input data file that has a fixed sequence of answers, reasking a question will cause an input data file to become out-of-sync with the program. **MatrixWarrior** provides an option to stop the program with an error message if an invalid input value is encountered.

Following is the list of questions **MatrixWarrior** will ask and your choices for response.

1. MatrixWarrior Question (Batch or interactive job):

```
Is this a batch job(y or n)[N]?>
```

Your Response:

If you are running **MatrixWarrior** as a batch job enter Y. This will skip Input 2 and prevent reasking questions if an incorrect answer found to a question. The default value of N will provide the option to open your WEB browser and view the results as they are generated.

2. MatrixWarrior Question (WEB page creation):

```
Do you want to open your browser to view them(y or n)[Y]?>
```

Your Response:

If you want to view the WEB pages on your local machine, select the default value [Y]. A new window will appear on your monitor. This is your WEB browser that will display the output of **Dashboard**. If you are running **MatrixWarrior** for a second time and your browser is already opened to a **MatrixWarrior** page, you can answer [N] and then enter Control-r to refresh your display.

3. MatrixWarrior Question (Matrix type):

```
Enter the type of matrix you want to solve (1 to 5)
0 = Exit
1 = Real Symmetric
2 = Real Nonsymmetric
3 = Complex Hermitian
4 = Complex Symmetric
5 = Complex Nonsymmetric
```

```
(0:5)[5]>
```

Your Response:

You select the matrix symmetry and data type by entering the module number, 1 to 5. The default value of 5 will solve a nonsymmetric matrix with complex data.

If you just want to test the initialization phase, you may enter 0 to exit.

Problem Types for [A]{X}={B}

Module	Data Type	Matrix Symmetry	Storage Space	Compute Time
1	Real	Symmetric	X=1	X=1
2	Real	Nonsymmetric	2X	2X
3	Complex	Hermitian	2X	4X
4	Complex	Symmetric	2X	4X
5	Complex	Nonsymmetric	4X	8X

MatrixWarrior uses ***FMS***lib to solve the linear system of equations **`[A]{X}={B}`**, where **`[A]`** is an NxN coefficient matrix, **`{B}`** is 1 or more known vectors and **`{X}`** are the solution vectors to be computed.

MatrixWarrior can solve problems with the data types and matrix symmetry listed in the table

The amount of storage and computing required depend on the symmetry of the matrix and the data type. The table lists the storage and compute requirements, relative to a real symmetric matrix. Nonsymmetric matrices require twice the storage and processing time as Symmetric and Hermitian matrices. Problems with complex data type require twice the storage for the (real, imaginary) data pairs and four times the processing time than problems with real data.

FMS performs all computation with double-precision (64-bit) arithmetic. For complex data, a pair of (64-bit) values are used for the real and imaginary parts.

4. MatrixWarrior Question (Storage location):

```
Do you want to use disks(y or n)[N]?>
```

Your Response:

If you answer [N], by default, **MatrixWarrior** stores the data in memory. If you answer [Y], the data will be stored on disk.

A major benefit of ***FMS***lib is the ability to store the data on disk. This allows the solution of significantly larger problems. For **MatrixWarrior**, this provides an opportunity to test your file system.

To use this feature you should create a directory on your file system where you want to store the data. You then set the environment variable FMSDISK to that directory name. During the solution, ***FMS***lib will create files in the

FMSDISK directory for storing the matrix data. At the end of **MatrixWarrior**, these files will be deleted.

FMSlib also contains a file striping system which allows multiple disks to be operated in parallel. This can provide a higher transfer rate and more capacity. A description of this feature is contained in the section on File Striping.

If you answer [Y] and have not created a directory and defined FMSDISK, **MatrixWarrior** will create directory `C:/FMS/FMSDISK` for you and store the data there.

5. MatrixWarrior Question (Problem size):

`Enter the size of the matrix[`***value***`]>`

Your Response:

Based on the problem type you selected in (2), **MatrixWarrior** computes the maximum problem size that that will fit in the storage you selected in (3). This computed maximum problem size becomes the default [***value***]. You may select this or enter a smaller value.
CAUTION: On a slow machine with a large disk, the maximum problem size can take a long time. When using disks, a value lower than the maximum problem size is usually specified.

6. MatrixWarrior Question (Solve for {X}):

`Enter the number of solution vectors (0:`***value***`)[0]>`

Your Response:

The solution proceeds in two steps:

- Matrix Factoring, which involves only the matrix data in **`[A]`**, and
- Vector Solution, which uses the matrix factors and the known vectors **`{B}`** to solve for the vectors **`{X}`**.

By default, a value of [0] is selected, which skips the vector solution process. The most numerically intensive part of the computation is matrix factoring, which provides the best test for most **MatrixWarrior** runs.
If you do not select the default of [0], **MatrixWarrior** computes the maximum number of solution vectors which will fit in the remaining storage and lists this in the [***value***] part of the question. You may select this value or enter a smaller value. When using disks, the same caution applies: selecting the maximum value may result in a long solution time.

If you plan to perform this solution phase, you should select a matrix size in step (4) that is less than the maximum so storage will be available for the solution vectors.

At the end of the test, **MatrixWarrior** computes the solution error by comparing the computed values in **`{X}`** with the known solution for this test problem data.

7. MatrixWarrior Question (*FMS* Parameters):

```
You may now alter any FMS Parameter.
When you are finished, type the letters RETURN
FMSSET>
```

Your Response:

Here you can make additional selections on what problem is solved and what hardware is used. ***FMS*** contains an extensive list of Parameters which provide special matrix algebra features or alter the hardware that is used. The Using FMS Parameters section provides an introduction and the appendix FMS Parameters provides a complete list.

Adjusting Hardware

Parameter	Default	Description
MAXCPU	All	Number of CPU cores
MAXGPU	All	Number of GPUs
MEMPCT	50	Percent of memory

At startup, ***FMS***lib computes default values, based on the hardware available. You can choose to use less than the maximum physically installed hardware by specifying a Parameter. The most popular hardware Parameters are listed in the table. For example if your system has 32 processing cores and you choose to test 16, you would specify `MAXCPU=16` After you have entered all the Parameters, you enter `RETURN` to proceed with running the problem.

8. MatrixWarrior runs the problem.

9. MatrixWarrior Question (End the test?):

```
Do you want another solution(y or n)[N]?>
```

Your Response:

If you want to run another problem, enter `[Y]`. You will then return to question (2) where you may specify another problem to solve. If your testing is complete, select the default value of [N].

Notes on **Dashboard**

MatrixWarrior includes **Dashboard**, which displays the performance as WEB pages. You can control what reports **Dashboard** generates and how they are displayed with the Parameters listed in the table.

Controlling Report Generation

IWATCH determines what reports get generated, NSUPD determines how often the WEB pages are

Parameter	Default	Description
IWATCH	99	Which reports are generated A value of 115 will generate a movie.
NSUPD	5 sec.	How frequently **Dashboard** updates the pages
NSREF	3 sec.	How frequently the browser reloads the pages

updated and NSREF determines how often the WEB browser looks for updates.

The Performance page summarizes the results of the test, displaying a variety of information on the status of the run and how the hardware is performing. As new data becomes available, this page is updated. As an alternative, you can set `IWATCH=115` to generate a ***movie*** of the Performance pages during the solution.

While **MatrixWarrior** is normally run with **Dashboard** providing WEB displays, this is not a requirement. Setting `IWATCH=0` in the license file will skip all calls to **Dashboard**.

The chapter on Dashboard contains information on installing **Dashboard** in your application, how to select what reports are generated and how they are displayed.

Copyright © Multipath Corporation

A. FMS THEORY

This appendix contains a mathematical description of the matrix algebra functions performed by ***FMS***.

Introduction

Simultaneous Equations

The solution of the following system of simultaneous equations is one of the most important calculations in scientific analysis:

$A_{11}X_1 + A_{12}X_2 + \ldots + A_{1n}X_n = B_1$

$A_{21}X_1 + A_{22}X_2 + \ldots + A_{2n}X_n = B_2$

...

...

$A_{n1}X_1 + A_{n2}X_2 + \ldots + A_{nn}X_n = B_n$

The calculation required is to determine the values of N unknowns X(N) from the array of known coefficients A(N,N) and vector B(N). Because the time required to solve this system increases as the cube of the number of equations N, this calculation often dominates analysis programs, resulting in limitations being placed on the analysis resolution, N.

In discussing techniques used to solve simultaneous equations, it is convenient to use the following matrix notation:

`[A]{X} = {B}`

where

$$\mathbf{[A]} = \begin{bmatrix} A_{11} & A_{12} & \ldots & A_{1n} \\ A_{21} & A_{22} & \ldots & A_{2n} \\ & \ldots & & \\ & \ldots & & \\ A_{n1} & A_{n2} & \ldots & A_{nn} \end{bmatrix} \quad \mathbf{\{X\}} = \begin{bmatrix} X_1 \\ X_2 \\ \ldots \\ \ldots \\ X_n \end{bmatrix} \quad \mathbf{\{B\}} = \begin{bmatrix} B_1 \\ B_2 \\ \ldots \\ \ldots \\ B_n \end{bmatrix}$$

The symbol **`[A]`** denotes the N-by-N matrix of known coefficients. A single term in the matrix **`[A]`** is referred to as A_{ij}, where the first subscript i is the row and the second subscript j is the column. The set of N known values in **`{B}`** is called the ***right-hand side*** vector while the set of N unknown values represented by **`{X}`** is called the ***solution*** vector.

Systems of simultaneous equations are often used to solve boundary value problems in the fields of structural mechanics, thermal analysis, fluid mechanics, diffusion, and electromagnetics. For these applications, the vector

{X} usually represents unknown values of the field variable at sample points distributed throughout the domain of the problem. The vector **{B}** contains the known boundary condition values at the sample points. The matrix **[A]** serves as a mathematical model of the domain, relating the known boundary conditions **{B}** to the unknown field variable values **{X}**. The number of sample points, N, generally determines the resolution of the analysis.

As an example, suppose a structural analysis is performed. The matrix **[A]**, known as the stiffness matrix, serves as a mathematical model of the object being analyzed. The vector **{B}** contains loads that are applied to the object **[A]**, resulting in the displacements **{X}**.

Several techniques are currently used for generating simultaneous equation models of boundary value problems. Among the most popular are finite elements, finite difference, boundary integrals and the moment method. Simultaneous equations can also be generated directly from a model of discrete elements, as occurs in power network and electrical circuit analysis.

Related Systems of Equations

Often several related systems of equations that all have the same coefficient matrix **[A]** must be solved. For example, each vector **{B}** may represent a different set of boundary conditions that are applied to the same object **[A]**. If the systems are independent, where the values of each **{B}** do not depend on the previous solutions **{X}**, then all systems may be solved at the same time.

FMS includes special features for parallel processing of multiple right-hand side and solution vectors that provide extremely fast processing of related systems of equations. Therefore, the symbols **{X}** and **{B}** are used throughout this manual to mean one or more vectors.

Linear and Nonlinear Systems of Equations

Systems of simultaneous equations can be linear or nonlinear. For linear problems, the matrix coefficients A_{ij} are constant, while for nonlinear problems they depend on the solution values **{X}**. Nonlinear problems are usually solved as a sequence of linear problems, with the solution values **{X}** used to update the matrix coefficients A_{ij} at each stage of the analysis. ***FMS*** contains several features that minimize the amount of computation performed when only some of the matrix coefficients A_{ij} are changed.

Matrix Symmetry and Data Type

The matrix **[A]** can be symmetric, Hermitian or nonsymmetric. The corresponding values in the upper and lower triangle of a symmetric matrix are equal, $A_{ij} = A_{ji}$. For a Hermitian matrix, the upper triangle terms are the complex conjugate of the lower triangle terms and the diagonal terms are real.

Many boundary value problems produce symmetric matrices because the influence of one point i on another point j is the same as the influence of point j on point i. The property of symmetry is used in the equation solving algorithm to eliminate almost half of the matrix data storage and computation. The more general case of nonsymmetrical matrices, which provides for arbitrary coefficients A_{ij}, is also included.

The data type can be real or complex. Real data requires only a single floating-point value to represent each term, while complex data requires both the real and imaginary parts. In addition to requiring twice as much storage, complex data requires four times as much processing because cross-products between the real and imaginary parts must be evaluated. For example, it requires one multiply(*) and one add (+) to compute the following for real data:

A = B + C * D

However, for complex data, the calculation becomes the following:

(A+ai) = (B+bi) + (C+Ci)*(D+di)

Uppercase letters denote the real part of the data, lowercase letters denote the imaginary part of the data, and i is the square root of (-1). The real and imaginary parts of the data are computed as follows:

```
A = B + C*D - c*d
a = b + C*d + c*D
```

This computation requires four multiplies(*) and four additions (+ or -).

To optimize the use of memory pipelines, complex data is stored as the real part followed by the imaginary part. This storage is the default for variables typed COMPLEX in FORTRAN programs.

Profile Matrix Storage

FMS uses profile sparsity storage to account for large areas of the matrix **[A]** that may be zero. The original matrix is stored by lower triangular part **[AL]**, diagonal **[D]**, and upper triangular part **[AU]** as follows:

[A] = [AL] + [D] + [AU]

[A] is stored by rows, starting at the first nonzero term and proceeding across the row up to, but not including, the diagonal. Similarly, **[AU]** is stored by columns, starting at the first nonzero term and proceeding down the column to, but not including, the diagonal. The diagonal is stored as a separate vector.



A profile vector **{LOWEQ}** is used, which completely describes the profile sparsity of the matrix. This vector contains the lowest coupled equation for each equation. For the lower triangle matrix **[AL]**, this is the column number of the first nonzero term in each row. For the upper triangle matrix **[AU]**, this is the row number of the first nonzero term in each column. The number of off-diagonal terms in row I of **[AL]** or column I of **[AU]** is I - LOWEQ(I). For a full matrix, all terms in the vector **{LOWEQ}** would be 1.

Nonsymmetric Matrices

For nonsymmetric matrices, ***FMS*** uses a single Profile Vector **{LOWEQ}** to describe the profiles of both **[AL]** and

[AU]. This assumption of symmetric sparsity results in a considerably more efficient data handling algorithm than if separate profile vectors were used for **[AL]** and **[AU]**. Physically, symmetric sparsity implies that if point i couples to point j, then point j also couples to point i. The values of the compling coefficients A_{ij} and A_{ji} may be equal or unequal to produce a symmetric or nonsymmetric matrix. Naturally, zero matrix coefficients may be used if the profile is not symmetric.

FMS profile storage includes full matrices, which always start at row and column 1, and banded matrices, which start at a constant distance from the diagonal. As discussed during the factoring process, the matrix factors may always be overlaid on the profile storage described above.

Symmetric Matrices

For symmetric matrices **[A]**, the upper triangular matrix **[AU]** is the transpose of the lower triangular matrix **[AL]**, **[AU] = [AL]**^{T}. Therefore, **[AU]** is not required or used for symmetric matrix problems. Note that the ***FMS*** symmetric storage, which is by rows in **[AL]**, is equivalent to storage by columns in **[AU]**. The ***FMS*** storage can be viewed as a generalization of the skyline format storage frequently used for symmetric matrices.

Advantages of Profile Storage

Profile storage offers both mathematical and computational advantages over other methods of storing the matrix **[A]**.

Mathematical Advantages

First, the matrix factors may be overlaid on the original matrix during the factoring process, resulting in efficient use of memory and disk space. Second, each matrix factor term is computed from the corresponding term in the original matrix by subtracting an inner-product sum. Therefore, the matrix terms are transformed one-by-one from initial matrix values to final factored values without storing intermediate results. In addition to being computationally efficient, this transformation provides inherent restart capability. The ability to modify part of the matrix and restart the factoring process is a valuable property that eliminates a significant amount of processing for nonlinear problems.

Computational Advantages

In addition to mathematical advantages of profile storage, there are also computational advantages for modern computers. The inner-products required to compute the matrix factor terms are computed using incremental addressing when the matrix is stored in ***FMS*** profile format. High-speed scientific computers operate the arithmetic units at a faster speed than the memory access time. To provide data at the faster rate, memory is divided into banks (interleaved). As long as memory addresses are generated sequentially, data references rotate among the memory banks to provide a data stream equal to the arithmetic processing speed. Random access of data, which results from indirect addressing or a step size other than unity, results in the arithmetic units waiting for data from memory. The profile storage described above results in the best possible performance for modern scientific

computers.

Vector Inner-Products

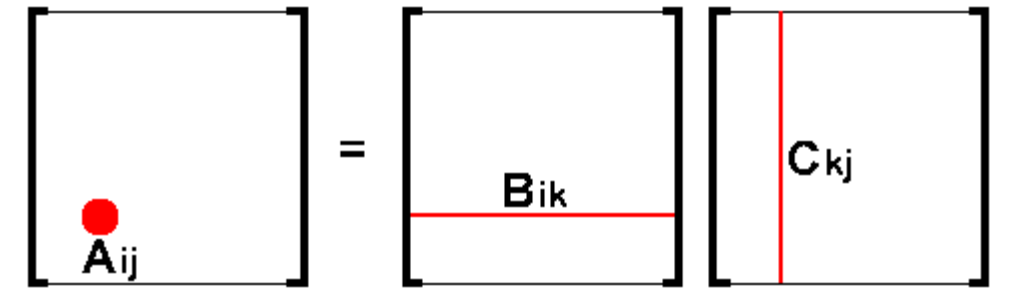
The ***FMS*** package is based on the efficient implementation of the vector inner-product. This operation requires two vectors as input and produces a scalar result. The inner-product is also called a dot product or scalar product.
In FORTRAN, the vector inner-product is implemented by a function as follows:

```
      DOUBLE PRECISION FUNCTION DOT (A, B, N)
      REAL*8 A(N), B(N)
      DOT = 0.0D0
      DO I=1,N
         DOT = DOT + A(I) * B(I)
      END DO
      RETURN
      END
```

This calculation, which uses unit address increments for the vectors **{A}** and **{B}**, achieves maximum memory performance. Several modern processors contain a multiply-accumulate instruction which implements the above loop in hardware.

Factoring

Before deriving expressions for factoring the matrix **[A]**, it is useful to examine the matrix multiplication process. Let **[A]** be the product of two full matrices **[B]** and **[C]**, as follows:



[A] = [B][C]

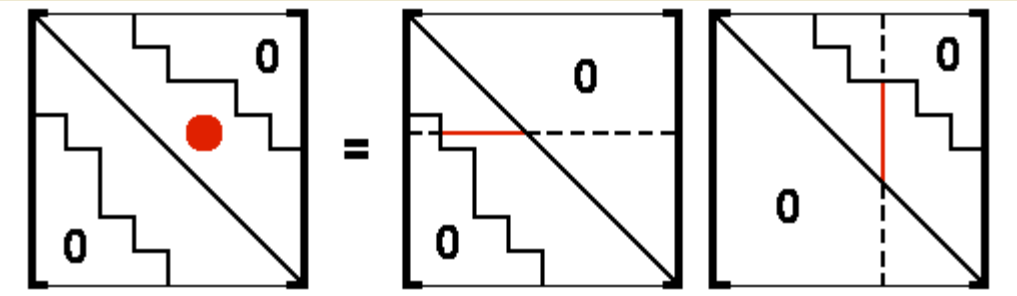
A typical term of the matrix **[A]**, A_{ij} is computed from the inner-product of the i-th row of matrix **[B]** and the j-th column of matrix **[C]**,

$$A_{ij} = \sum_{k=1}^{N} B_{ik} C_{kj}$$

If the first matrix **[B]** is stored by rows and the second matrix **[C]** is stored by columns, the resulting inner-product uses incremental memory addressing that may be pipelined.

Nonsymmetric Factoring

The nonsymmetric factoring process consists of computing the lower and upper triangular matrices whose product equals the original matrix **[A]**, as follows:



[A] = [L][U]

To obtain an expression for computing a typical term U_{ij} in **[U]**, consider its corresponding term A_{ij} in **[A]**. Based on the matrix multiplication process described above, the term A_{ij} is computed from the inner-product of the i-th row of **[L]** and the j-th column of **[U]**,

$$A_{ij} = \sum_{k=1}^{N} L_{ik}\, U_{kj}$$

This inner-product can be divided into the following four parts:

$$A_{ij} = \sum_{k=1}^{(IJLOW-1)} L_{ik}\, U_{kj} + \sum_{k=IJLOW}^{(i-1)} L_{ik}\, U_{kj} + L_{ii}U_{ij} + \sum_{k=(i+1)}^{N} L_{ik}\, U_{kj}$$

In this expression, the term IJLOW is equal to the maximum of ILOW and JLOW, where ILOW = LOWEQ(I is the column number of the first nonzero term in i-th row of **[L]**, and JLOW = LOWEQ(J) is the row number of the first nonzero term in column j-th column of **[U]**.

The value of the first part of the inner-product is zero, because either or both of L_{ik} and U_{kj} will be outside the matrix profile. For this reason profile sparsity is used on large matrices that are to be factored. The terms L_{ik} and U_{kj} for k=1,(IJLOW-1) do not contribute to the matrix factors and remain zero during the factoring process.

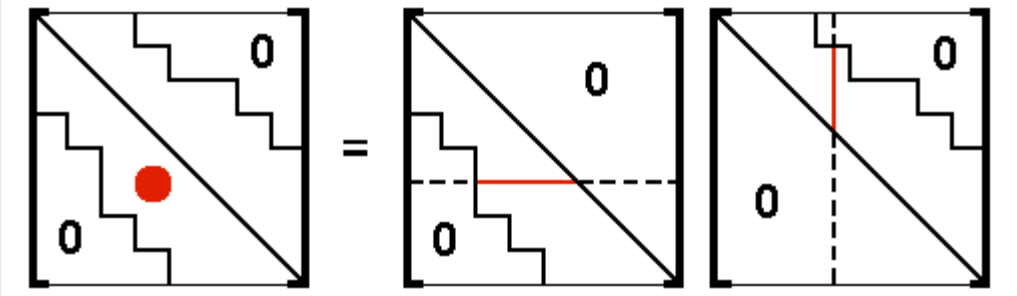
The last part of the inner-product involving the sum from (i+1) to N will also be zero because L_{ik} is zero for terms in the upper triangle.

In computing the factors **[L]** and **[U]**, there are N(N+1) terms to be determined from the N^2 terms in **[A]**. Therefore, N terms of **[L]** or **[U]** may be chosen arbitrarily. The choice that results in the most efficient algorithm is to select the diagonal terms of **[L]** to be 1. Using $L_{ii} = 1$, the above equation may be solved for U_{ij} as follows:

$$\sum_{k=1}^{(i-1)}$$

$$U_{ij} = A_{ij} - \sum_{k=IJLOW} L_{ik} U_{kj}$$

Terms in the Lower Triangular Matrix



[A] = [L][U]

Expressions for computing terms in the lower triangular factor **[L]** can be derived in an identical way. Consider a term A_{ij} that is below the diagonal,

$$A_{ij} = \sum_{k=1}^{N} L_{ik} U_{kj}$$

This inner-product can be divided into the following four parts:

$$A_{ij} = \sum_{k=1}^{(IJLOW-1)} L_{ik} U_{kj} + \sum_{k=IJLOW}^{(j-1)} L_{ik} U_{kj} + L_{ij}U_{jj} + \sum_{k=(j+1)}^{N} L_{ik} U_{kj}$$

As before, the term IJLOW is equal to the maximum of ILOW and JLOW where ILOW = LOQEQ(I) is the column number of the first nonzero term in row I and JLOW = LOWEQ(J) is the row number of the first nonzero term in column J.

The values of the first and fourth parts of the inner-product are zero as before because they involve terms outside the matrix profile or terms in the lower triangle of **[U]**. Solving the equation for the term L_{ij} gives the following:

$$L_{ij} = (A_{ij} - \sum_{k=IJLOW}^{(j-1)} L_{ik} U_{kj})/U_{jj}$$

Therefore, terms in the lower triangular factor **[L]** are also computed by subtracting an inner-product from the corresponding term in the original matrix **[A]**. However, the result is then multiplied by the diagonal factor reciprocal $1/U_{jj}$.

During the factoring and vector solution processes, the diagonals of **[U]** are only used as denominators. Because multiplication is faster than division, the terms U_{ii} are inverted immediately after being computed and the reciprocal values 1/U are overlaid on the original matrix diagonals A in the diagonal vector.

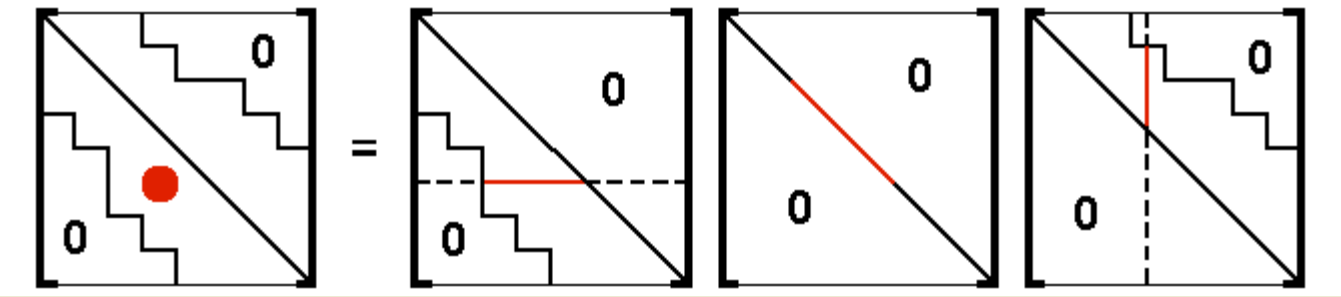
ii ii

Term-by-term Computation

The expressions for U_{ij} and L_{ij} compute the matrix factors **[U]** and **[L]** term-by-term, replacing each corresponding term of A_{ij} as the calculation proceeds. In general, the calculations must progress from the upper left corner of the matrix A_{11} and end at the lower right corner A_{nn}. The only restriction is that the terms required to compute each U_{ij} or L_{ij} be known from previous calculations. The actual ***FMS*** order is based on maximizing the reuse of high-speed registers once they have been loaded with data.

Symmetric Factoring

The symmetric factoring process consists of computing lower triangular and diagonal matrices whose product **[L] [D] [L]**^{T} equals the original symmetric matrix **[A]**, as follows:



[A] = [L] [D] [L]^{T}

$$A_{ij} = \sum_{k=1}^{N} L_{ik}\, D_{kk}\, L_{jk}$$

In this expression, the intermediate result $L_{ik}\, D_{kk}$ is computed, which is represented by the symbol $(LD)_{ik}$, as follows:

$(LD)_{ik} = L_{ik}\, D_{kk}$

$$A_{ij} = \sum_{k=1}^{N} (LD)_{ik}\, L_{jk}$$

This inner-product can be divided into the following four parts:

$$A_{ij} = \sum_{k=1}^{(IJLOW-1)} (LD)_{ik}\, L_{jk} + \sum_{k=IJLOW}^{(j-1)} (LD)_{ik}\, L_{jk} + (LD)_{ij} L_{jj} + \sum_{k=(j+1)}^{N} (LD)_{ik}\, L_{jk}$$

The term IJLOW is equal to the maximum of ILOW and JLOW, where ILOW = LOWEQ(I) and JLOW = LOWEQ(J)

are the profile equation numbers for rows I and J of **[L]**.

The value of the first part of the inner-product is zero because $(LD)_{ik}$ or L_{jk} is outside the matrix profile. The last part of the inner-product is zero because terms in the upper triangle of **[L]** are zero. Noting that $L_{jj} = 1$, the above equation can be solved to give the following:

$$(LD)_{ij} = A_{ij} - \sum_{k=IJLOW}^{(j-1)} (LD)_{ik} L_{jk}$$

Therefore, the first step in computing the matrix factor term L_{ij} is to subtract an inner-product from the corresponding term in the original matrix, A_{ij}. The inner-product is between all (LD) terms on row i to the left of L_{ij} and the corresponding terms in row j of **[L]**. As the calculation proceeds across row i, the original matrix terms are transformed from A_{ij} to $(LD)_{ij}$.

The final values of L_{ij} for row i are obtained by multiplying the terms $(LD)_{ik}$ by the diagonal factor reciprocals $1/D_{kk}$.

$L_{ik} = (LD)_{ik} (1/D_{kk})$

Because the diagonal factors **[D]** are only used as denominators, they are inverted and stored in the diagonal vector as reciprocals.

Before completing the calculation for L_{ij} the value in $(LD)_{ij}$ must be used to compute the diagonal factor term D_{ii}. Letting j = i in the equation for (LD) is as follows:

$$(LD)_{ii} = A_{ii} - \sum_{k=IJLOW}^{(i-1)} (LD)_{ik} L_{ik}$$

Noting that:

$(LD)_{ii} = L_{ii} D_{ii}$

and that $L_{ii} = 1$, the equation for D_{ii} becomes the following:

$$D_{ii} = A_{ii} - \sum_{k=IJLOW}^{(i-1)} (LD)_{ik} L_{ik}$$

The computation of the diagonal for row i involves an inner-product between the intermediate result for row i, $(LD)_{ik}$, and the final result for row i, L_{ik}. Therefore, the conversion of $(LD)_{ik}$ and the calculation of D_{ii} proceed together as follows:

```
      DOT = 0.0D0
      DO K = ILOW,(I-1)
         LDSAVE = LD(I,K)
```

```
            L(I,K) = DINV(K) * LD(I,K)
            DOT    = DOT + LDSAVE * L(I,K)
         END DO
         D(I,I) = A(I,I) - DOT
         DINV(I) = 1.0D0/D(I,I)
```

When computing **[L]** and **[D]**, the order of computation must begin at the upper left corner of the matrix A_{11} and end at the lower right A_{nn}. The only restriction is that terms required to compute each L_{ij} and D_{ii} be known from previous calculations. Unless additional storage is to be used, the calculation of the diagonal factor D_{ii} and the multiplication of the i-th row by the diagonal factor reciprocals must occur together, as discussed above. The actual ***FMS*** order is based on maximizing the reuse of high-speed registers once they have been loaded with data.

Vector Solution

Once the matrix **[A]** is factored, the following system of equations can be solved for the unknowns **{X}**:

$[L][D][L]^T\{X\} = \{B\}$ (symmetric)

or

[L][U]{X} = {B} (nonsymmetric)

The first step is to solve the intermediate problem for the vector **{Y}** as follows:

[L]{Y} = {B}

For the data structure and algorithms in ***FMS***, this is exactly the same for symmetric and nonsymmetric problems. In fact, if a symmetric matrix is processed with the nonsymmetric module, the matrix factor **[L]** is identical to the matrix factor **[L]** that would have been obtained by using the symmetric module.

An expression for computing a typical term in vector **{Y}**, Y_i, may be obtained from the vector calculation for B_i

$$B_i = \sum_{k=1}^{N} L_{ik} Y_k$$

This inner-product can be divided into the following four parts:

$$B_i = \sum_{k=1}^{(ILOW-1)} L_{ik} Y_k + \sum_{k=ILOW}^{(i-1)} L_{ik} Y_k + L_{ii} Y_i + \sum_{k=(i+1)}^{N} L_{ik} Y_k$$

ILOW = LOWEQ(I) is the column number of the first nonzero term in row i of **[L]**. The value of the first part of the inner-product is zero because L_{ik} is outside the matrix profile. The last part is also zero because L_{ik} is zero for terms in the upper triangle. Recalling that $L_{ii} = 1$, the expression for Y_i is obtained as follows:

$$Y_i = B_i - \sum_{k=ILOW}^{(i-1)} L_{ik} Y_k$$

Forward Reduction

As the calculation proceeds in the forward direction of increasing i, the terms in Y_k are known when they are required to compute Y_i. For this reason, this calculation is usually called forward reduction and the vector **{Y}** is called the reduced vector. As each term Y_i is computed, it can be overlaid on the right-hand side vector term B_i. ***FMS*** stores the matrix **[L]** by rows and the terms Y_k as a vector. Therefore the inner-product used for forward reduction produces incremental addressing that can be pipelined.

Diagonal Scaling

For symmetric matrices, the second step of vector solution is diagonal scaling. This operation computes the intermediate vector **{Z}** from the reduced vector **{Y}** as follows:

[D]{Z} = {Y}

Recalling that the diagonal factor **[D]** was stored in reciprocal form, this calculation becomes the following:

$Z_i = (1/D_{ii})\ Y_i$

As each multiplication is performed, the values of the scaled vector Z_i are overlaid on the reduced vector Y_i.

Back Substitution in Nonsymmetric Problems

The final step in vector solution is back substitution. For nonsymmetric problems, this step involves solving the following equation for the solution **{X}**:

[U]{X} = {Y}

To obtain an expression for computing **{X}**, consider the vector calculation to compute Y_i, as follows:

$$Y_i = \sum_{k=1}^{N} U_{ik} X_k$$

This inner-product can be divided into the following three parts:

$$Y_i = \sum_{k=1}^{(i-1)} U_{ik} X_k \quad + U_{ii}X_i \quad + \sum_{k=(i+1)}^{N} U_{ik} X_k$$

The first part is zero because it involves terms in the lower triangle of **[U]**. Letting

$(XU) = X\ U$

i i ii

the above equation can be solved for the following:

$$(XU)_i = Y_i - \sum_{k=(i+1)}^{N} U_{ik} X_k$$

and

$X_i = (XU)_i (1/U_{ii})$

As the calculations proceed in a backwards direction of decreasing i, the terms in X_k are known when they are required to compute X_i. For this reason, this calculation is usually called back substitution.

The inner-product

$$\sum_{k=(i+1)}^{N} U_{ik} X_k$$

is not evaluated directly because it involves nonunity address increments in **[U]**. Instead partial sums are computed and stored after each X_i is obtained. Using the same storage locations for **{Y}**, **{XU}**, and **{X}**, the actual back substitution algorithm becomes the following:

$X_i = (XU)_i (1/U_{ii})$

$$(XU)_k = (XU)_k - \sum_{k=ILOW}^{(i-1)} U_{ki} X_i$$

In the previous equation, ILOW = LOWEQ(I). This calculational order provides incremental addressing of terms in **[U]** and processes only those terms that are within the matrix profile. After each term X_i is computed, it scales the i-th column of **[U]**, U_{ki}. The resulting vector terms are subtracted from the terms in **{XU}** below X_i.

Back Substitution in Symmetric Problems

The symmetric back substitution process is almost identical, except $\mathbf{[L]}^{\mathbf{T}}$ is used for **[U]** with $L_{ii} = 1$. A typical term X_i is obtained from the scaled vector Z_i by subtracting the inner-product as follows:

$$X_i = Z_i - \sum_{k=(i+1)}^{N} L_{ki} X_k$$

This inner-product also involves nonunity addressing in **[L]**. Therefore partial sums are stored after each term X_i is computed. The actual algorithm uses the same storage for **{X}** and **{Z}** and computes the following:

$$X_k = X_k - \sum_{k=ILOW}^{(i-1)} L_{ik} X_i$$

In this equation ILOW = LOWEQ(I). As each term X_i is computed, it scales the i-th row of **[L]**. The resulting vector terms are subtracted from the values of **{X}** below X_i.

Matrix and Vector Multiplication

FMS includes multiply operations that occur frequently in matrix algebra. The first operation computes the following:

{C} = {D} - [A]{B}

{C} = [A]{B}

{C} = {D} + [A]{B}

In these operations, **[A]** is a matrix stored in ***FMS*** matrix format and **{B}**, **{C}** and **{D}** are one or more vectors stored in ***FMS*** vector format. One application of these multiply operations is to compute the residual error after solving a system of equations,

{E} = {B} - [A]{X}

The resulting error can then be used in interative refinement to compute a correction to the solution **{X}**.

A similar set of multiply routines are provided which use submatrix files instead of the assembled matrix:

{C} = {D} - [Si]{B}

{C} = [Si]{B}

{C} = {D} + [Si]{B}

This calculation may be more efficient if the assembled matrix **[A]** is sparse.

FMS also contains the following multiplication routines:

{C} = {D} - {A}{B}

{C} = {A}{B}

{C} = {D} + {A}{B}

{C} = {D} - {A}^{T}{B}

{C} = {A}^{T}{B}

{C} = {D} + {A}^{T}{B}

where **{A}**, **{B}**, **{C}** and **{D}** are ***FMS*** vector files. The above products may also be stored in ***FMS*** matrix files using

[C] = [D] - {A}^{T}{B}

[C] = [D] {A}^{T}{B}

[C] = [D] + {A}^{T}{B}

FMS can also store the vector products in a memory-based array to compute

[F] = {X}^{T}{Y}

where **[F]** is a full matrix and **{X}** and **{Y}** are ***FMS*** vector files containing one or more vectors. For symmetric problems, only the lower triangle of **[F]** is computed.

These multiply operations have several practical applications in computing Eigenvalues of the matrix **[A]**. For example, if **{X}** is a known Eigenvector, the corresponding Eigenvalue can be computed from the following:

e = **({X}^{T}[A]{X})/({X}^{T}{X})**

This calculation uses the matrix-vectors multiply operation once and the vectors-vectors multiply operation twice.

These multiply operations are also used to project the matrix **[A]** into a subspace **[A*]** where its Eigenvalues can be computed more efficiently. This projection involves the following computation:

[A*] = {X}^{T}[A]{X}

where **{X}** are the vectors that define the subspace.

In the special case where the matrix is diagonal **[D]**, ***FMS*** directly computes the following quadratic form:

[F] = {X}^{T}[D]{X} (symmetric)

[F] = {X}^{T}[D]{Y} (nonsymmetric)

FMS also includes the vectors matrix multiply calculation

{Y} = {X}[F]

This calculation may be used to project subspace eigenvectors back into the problem space or to compute new vectors **{Y}** from a linear combination of old vectors **{X}**.

Copyright © Multipath Corporation

B. FMS FILES

FMS stores data in ***Vector***, ***Submatrix*** and ***Matrix*** Files. A ***File Attribute List*** is used to store critical information for each file type, including unit numbers, record sizes, number of records and data type. The File Attribute Lists are passed to ***FMS*** subroutines to perform operations on the stored data.

The following table contains links to descriptions of each file type and their File Attribute Lists:

FMS Files

File Type	File Attribute List
Vector Files	Vector File Attribute List
Submatrix Files	Submatrix File Attribute List
Matrix Files	Matrix File Attribute List

Copyright © Multipath Corporation

B.1. FMS VECTOR FILES

FMS uses vector files to store right-hand side and solution vectors when solving simultaneous equations. ***FMS*** also used vector files to store **{X}** and **{Y}** vectors for multiply operations.

When you call FMSOV or FMSOV2 to create a vector file, ***FMS*** completes the Vector File Attribute List LUX(25), which you provide. This array contains all the information ***FMS*** needs to manage the vector file.

Each vector is stored as a record on file LUX(1). The record length is computed by ***FMS*** and stored in LUX(4). You should always use this record length when addressing the file to store multiple vectors.

Copyright © Multipath Corporation

B.1.1. VECTOR FILE ATTRIBUTE LIST

The Vector File Attribute List contains all information ***FMS*** needs to manage the files used for storing vectors. You provide an empty array to subroutine FMSOV or FMSOV2 when the file is opened and ***FMS*** fills in the necessary information. You then provide this array to each ***FMS*** subroutine you call to perform operations on the vector.

LUX(1) contains the file number and LUX(4) the record length. You will need this information when you read and write data to the file.

The following table lists the information stored in the Vector File Attribute List.

Vector File Attribute List

Word	Name	Description
1	LUX	***FMS*** file number.
2	NONE	Reserved for future use.
3	NUMEQ	Number of equations in the vector.
4	LENREC	Length of the vector in words. This is the record length of file LUX in 8-byte words. This may be rounded up from IDTYPE*NUMEQ.
5	IDTYPE	Data type (1=real, 2=complex).
6	NUMVEC	Number of vectors. This is the number of records on file LUX.
7	IVSTAT	Vector status. • 0, Right-hand side vectors. • 1, Reduced vectors. For symmetric problems, the vectors also have been scaled by the diagonal factor reciprocals. • 2, Solution vectors. This parameter catches possible errors when vector files are specified during the factoring and solution process. When the file is opened, IVSTAT is initialized to 0. After forward reduction and diagonal scaling, IVSTAT is set to 1. At the end of vector solution, IVSTAT is set to 2. If a nonlinear analysis is being performed,and right-hand side vectors

		are written to the file more than once, IVSTAT must be reset to 0 as follows: `LUX(7) = 0.`
8	IQSV	Queue/Stripe/Volume number.
9	IOPEN	Address space. • 0, In ***FMS*** managed memory or disk, • 1, In the application. This file was created by calling FMSOV2.
10 - 25	NONE	Reserved for future use.

Copyright © Multipath Corporation

B.2. FMS SUBMATRIX FILES

FMS uses submatrix files for finite element programs. Each file record is designed to hold the information for one element.

When you call FMSOS to create a submatrix file, ***FMS*** completes the Submatrix File Attribute List LUS(25), which you provide. This array contains all the information ***FMS*** needs to manage the submatrix file.

FMS uses three files to store the data for each submatrix:

- Real data file LUS(1)

  This file stores the submatrix data **[S]**.

- Integer data file LUS(2)

  This file stores the following:
  - M, the size of the submatrix
  - ITYPE, how the data is stored on the real data file LUS(1)
  - **{IEQSUB}**, the global equation vector that relates the rows and columns of the submatrix to rows and columns of the global matrix.

- Vector data file LUS(8) (Optional)

  This file stores the submatrix vector data V(M,NUMVEC).

There is one record on each file for each submatrix.

Submatrix Real File, LUS(1)

Each record of the submatrix real file LUS(1) contains a submatrix stored according to one of the following formats:



SUBMATRIX FORMAT TYPE 1

Symmetric or Nonsymmetric;

Full matrix stored by columns;

Equation numbers may be in a random order, including repeated equation numbers.

This format is standard FORTRAN format for the submatrix S(M,M).

SUBMATRIX FORMAT TYPE 2

Symmetric or Nonsymmetric;



Full matrix stored by rows;
Equation numbers can be in a random order, including repeated equation numbers.
This format is included for compatibility with existing programs.

SUBMATRIX FORMAT TYPES 3 and 4



Symmetric only;
Lower triangular stored by rows starting with column 1 and proceeding through the diagonal.
For format type 3, equation numbers must be in numerical order with no repeated equation numbers. This format provides the fastest assembly for symmetric matrices.
For format type 4, equation numbers can be in a random order, including repeated equation numbers that may result from degenerate finite elements.



SUBMATRIX FORMAT TYPE 5

Symmetric only
Row matrix.
The equations can be in any order including repeated equation numbers, except the last equation number must be the highest and correspond to the diagonal term. This format is typically used to include constraint equations with Lagrange multipliers.

Submatrix Integer File, LUS(2)

Each record of the submatrix integer file LUS(2) contains two integers, followed by the global equation numbers as follows:

Submatrix Integer File Format

Word	Name	Description
1	M	Number of equations in the submatrix.
2	ITYPE	Submatrix format type (1 to 5).
3:m+2	**{IEQSUB}**	Integer array of length M which contains the global equation numbers relating rows and columns of the submatrix to rows and columns of the global matrix. The submatrix term `S(I,J)` contributes to the global matrix term `A(IEQSUB(I),IEQSUB(J))`. If ITYPE=3, this list and the associated submatrix data must be in numerical order and cannot contain repeated numbers. If a zero

		value is specified for an entry in IEQSUB, the corresponding row and column of the submatrix are skipped and not added to the global matrix.
M+3:LENI	FILL	Optional fill to a constant record length of LENI = `LUS(4)`.

A single equation vector IEQSUB(M) relates both rows and columns of the submatrix to rows and columns of the global matrix. This relationship implies that the diagonals of the submatrix lie on the diagonals of the global matrix. This interface is used for finite element programs. User supplied subroutines should be used if a more general interface is required.

The element submatrix **[S]** and vector **{V}** are assembled into the global matrix **[A]** and global vector **{B}** according to the equation number vector **{IEQSUB}** supplied on each record. The actual assembly process for a full matrix and right-hand side vector is equivalent to the following FORTRAN statements:

```
         DO I=1, M
            IGLOB = IEQSUB(I)
            IF(IGLOB .GT. 0) THEN
C              Add in matrix term:
               DO J=1, M
                  JGLOB = IEQSUB(J)
                  IF (JGLOB .GT. 0) THEN
                     A(IGLOB,JGLOB) = A(IGLOB, JGLOB) + S(I,J)
                  END IF
               END DO
C              Add in vector term:
               DO J=1, NUMVEC
                  B(IGLOB,J) = B(IGLOB,J) + V(I,J)
               ENDDO
            END IF
         END DO
```

If you are using wavefront numbering, create a single submatrix file. The order that you write the records to the file will determine the wavefront numbering scheme. When you write the integer file `LUS(2)`, provide the equation nicknames in the equation vector IEQSUB(M). The ***FMS*** subroutine FMSWF will rewrite this file for you, replacing your nicknames with ***FMS*** equation numbers.

If you are not using wavefront numbering, supply the submatrix equation numbers directly in the `IEQSUB(M)` array. If possible, you should write the submatrices in the order of increasing lowest equation number. That is, all submatrices that contribute to equation 1 are written first, followed by all remaining submatrices that contribute to equation 2, etc. This record order results in the minimum amount of data transfers during the assembly process.

In order to achieve the above record order, you may want to use multiple submatrix files. For example, you may

want to place each element type on its own file. When using multiple submatrix files, you must store the file attributes in a single array `LUS(25,NUMSF)`.

Submatrix Vector File, LUS(8)

You may also provide a submatrix vector file `LUS(8)` which contains contributions to the right-hand side vectors. Each record on the file contains the array `V(M,NUMVEC)` for the submatrix.

Record Format

The record format used for the files is one of the following:

- Fixed Length Record

  You can create fixed length record files by specifying the record length for LENR, LENI and LENV when you call FMSOS. Use this file type if all records are approximately the same size. Note that you can use multiple files if necessary. If you do use multiple files, their file attributes must be stored together in the array `LUS(25,NUMSF)`.

- Variable Length Record

  If you specify negative or zero values for LENR, LENI and LENV when you call FMSOS, ***FMS*** creates variable length record files. Use this option if you are writing a single file for wavefront numbering or your records have a variety of lengths. You can help ***FMS*** organize these files by providing an estimate of the average record length. This estimate is specified as a negative number for LENR, LENI and LENV.

Copyright © Multipath Corporation

B.2.1. SUBMATRIX FILE ATTRIBUTE LIST

The Submatrix File Attribute List contains all information ***FMS*** needs to manage the files used for storing submatrices. You provide an empty array to subroutine FMSOS that opens the submatrix file an ***FMS*** fills in the necessary information. You then provide this array to each ***FMS*** subroutine you call to perform operations on the submatrices.

The Submatrix File Attribute List contains the files and record lengths you will need to write data to the Real, Integer and Vector submatrix files. You should never change the information stored in this array.

The following table lists the information stored in the Submatrix File Attribute List.

Submatrix File Attribute List

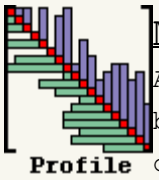
Word	Name	Description
1	LUSR	***FMS*** file number for the matrix file.
2	LUSI	***FMS*** file number for the integer file.
3	LENR	Matrix file record length.
4	LENI	Integer file record length.
5	NUMREC	Number of records.
6	NONE	Reserved for future use.
7	IRFMAT	Record format (0=fixed length, 1=variable).
8	LUSV	***FMS*** file number for the vector file.
9	NUMV	Number of vectors on LUSV.
10	LENV	Vector file record length.
11 - 25	NONE	Reserved for future use.

Copyright © Multipath Corporation

B.3. FMS MATRIX FILES

FMS offers three options for storing and processing the matrix data. The ***FMS*** Parameter MFMAT is used to select which option you want.

The options are:



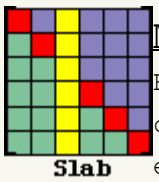
MFMAT=1: Profile Solver

Accounts for the sparsity of matrix **[A]** on an equation by equation basis. Typical applications include finite element and finite difference programs. This is the default for sparse matrices.



MFMAT=2: Block Solver

Divides the matrix **[A]** into square blocks, accounting for sparsity on a block by block basis. This solver uses industry standard BLAS3 kernels and provides excellent I/O performance on extremely large problems. Typical applications include boundary integral and moment method programs. This is the default for full matrices.



MFMAT=3: Slab Solver

Extends the functionality of the Block Solver by providing full column partial pivoting for full nonsymmetric matrices. You must explicitly set MFMAT to select this option.

When you create a matrix file, ***FMS*** completes the Matrix File Attribute List , LUA(25), which you provide. This array contains all the information ***FMS*** needs to manage the matrix file.

FMS uses four files to store the data for each matrix:

- ## Lower Triangle Data File, LUA(1)

  Stores the terms in the lower triangle **[AL]**.

- ## Diagonal Data File, LUA(2)

  Stores the terms on the diagonal as a single vector, **{D}**. You may read and write this data using FMSRED and FMSWRT.

- ## Upper Triangle Data File, LUA(3)

  Stores the terms in the upper triangle **[AU]**. This file is not used for symmetric matrices.

- Integer Data File, LUA(4)

  Stores all integer pointers necessary to access the file.

The data storage in files LUA(1), LUA(3) and LUA(4) is problem and machine dependent. You should never try and write these files directly.

Copyright © Multipath Corporation

B.3.1. MATRIX FILE ATTRIBUTE LIST

The Matrix File Attribute List contains all information ***FMS*** needs to manage the files used for storing a matrix. You provide an empty array to the ***FMS*** subroutine that opens the matrix file and ***FMS*** fills in the necessary information. You then provide this array to each ***FMS*** subroutine you call to perform operations on the matrix.

For most applications, you do not need to know what ***FMS*** stores in this array. If you want to read or write the diagonal vector, the file is LUA(2). If you are reusing a matrix file as part of a nonlinear analysis, you may need to reset LUA(7) after each iteration. Other than these two conditions, you should not use terms directly from this array.

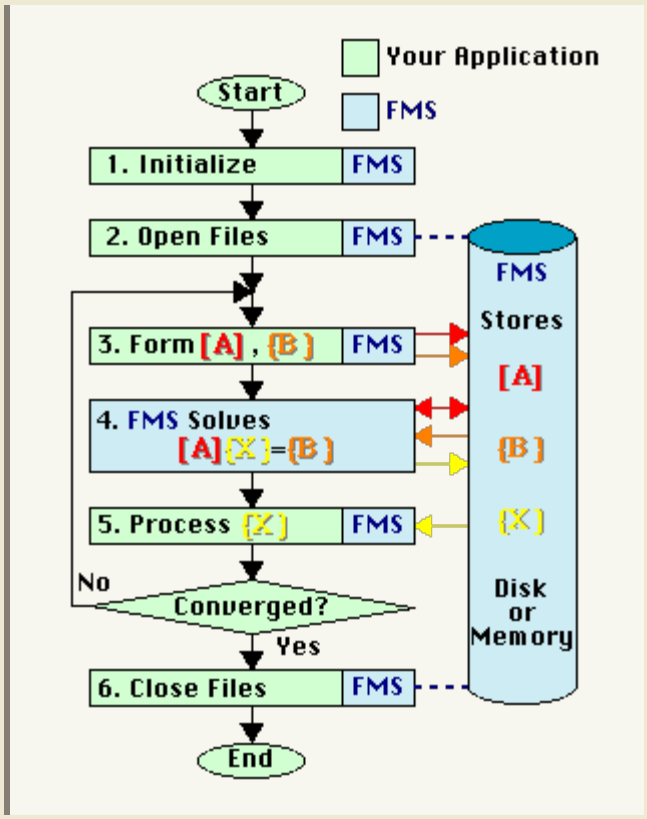
Matrix File Attribute List

Word	Name	Description
1	LUAL	***FMS*** file number for lower triangle matrix **`[AL]`**.
2	LUAD	***FMS*** file number for matrix diagonals **`[D]`**.
3	LUAU	***FMS*** file number for upper triangle matrix **`[AU]`**. This file is only used for nonsymmetric matrices. For symmetric matrices, LUAU=0.
4	LUAI	***FMS*** file number for the Segment Table.
5	LENRLU	Record length for files LUAL and LUAU.
6	NUMSEG	Number of matrix segments.
7	NEWSEG	Matrix segment status parameter. • 0, Files do not contain any matrix data. • 1 to NUMSEG, Segments 1 through (NEWSEG-1) contain factored data and segments NEWSEG through NUMSEG contain unfactored data. • NUMSEG+1, Files contain all factored data. This attribute indicates the status of the matrix data. When the file is opened, NEWSEG is set to 0 to indicate that there is no matrix data on the files. When the matrix assembly subroutine is called, NEWSEG is updated to be the matrix segment containing equation LOWASM, which is an ***FMS*** Parameter. If the assembly process begins at the first equation (LOWASM=1), the value of NEWSEG after assembly is 1, indicating the matrix files contain entirely unfactored data. However, if a nonlinear analysis is being

		performed and only the bottom part of the matrix is to be reassembled and factored, NEWSEG is set to the segment containing the first changed equation, LOWASM. The factoring process begins at segment NEWSEG and continues down the matrix to the last segment NUMSEG. At the end of the factoring process, NEWSEG is set to NUMSEG+1 to indicate that the files contain factored matrix data. During the solution the value NEWSEG is checked to be equal to NUMSEG+1 to prevent an unfactored matrix from being used for vector solution.
8	NUMEQ	Number of equations.
9	LENDIA	Record length of the diagonal file LUAD.
10	NUMRLU	Number of records on files LUAL and LUAU.
11	IDTYPE	Data type (1=real, 2=complex).
12	ISTYPE	Matrix symmetry (1=symmetric, 2=nonsymmetric, 3=Hermitian).
13	MFMAT	Matrix format (1=PROFILE, 2=BLOCK, 3=SLAB)
14	MPIVOT	Reserved for future use.
15	LENTAB	Length of the Segment Table, which is stored on file LUAI.
16	IAVGBW	Average bandwidth
17	IQSV	Queue/Stripe/Volume number
18	IOPEN	Address space. =0, In ***FMS*** managed memory or disk, =1, In the application. This file was created by calling RSDANN, RNDANN, CHDANN, CSDANN, CNDANN.
19	IROWCO	File format (0=Standard, 1=FMSROW, 2=FMSCOL, 3,4=FMSPUT)
20	IFULL	Sparsity (0=Profile, 1=Full)
21	NEQBIO	I/O block dimension NEQBIO (BLOCK and SLAB only)
22	NEQBLK	Compute block dimension NEQBLK
23	IJSTEP	Stride toward diagonal IJSTEP
24	INSIDE	Blocking pattern INSIDE
25	NONE	Reserved for future use

Copyright © Multipath Corporation

C. FMS SUBROUTINES



- STEP1: Initialize ***FMS***
- STEP2: Open ***FMS*** Files
- STEP3: Write Data to ***FMS*** Files
- STEP4: Perform Matrix Algebra
- STEP5: Read Data from ***FMS*** Files
- STEP6: Close ***FMS*** Files and End
- Memory Management Subroutines
- Printing Subroutines
- Parallel Processing Subroutines

Copyright © Multipath Corporation

C.1. STEP1: INITIALIZE FMS

Index	NAME	Function
C.1.1	FMSINI FMSIN2	Initialize ***FMS*** - First Subroutine Called
C.1.2	FMSIST FMSRST FMSCST	Set an Integer Parameter Set a Real Parameter Set a character Parameter
C.1.3	FMSIGT FMSRGT FMSCGT	Read an Integer Parameter Read a Real Parameter Read a Character Parameter
C.1.4	FMSSET FMSSETC	Set ***FMS*** Parameters interactively

Copyright © Multipath Corporation

C.1.1. SUBROUTINES FMSINI, FMSINI2

PURPOSE

Initialize ***FMS***

SYNOPSIS

```
CALL FMSINI
CALL FMSIN2 (IOPT)
```

INPUT PARAMETERS

`FMSINI`

None

`FMSINI2`

- IOPT=Integer

  Specifies initialization options as follows:

  =0, Complete initialization, same as calling FMSINI,

  =1, Reduced initialization with normal output,

  =2, Reduced initialization with no output.

OUTPUT PARAMETERS

None

Description

This must be the first ***FMS*** subroutine called and should be the first executable statement in your application. You may call this subroutine only once in your application.

These subroutines performs the following tasks:

- Check for shared object libraries required by special features, including GPUs and nonuniform memory access
- Determine the hardware configuration including CPUs, GPUs, Memory and Disks
- Determine the software environment including operating system, ***FMS*** version and other libraries
- Initialize all ***FMS*** Parameters to default values

- Read the license file and update the default values of the ***FMS*** Parameters
- Initialize the semaphores and locks used for synchronizing ***FMS*** tasks (FMSINI only)
- Start the CPU children (FMSINI only)
- Start the I/O processes (FMSINI only)
- Allocate the ***FMS*** memory pool
- Start the GPU threads (FMSINI only)
- Print file and disk information.

The reduced initialization provided by FMSIN2(1) or FMSIN2(2) does not start any ***FMS*** threads for parallel processing or performing I/O. Therefore no I/O or matrix algebra routines may be called when initializing with FMSIN2(1) or FMSIN2(2). This option is primarily used to compute the amount of disk space that will be required when ***FMS*** opens files in a later application. The ***FMS*** Parameter NOOPEN can be used with a reduced initialization to skip the physical opening of files.

Copyright © Multipath Corporation

C.1.2. SUBROUTINES FMSIST, FMSRST, FMSCST

PURPOSE

To provide an way to change ***FMS*** Parameters within your application.

SYNOPSIS

```
CALL FMSIST (NAME, VALUE)
CALL FMSRST (NAME, VALUE)
CALL FMSCST (NAME, VALUE)
```

INPUT PARAMETERS

- NAME = Character String.

  The name of the ***FMS*** Parameter

- VALUE = Integer (FMSIST)

  VALUE = Real (FMSRST)

  VALUE = Character String (FMSCST)

  Specifies the value to assign to the ***FMS*** Parameter.

OUTPUT PARAMETERS

None

Description

The `FMSIST, FMSRST,` and `FMSCST` subroutines set integer, real, and character ***FMS*** Parameters respectively. They can be called at any point in the application program after FMSINI. If the specified name is not valid, or the value is out of range, a warning message is written on file LUERR and the statement is ignored.

Examples

To turn off all ***FMS*** output, your program would contain the following statement:

```
CALL FMSIST  ('PRINT', 0)
```

To show the current contents of the ***FMS*** memory pool, your program would contain the following statement:

```
          CALL FMSCST  ('SHOW', 'MEMORY')
```

Copyright © Multipath Corporation

C.1.3. SUBROUTINES FMSIGT, FMSRGT, FMSCGT

PURPOSE

To provide an way to obtain the value of ***FMS*** Parameters within your application.

SYNOPSIS

```
CALL FMSIGT (NAME, VALUE)
CALL FMSRGT (NAME, VALUE)
CALL FMSCGT (NAME, VALUE)
```

INPUT PARAMETERS

- NAME = Character String.

  The name of the ***FMS*** Parameter

OUTPUT PARAMETERS

- VALUE = Integer (FMSIST)

  VALUE = Real (FMSRST)

  VALUE = Character String (FMSCST)

  Contains the value currently assigned to the ***FMS*** Parameter.

Description

The FMSIGT, FMSRGT, and FMSCGT subroutines obtain the value of integer, real, and character ***FMS*** Parameters, respectively. They can be called at any point in your program after FMSINI.

If the specified name is not valid, a warning message is written to file LUERR and no value is returned.

Examples

As an example, to obtain the number of pivot sign changes found during factoring, your program would contain the following statement:

```
CALL FMSIGT  ('NUMSCG', NUMSCG)
```

Copyright © Multipath Corporation

C.1.4. SUBROUTINES FMSSET, FMSSETC

PURPOSE

To provide an interactive way to change ***FMS*** Parameters.

SYNOPSIS

```
CALL FMSSET
CALL FMSSETC()
```

INPUT PARAMETERS

None

OUTPUT PARAMETERS

None

Description

The `FMSSET` and `FMSSETC` subroutines can be called at any point in your program after FMSINI to examine and change ***FMS*** Parameter. They provides a convenient tool for debugging and performance studies where frequent changes to ***FMS*** Parameters are required. If your program is written in FORTRAN, call `FMSSET`. If it is written in C, call `FMSSETC`.

In the following discussion `FMSSET` is used but implies `FMSSET` or `FMSSETC` depending on your application program language. `FMSSET` is designed to be run interactively from a terminal. Once called, FMSSET will solicit input by displaying the following prompt:

`FMSSET>`

To examine a specific ***FMS*** parameter, enter the name of the parameter as follows:

`FMSSET>PARAMETER_NAME`

To change a specific ***FMS*** parameter, enter the name and value as follows:

`FMSSET>PARAMETER_NAME=VALUE`

where the equal sign (=) serves as a delimiter between the ***FMS*** Parameter name and its value to be assigned. The SHOW parameter can be used to obtain various reports on the internals of ***FMS***, including a list of all parameter values.

```
FMSSET>SHOW
```

To return to your application and continue processing, enter the following command:

```
FMSSET>RETURN
```

Examples

As an example, you can turn all ***FMS*** output off except for displaying megaflops during factoring. Your application would be coded as follows:

```
      . . .
      CALL FMSINI
      CALL FMSSET
      . . .
```

The following commands would be entered from the terminal or command file running the job:

```
FMSSET>PRINT=0
FMSSET>IPRF=1026
FMSSET>RETURN
```

Copyright © Multipath Corporation

Copyright © Multipath Corporation

C.2. STEP2: OPEN FMS FILES

Index	NAME	Function
C.2.1	FMSOV	Open a Vector File
C.2.2	FMSOV2	Open an Incore Vector File
C.2.3	FMSOS	Open a Submatrix File
C.2.4	RSDI RNDI CHDI CSDI CNDI	Open a General Matrix
C.2.5	RSDANN RNDANN CHDANN CSDANN CNDANN	Open a Full Incore Matrix
C.2.6	FMSOM	Open a Second Matrix File
C.2.7	FMSBW	Compute Matrix Bandwidth
C.2.8	FMSPF	Compute Matrix Profile
C.2.9	FMSWF	Compute Matrix Profile from Wavefront Storage

Copyright © Multipath Corporation

C.2.1. SUBROUTINE FMSOV

PURPOSE

To open a Vector File for storing one or more vectors of equal length.

SYNOPSIS

```
CALL FMSOV (NUMEQ, IDTYPE, NUMVEC, NAME, LUX)
```

INPUT PARAMETERS

- NUMEQ = Integer.

  Number of equations (terms) in the vector

- IDTYPE = Integer.

  Data type (1=real, 2=complex)

- NUMVEC = Integer.

  Number of vectors (records) on the file.

- NAME = Character String.

  A 1 to 119 character string that contains a unique name for the vector file. ***FMS*** appends a 'V' to the file name before it is created.

OUTPUT PARAMETERS

- LUX(25) = Integer array.

  FMS Vector File Attribute List for the new created file.

  NOTE: This is an input parameter when opening an existing file.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
INCORE	File location
IEXIST	Use existing file
NOOPEN	Do not physically open the file.

DESCRIPTION:

Subroutine FMSOV is used to open a file used to supply ***FMS*** with vector data.

The ***FMS*** Parameter IEXIST determines if ***FMS*** creates a new file or uses an existing file. If you know you are using an existing file (IEXIST=1), the parameters NUMEQ, IDTYPE and NUMVEC are not used and you can specify dummy arguments. If you might use an existing file or create a new one (IEXIST=2), the values specified for NUMEQ, IDTYPE and NUMVEC must match the existing file described by LUX(25). In both cases (IEXIST > 0), you must supply the file attribute list LUX(25) that was used to initially create the file.

If you are only processing one vector, it is probably better to allocate it in memory (unless you need to save it between jobs). You can create a vector file in memory by setting the INCORE parameter before calling this subroutine or by calling FMSOV2.

The vector values can be written to the opened file by using the FMSWRT subroutine.

EXAMPLE

If the real array RHS(NUMEQ, NUMVEC) contains the vector values, they can be written to the file with the following statements:

```
CALL FMSOV (NUMEQ, 1, NUMVEC, 'RHS', LUX)
LOCD = 1
DO NV=1, NUMVEC
   CALL FMSWRT (LUX(1), LOCD, RHS(1,NV), NUMEQ)
   LOCD = LOCD + LUX(4)
END DO
```

Note that LUX(4) contains the record length for the file as computed by ***FMS***. This value may be greater than IDTYPE*NUMEQ to align records on sector boundaries. You should always use LUX(4) to address the vector file.

Copyright © Multipath Corporation

C.2.2. SUBROUTINE FMSOV2

PURPOSE

To create a ***FMS*** vector file for direct access to the array `X(NUMEQ,NUMVEC)`, which is dimensioned in your program.

SYNOPSIS

```
CALL FMSOV2 (NUMEQ, IDTYPE, NUMVEC, X, LDX, LUX)
```

INPUT PARAMETERS

- NUMEQ = Integer.
  Number of equations (terms) in the vector.
- IDTYPE = Integer.
  Data type (1=real, 2=complex)
- NUMVEC = Integer.
  Number of vectors (records) on the file.
- X(LDX,NUMVEC) = Real array (IDTYPE=1)
  X(LDX,NUMVEC) = Complex array (IDTYPE=2)
  Array in your program for storing vector data.
- LDX = Integer.
  Leading dimension of array X. Usually NUMEQ, but must be at least as large as NUMEQ.

OUTPUT PARAMETERS

- LUX(25) = Integer array.
  FMS Vector File Attribute List for the incore data stored in array X.

DESCRIPTION:

This subroutine creates a ***FMS*** vector file attribute list LUX(25) which addresses the array `X(NUMEQ,NUMVEC)` in your program. It provides an efficient and easy to use interface when the vector data remains in memory during ***FMS*** processing. You should use this subroutine whenever there are only a few vectors.
When using this subroutine, it is not necessary to create the vector file by calling FMSOV, or to read and write the

vector data. ***FMS*** performs all operations directly on the vector data contained in your array X.

The array X must be aligned on a natural address boundary.

Copyright © Multipath Corporation

C.2.3. SUBROUTINE FMSOS

PURPOSE

To open a Submatrix File for storing submatrices.

SYNOPSIS

```
CALL FMSOS (LENR, LENI, LENV, NUMVEC, NUMSUB, NAME, LUS)
```

INPUT PARAMETERS

- LENR = Integer.

Record length for matrix file in units of REAL*8 words.

>0, use fixed length records. If M is the dimension of the largest submatrix on the file, then LENR must be at least the following:

Submatrix Format	Data Type	LENR
1,2 (full)	Real	M*M
	Complex	2*M*M
3,4 (triangular)	Real	M*(M+1)/2
	Complex	M*(M+1)
5 (row)	Real	M
	Complex	2*M

<= 0, use variable length records.

- LENI = Integer.

Record length for integer file in units of integers.

>0, use fixed length records. If M is the dimension of the largest submatrix on the file, then LENI must be at least (M+2).

<= 0, use variable length record files.

- LENV = Integer.

Record length for vector file in units of REAL*8 words.

>0, use fixed length records. If M is the dimension of the largest submatrix on the file, then LENV must be at least NUMVEC*M for real data, 2*NUMVEC*M for complex data.

<= 0, use variable length record files.

- NUMVEC = Integer.

Number of vectors on vector file.

= 0, do not use vector file.

- NUMSUB = Integer.

Number of records.

- NAME = Character string.

A 1 to 119 character string that contains a unique name for the submatrix file. ***FMS*** appends an 'R' to this name for the matrix file, an 'I' for the integer file and a 'S' for the vector file.

OUTPUT PARAMETERS

- LUS(25) = Integer array.

FMS Submatrix File Attribute List for the new submatrix file.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IEXIST	Use existing file
INCORE	File location (disk or memory)
LBUFSI	Buffer size for submatrix integer records
LBUFSR	Buffer size for submatrix real records
LBUFSV	Buffer size for submatrix vector records

DESCRIPTION:

Submatrix Files pass submatrix data to the ***FMS*** assembly process. Finite element programs use submatrix files when ***FMS*** performs the global matrix assembly.

You may create multiple submatrix files by calling this subroutine more than once. The submatrix file attributes, however, must be stored in a single array LUS(25,NUMSF). On each call, i, you pass the attribute list for that submatrix file, LUS(1,i).

If you want ***FMS*** to assemble the submatrix vectors, specify the number of vectors for parameter NUMVEC. If you

specify 0 for NUMVEC, ***FMS*** skips opening files for the submatrix vectors.

Record Format

The record format used for the files is one of the following:

- Fixed Length Record

  You can create fixed length record files by specifying the record length for LENR, LENI and LENV. Use this file type if all records are approximately the same size. Note that you can use multiple files if necessary. If you do use multiple files, their file attributes must be stored together in the array LUS(25,NUMSF).

- Variable Length Record

  If you specify negative or zero values for LENR, LENI and LENV, ***FMS*** creates variable length record files. Use this option if you are writing a single file for wavefront numbering or your records have a variety of lengths. You can help ***FMS*** organize these files by providing an estimate of the average record length. This estimate is specified as a negative number for LENR, LENI and LENV.

Record Buffering

By default, all submatrix files are double buffered to increase the physical I/O size and overlap I/O with data generation. The ***FMS*** parameter LBUFSI controls the buffer size for the integer file LUS(2), LBUFSR controls the buffer size for the real file LUS(1) and LBUFSV controls the buffer size for the vector file LUS(8). You may use unbuffered files for fixed length record files by setting LBUFSI, LBUFSR and LBUFSV to 0 before calling FMSOS. You should only use unbuffered files if you are writing the submatrices in a random order.

Existing Files

If the ***FMS*** Parameter IEXIST is set to 0 (default), ***FMS*** will create new files. If the ***FMS*** Parameter IEXIST is set to 1, ***FMS*** will use existing files. When using existing files, the parameters LENR, LENI, LENV, NUMVEC, NUMSUB and NAME must match the existing file or a fatal error is generated.

Copyright © Multipath Corporation

C.2.4 SUBROUTINES RSDI, RNDI, CHDI, CSDI, CNDI

PURPOSE

To open a Matrix File for storing a matrix **[A]**.

SYNOPSIS

```
CALL RSDI (LOWEQ, NUMEQ, NAME, LUA)
CALL RNDI (LOWEQ, NUMEQ, NAME, LUA)
CALL CHDI (LOWEQ, NUMEQ, NAME, LUA)
CALL CSDI (LOWEQ, NUMEQ, NAME, LUA)
CALL CNDI (LOWEQ, NUMEQ, NAME, LUA)
```

INPUT PARAMETERS

- LOWEQ(NUMEQ) = Integer array.

Matrix profile vector that specifies the lowest coupled equation for each equation. For the lower triangular matrix **[AL]**, this is the column number of the first nonzero term in each row. For the upper triangular matrix **[AU]**, this is the row number of the first nonzero term in each column. For a full matrix you may use a shortcut and specify LOWEQ(1) = -1. If you are opening an existing file and know you will not be creating a new one (IEXIST=1), **{LOWEQ}** is not required and you may specify a dummy argument.

- NUMEQ = Integer.

Number of equations in matrix **[A]**.

- NAME = Character.

A 1 to 119 character string that contains a unique alphanumeric name for the matrix. ***FMS*** generates names for the lower triangle, diagonal, upper triangle (nonsymmetric) and segment control files by appending 'L', 'D', 'U', and 'T' to NAME.

OUTPUT PARAMETERS

- LUA(25) = Integer array.

FMS Matrix File Attribute List that describes the files used to store the matrix **[A]**. If several matrices are to be stored at the same time, a separate attribute list must be maintained for each matrix.

NOTE: For files being reopened, this is an input parameter.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IPRI	Initialization print code
MFMAT	Matrix format type (PROFILE, BLOCK, SLAB)
LENSDF	I/O record size (PROFILE format)
NEQBIO	I/O block size (BLOCK & SLAB format)
NEQBLK	Compute block size
NEQBLO	Minimum compute block size
NEQBHI	Maximum compute block size
IJSTEP	Stride toward diagonal
INSIDE	Blocking pattern (PROFILE format)
INCORE	File location (disk or memory)
IEXIST	Use existing file
NOOPEN	Do not physically open the file
LSECTR	Disk sector length (8-byte words)
IEXIST	Use existing file

DESCRIPTION:

This subroutine opens files LUA(1), LUA(2), LUA(3), and LUA(4) for storing the lower triangle, diagonal, upper triangle, and Segment Table for matrix **[A]**. The file names are generated by appending 'L', 'D', 'U', and 'T' to the actual parameter NAME. No matrix data is written to files LUA(1), LUA(2) or LUA(3) at this time.
Depending on the ***FMS*** parameter IEXIST, ***FMS*** will use an existing file or create a new one. When using existing files, you must supply the file attribute list LUA(25) that was used when the file was created.

For new files, the attribute list LUA(25) is computed by ***FMS***. This list contains all information necessary for performing data transfers to any of the four matrix files. The attribute list is supplied by you to the remaining ***FMS*** subroutines as an actual parameter to define the matrix files.

Memory resident matrix files can be created by setting the INCORE parameter prior to calling this subroutine.

The Segment Table on file LUA(4) contains all the pointers and addresses used by ***FMS*** for processing the profile matrix. This information is not altered by ***FMS*** subroutines. Therefore, unless the matrix profile contained in the vector **{LOWEQ}** is changed, this initialization subroutine should be called only once.

FMS determines the record size for files LUA(1) and LUA(3) based on the available memory, cpu and gpu hardware and matrix profile. If you require a specific size, use the LENSDF parameter for PROFILE matrices or the NEQBIO parameter for BLOCK and SLAB matrices. ***FMS*** may alter your specified value to satisfy other constraints.

The profile vector `LOWEQ(NUMEQ)` must be computed in your program prior to calling this subroutine. If the matrix is being assembled from submatrices, subroutine FMSPF or FMSWF may be used to compute `LOWEQ(NUMEQ)` automatically.

EXAMPLE

The following example illustrates how the profile vector could be computed for a matrix with constant bandwidth IBAND.

```
      SUBROUTINE PBAND (IBAND, LOWEQ, NUMEQ)
C     Profile vector for a constant band matrix:
      INTEGER IBAND, NUMEQ, LOWEQ(NUMEQ)
      DO I  =  1,NUMEQ
         LOWEQ(I)  =  MAX0(1,I-BAND+1)
      END DO
      RETURN
      END
```

Copyright © Multipath Corporation

C.2.5. SUBROUTINES RSDANN, RNDANN, CHDANN, CSDANN, CNDANN

PURPOSE

Create a ***FMS*** matrix file from a full incore matrix A(LDA,N).

SYNOPSIS

```
CALL RSDANN (A, LDA, N, LUA)
CALL RNDANN (A, LDA, N, LUA)
CALL CHDANN (A, LDA, N, LUA)
CALL CSDANN (A, LDA, N, LUA)
CALL CNDANN (A, LDA, N, LUA)
```

INPUT PARAMETERS

- A(LDA,N) = Real array (RSDANN, RNDANN).

  A(LDA,N) = Complex array (CHDANN, CSDANN, CNDANN).

  Full matrix containing your data.

- LDA = Integer.

  Leading dimension of array **A**. Usually N, but must be at least as large as N.

- N = Integer.

  Number of equations.

OUTPUT PARAMETERS

- A(LDA,N) = Real array (RSDANN, RNDANN).

  A(LDA,N) = Complex array (CHDANN, CSDANN, CNDANN).

  Matrix data moved to ***FMS*** format.

- LUA(25) = Integer array.

  FMS Matrix File Attribute List.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IPRA	Assembly print code

DESCRIPTION:

This subroutine reformats a full incore matrix A(LDA,N) into ***FMS*** format and computes the matrix file attribute list LUA(25). It provides an efficient and easy to use interface between ***FMS*** and existing application programs which are processing full incore matrices.
The matrix data in array A(LDA,N) is moved to provide contiguous vectors for ***FMS*** processing. The segment table file LUA(4) is generated incore in ***FMS*** managed memory. The matrix file attributes LUA(25) are computed to point to an incore file whose data is stored in **A**.

Because ***FMS*** moves the data in array **A**, you must not change this data after calling this subroutine. If your program performs a nonlinear analysis, which requires updating terms in **A**, the old file should be closed by calling FMSCM and a new file created by calling this subroutine after the array **A** is regenerated.

The array **A** must be aligned on a natural boundary.

Copyright © Multipath Corporation

C.2.6. SUBROUTINE FMSOM

PURPOSE

To open a second matrix file for storing a matrix.

SYNOPSIS

```
CALL FMSOM (LUA, NAME, LUF)
```

INPUT PARAMETERS

- LUA(25) = Integer array.
  Matrix File Attribute List for the existing file that was opened by RSDI, RNDI, CHDI, CSDI, CNDI.
- NAME = Character string.
  A 1 to 119 character string that contains a unique alphanumeric name for the new matrix file. ***FMS*** generates names for the lower triangular, diagonal, upper triangular and Segment Table files by appending 'L', 'D', 'U' and 'T' to NAME.

OUTPUT PARAMETERS

- LUF(25) = Integer array.
  FMS Matrix File Attribute List for the new created files.

DESCRIPTION:

FMS matrix factoring subroutines are designed to overlay the matrix factors on the original matrix to conserve disk space. However, for some applications, it may be necessary to preserve the input matrix. For these applications, subroutine FMSOM should be called to create the files for storing the factored matrix before calling the ***FMS*** factoring subroutine.
Subroutine FMSOM allocates file space and computes the attribute list LUF. The Segment Table contained on the matrix file LUA(4) is copied to the matrix file LUF(4). The matrix data contained on files LUA(1), LUA(2), and LUA(3) is not copied to the output files. This matrix data is written during the factoring process.

The file location of LUF is the same as LUA. If LUA is incore, LUF will also be incore. If LUA is on disk, LUF will be on disk.

Copyright © Multipath Corporation

C.2.7. SUBROUTINE FMSBW

PURPOSE

To compute the maximum, average and effective factoring bandwidths for use in estimating matrix storage and processing time.

SYNOPSIS

```
CALL FMSBW (LOWEQ, NUMEQ, BWMAX, BWAVG, BWEFF)
```

INPUT PARAMETERS

- LOWEQ(NUMEQ) = Integer array.
  Matrix profile vector that specifies the lowest coupled equation for each equation. For the lower triangular matrix **[AL]**, this is the column number of the first non-zero term in each row. For the upper triangular matrix **[AU]**, this is the row number of the first non-zero term in each column.
- NUMEQ = Integer.
  Number of equations in matrix **[A]**.

OUTPUT PARAMETERS

- BWMAX = Real.
  Maximum bandwidth, computed from

  ```
  MAX0(I - LOWEQ(I) + 1)
  ```

- BWAVG = Real.
  Average bandwidth, computed from

  ```
  SUM(I - LOWEQ(I) + 1)/NUMEQ
  ```

- BWEFF = Real.
  Effective factoring bandwidth computed from the actual operation count of the factoring process.

DESCRIPTION:

This subroutine computes bandwidths that statistically describe the profile sparsity of the matrix. These bandwidths may be used to estimate the amount of processing time and storage requirements for the matrix.

Copyright © Multipath Corporation

C.2.8. SUBROUTINE FMSPF

PURPOSE

To compute the profile vector from submatrix integer files and optionally impose constraints.

SYNOPSIS

```
CALL FMSPF (LUS, NUMSF, NEWNUM, MAXEQ, LOWEQ, NUMEQ)
```

INPUT PARAMETERS

- LUS(25,NUMSF) = Integer array.

  File attributes for each submatrix file.

- NUMSF = Integer.

  Number of submatrix files.

- NEWNUM(NUMEQ) = Integer array.

  Flags constrained equations as follows:

  = -1, constrained to a nonzero value,

  = 0, constrained to a zero value,

  = 1, value to be determined.

  The equation numbers 1 through MAXEQ must correspond to the equation numbers supplied on the submatrix integer files `LUS(2,i)`.

- MAXEQ = Integer.

  Maximum number of equations.

  = 0, compute profile vector only. Do not renumber equations to eliminate constraints.

OUTPUT PARAMETERS

- NEWNUM(MAXEQ) = Integer array.

  Gives the new equation number I from the old equation number J and flags equation constraint type as follows:

  `I = NEWNUM(J)`

  I < 0, equation I is constrained to a nonzero value.

  I = 0, equation I is constrained to zero,

  I > 0, equation I is to be determined.

- LOWEQ(NUMEQ) = Integer array.

Profile vector based on the new numbering system.

- NUMEQ = Integer.

New number of equations.

DESCRIPTION:

This subroutine performs the following tasks:

1. Eliminates equations whose solution value is constrained to zero.
2. Computes the vector NEWNUM(MAXEQ) which relates old equation numbers to new equation numbers.
3. Updates the submatrix integer files LUS(2,i) to the new equation numbers.
4. Computes the profile vector LOWEQ(NUMEQ) and new number of equations, NUMEQ.
5. The renumbering of equations, steps 1 through 3, may be skipped by specifying a value of 0 for MAXEQ.

This subroutine is intended for finite element programs solving mixed boundary value problems where some of the solution values are constrained to specified values. You first create the submatrix integer and real files using equation numbers that include both free and constrained equations. The NEWNUM(MAXEQ) vector is then created to flag the constrained equations. This subroutine is then called to renumber the equations into a packed global system which eliminates equations whose value is constrained to zero.

Your program must pack the right-hand side vectors to correspond to the new numbering system in NEWNUM(MAXEQ). In addition, nonzero constraints must be imposed, usually by scaling the diagonal and right-hand side by a big number, BIGNUM. After the solution is obtained, the solution may be expanded back to the original numbering system using NEWNUM(MAXEQ).

Copyright © Multipath Corporation

C.2.9. SUBROUTINE FMSWF

PURPOSE

To generate a set of global equation numbers based on the wavefront order and compute the profile vector.

SYNOPSIS

```
CALL FMSWF (LUS, NEWNUM, MAXEQ, LOWEQ, NUMEQ)
```

INPUT PARAMETERS

- LUS(25) = Integer array.

Submatrix File Attributes List for the submatrix file.

- NEWNUM(NUMEQ) = Integer array.

Flags constrained equations as follows:

= -1, constrained to a non zero value,

= 0, constrained to a zero value,

= 1, value to be determined.

The equation numbers 1 through MAXEQ must correspond to the equation numbers supplied on the submatrix integer file `LUS(2)`.

- MAXEQ = Integer.

Maximum number of equations.

OUTPUT PARAMETERS

- NEWNUM(MAXEQ) = Integer array.

Gives the new equation number I from the old equation number J and flags equation constraint type as follows:

`I = NEWNUM(J)`

I < 0, equation I is constrained to a nonzero value.

I = 0, equation I is constrained to zero,

I > 0, equation I is to be determined.

- LOWEQ(NUMEQ) = Integer array.

Profile vector based on the new numbering system.

- NUMEQ = Integer.

New number of equations.

DESCRIPTION:

This subroutine performs the following tasks:

1. Renumbers the equations in wavefront order, as determined by the order of the submatrices on file LUS.
2. Eliminates equations whose solution value is constrained to zero or equations not used.
3. Computes the vector NEWNUM(MAXEQ) which relates old equation numbers to new equation numbers.
4. Updates the submatrix integer file LUS(2) to the new equation numbers.
5. Computes the profile vector LOWEQ(NUMEQ) and new number of equations, NUMEQ.

This subroutine is intended for finite element programs designed to use a "wavefront" equation solver. Your program first creates submatrix integer and real files whose record order is the order in which elements will be added to the wavefront. The equation numbers on the integer file must be unique labels which describe the connectivity between elements but are not necessarily the global equation numbers used by ***FMS*** during solution.

Your program then creates the NEWNUM(MAXEQ) array which flags constrained equations and equation numbers not used. This subroutine is then called to generate a packed global system of equation numbers which eliminates unused equations and those whose solution value is constrained to zero. This subroutine numbers the equations in the order that they may be eliminated during solution, which results in the same number of arithmetic operations used by wavefront equation solvers.

Your program must reorder the right-hand side vector according to the returned values in NEWNUM(MAXEQ). In addition, nonzero constraints must be imposed, usually by scaling the diagonal and right-hand side by a big number, BIGNUM. After the solution is obtained, the solution values may be placed back in the original numbering system using NEWNUM(MAXEQ).

Copyright © Multipath Corporation

C.3. STEP3: WRITE DATA TO FMS

Index	NAME	Function
C.3.1	FMSWRT FMSWR8 FMSSWR	Direct access write to submatrix and vector files Allows large disk addresses on 32-bit machines Sequential write to submatrix and vector files /td>
C.3.2	FMSEEK FMSEK8	Position file for sequential access
C.3.3	FMSROW FMSCOL	Write rows/columns to matrix file
C.3.4	FMSPUT	Write a block of data to a matrix or vector file
C.3.5	FMSCPY	Make a copy of an ***FMS*** file.
C.3.6	RSUBLK RNUBLK CHUBLK CSUBLK CNUBLK	Define a block of matrix data
C.3.7	RNUSLB CNUSLB	Define a slab of matrix data

Copyright © Multipath Corporation

C.3.1. SUBROUTINES FMSWRT, FMSWR8, FMSSWR

PURPOSE

To write data to a ***FMS*** file.

SYNOPSIS

```
CALL FMSWRT (LOGUNT, LOCUNT, ARRAY, NUMWRD)
CALL FMSWR8 (LOGUNT, LOCUNT, ARRAY, NUMWRD)
CALL FMSSWR (LOGUNT, ARRAY, NUMWRD)
```

INPUT PARAMETERS

- LOGUNT = Integer.

The ***FMS*** file number, which is stored by ***FMS*** in the file attribute list as follows:

  - Vector file: LUX(1)
  - Matrix file diagonal: LUA(2)
  - Submatrix: Matrix LUS(1), Integer LUS(2), Vector LUS(8)

- LOCUNT = Integer. (FMSWRT only)

LOCUNT = REAL*8. (FMSWR8 only)

The file address (in 8-byte words) where you want to start to write. The first word on the file is located at LOCUNT=1. For FMSWRT, this is an INTEGER. For FMSWR8, this is a REAL*8 Floating Point Number. FMSWR8 allows the addressing of files greater than 2 Gbytes on systems supporting only 32-bit integers.

- ARRAY(NUMWRD) = Complex, Real or Integer array.

Memory array containing the data to write.

- NUMWRD = Integer.

Number of WORDS to transfer. For complex data, count each real and imaginary pair as two words.

OUTPUT PARAMETERS

None.

DESCRIPTION:

This subroutine provides write access to ***FMS*** files.

Subroutine FMSWRT provides random access, subroutine FMSSWR provides sequential access. You may reset the file location used by FMSSWR by calling FMSEEK, FMSRED, FMSSRD or FMSWRT. When using FMSSWR to write a fixed-length record file, you must specify the record length for NUMWRD.

These subroutines are typically used to write the right-hand vectors and submatrix files prior to ***FMS*** processing. They may also be used to write the matrix diagonals if they have been modified. The off-diagonal and integer matrix files should never be written directly. Their format is machine dependent.

If you have multiple right-hand vectors to write you can use FMSPUT to write them with a single call. The larger transfer can provide improved performance.

Copyright © Multipath Corporation

C.3.2. SUBROUTINES FMSEEK, FMSEK8

PURPOSE

To position a file prior to sequential access.

SYNOPSIS

```
CALL FMSEEK (LOGUNT, LOCUNT)
CALL FMSEK8 (LOGUNT, LOCUNT)
```

INPUT PARAMETERS

- LOGUNT = Integer.

  The ***FMS*** file number, which is stored by ***FMS*** in the file attribute list.

- LOCUNT = Integer. (FMSEEK only)

  LOCUNT = REAL*8. (FMSEK8 only)

  The file address (in words) where you want to position the file. Valid addresses begin at 1 (the start of the file).

  = 0, erase file contents and set LOCUNT to 1.

  For FMSEEK, this is an integer.

  For FMSEK8, this is a REAL*8 Floating Point Number. FMSEK8 allows the addressing of files greater than 2 Gbytes on systems supporting only 32-bit integers.

OUTPUT PARAMETERS

None.

DESCRIPTION:

This subroutine positions a file prior to sequential access by FMSSWR and FMSSRD.

When a file is first opened, it is positioned at the beginning. You may use subroutine FMSSWR to write the file sequentially the first time without calling this subroutine.

If you want to read the file sequentially, you should call this subroutine with the starting file address prior to calling FMSSRD.

If you want to rewrite the file and don't care about the previous file data, you should call this subroutine with LOCUNT = 0. This is a special case which resets the file pointers to the beginning of the file and erases the file

contents.

If you must alternate read and write operations to the file, you should use the direct access subroutines FMSRED and FMSWRT.

Copyright © Multipath Corporation

C.3.3. SUBROUTINES FMSROW, FMSCOL

PURPOSE

To write data to a ***FMS*** Matrix File.

SYNOPSIS

```
CALL FMSROW (IEQN, DATA, LUA)
CALL FMSCOL (IEQN, DATA, LUA)
```

INPUT PARAMETERS

- IEQN = Integer.

  Equation number.

  For FMSROW, this is the row number.

  For FMSCOL, this is the column number.

- DATA(*) = Real or Complex array.

  This is the row data for FMSROW or the column data for FMSCOL. For symmetric matrices, only the data up to and including the diagonal is used.

- LUA(25) = Integer array.

  FMS Matrix File Attribute List for the file where the matrix is to be stored.

OUTPUT PARAMETERS

None.

FMS PARAMETERS

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
LOWASM	First changed equation for nonlinear assembly.
IOSYNC	Perform synchronlys I/O
IOKIDS	Allow children to access a file
MAXMD	Amount of memory to use

MDUSED	Amount of memory already used

DESCRIPTION:

These subroutines provide an easy and efficient way to write data to ***FMS*** matrix files. You may define the matrix terms by calling FMSROW **OR** FMSCOL. You may not mix calls to FMSROW and FMSCOL when defining terms for the same matrix.
You may define the rows (columns) in any order but you must define all matrix terms. A fatal error condition occurs if you finish calling FMSROW (FMSCOL) and have not defined all rows (columns).

To use these subroutines, you must first open the matrix file by calling subroutines RSDI, RNDI, CHDI, CSDI CNDI. You then initialize FMSROW or FMSCOL to receive data by making one of the following calls:

```
        CALL FMSROW ( 0, DUMMY, LUA),   Access by parent only
        CALL FMSROW (-1, DUMMY, LUA),   Access by parent and children
        CALL FMSCOL ( 0, DUMMY, LUA),   Access by parent only
        CALL FMSCOL (-1, DUMMY, LUA),   Access by parent and children
```

The value of 0 or -1 for the equation number directs ***FMS*** to initialize files and assign buffers. If you specify a value of 0, only the parent can define the matrix by calling FMSROW (FMSCOL). If you are going to call FMSROW (FMSCOL) from subroutines you are running in parallel, specify a value of -1. In either case, this initialization call must be made by the parent, not the children. This initialization call must precede calls to FMSROW (FMSCOL) which transfer data.
FMS will use all remaining memory in the ***FMS*** memory pool to buffer your data for file LUA. If you plan to write to multiple files simultaneously, you must divide the remaining memory in the memory pool among the files.

You may do this by using the ***FMS*** Parameters MAXMD and MDUSED. For example, if you are populating 3 matrices whose file attributes are LUA(25,3), you would divide the memory among these files as follows:

```
C       Find out how much memory is left to allocate:
        CALL FMSIGT ('MAXMD' , MAXMD )
        CALL FMSIGT ('MDUSED', MDUSED)
        MDLEFT  = MAXMD - MDUSED
C
C       Divide the memory equally among the 3 files:
        MDINC   = MDLEFT/3
C
C       Save value of MAXMD:
        MAXMD_S = MAXMD
C
C       Initialize FMSROW for first file:
```

```
        MAXMD  = MDUSED + MDINC
        CALL FMSIST ('MAXMD' , MAXMD)
        CALL FMSROW (0, DUMMY, LUA(1,1))
C
C       Initialize FMSROW for second file:
        MAXMD  = MAXMD  + MDINC
        CALL FMSIST ('MAXMD' , MAXMD)
        CALL FMSROW (0, DUMMY, LUA(1,2))
C
C       Initialize FMSROW for third file:
        CALL FMSIST ('MAXMD' ,MAXMD_S)
        CALL FMSROW (0, DUMMY, LUA(1,3))
```

You should not set MAXMD beyond the first value returned by the call to FMSIGT. Doing so will cause all memory to be returned and new memory allocated.
The second step is to call FMSROW (FMSCOL) once for each row (column). You may define the rows (columns) in any order.

The final step is to direct ***FMS*** to release the buffers and close the files. This is accomplished by making the following call:

```
        CALL FMSROW (NUMEQ+1, DUMMY, LUA)
```

or

```
        CALL FMSCOL (NUMEQ+1, DUMMY, LUA)
```

where the value of NUMEQ+1 is greater than the number of equations. This call must be made by the parent.
At this point the data has been transferred to the ***FMS*** file but is not in standard ***FMS*** file format. This conversion will happen automatically during matrix assembly/factoring when subroutine RSDAF, RNDAF, CHDAF, CSDAF ,CNDAF is called. When calling the assemble/factor subroutine, you specify this matrix file LUA as an input matrix file.

You may call FMSROW|FMSCOL from subroutines you are running in parallel (which you started by calls to FMSPAR). This can reduce the time required to fill the matrix and vector arrays. To use this feature, you must direct ***FMS*** to open the file LUA for access by multiple processes. Use the ***FMS*** Parameter IOKIDS before the call which opens file LUA. The first and last calls to FMSROW|FMSCOL, which are used for setup and completion, must be called by the parent process before and after the calls to FMSPAR respectively.

Examples

Example 8 and Example 9 further illustrates how to use FMSROW to populate a matrix.

Copyright © Multipath Corporation

C.3.4. SUBROUTINE FMSPUT

PURPOSE

To write a block of data to a ***FMS*** Matrix or Vector File

SYNOPSIS

```
CALL FMSPUT (LUFMS, NROWS, NCOLS, IROW1, JCOL1, A(NROWS,NCOLS), LDA)
```

INPUT PARAMETERS

- LUFMS(25) = Integer array.
  FMS Matrix File Attribute List or FMS Vector File Attribute List which defines the files where the data is to be stored.
- NROWS = Integer.
  The number of rows to transfer.
- NCOLS = Integer.
  The number of columns to transfer.
- IROW1 = Integer.
  The starting global row where you want to place the block.
- JCOL1 = Integer.
  The starting global column where you want to place the block.
- DATA(NROWS,NCOLS) = Real or Complex array.
  This is the array of data you want to transfer to the file, in normal FORTRAN storage.
- LDA = Integer.
  The leading dimension of the DATA array. This must be at least as big as NROWS.

OUTPUT PARAMETERS

None.

FMS PARAMETERS

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description

LOWASM	First changed equation for nonlinear assembly.
IOKIDS	Allow children to access a file

DESCRIPTION:

This subroutine provides write access to ***FMS*** Matrix and Vector files. The array DATA(NROWS,NCOLS) is transferred directly to the file LUFMS without buffering.
To use this subroutine, you must first open file LUFMS using subroutine RSDI, RNDI, CHDI, CSDI, CNDI for matrix files, or subroutine FMSOV or FMSOV2 for vector files.

You then initialize FMSPUT to receive data by making one of the following calls:

CALL FMSPUT	LUFMS contents
`CALL FMSPUT (LUFMS, 0, 0, 0, 0, 0, 0)`	Vector file Nonsymmetric matrix file Symmetric matrix defined by terms in the upper triangle.
`CALL FMSPUT (LUFMS, 0, 0, 1, 0, 0, 0)`	Symmetric matrix defined by terms in the lower triangle.

The 0 value arguments direct ***FMS*** to initialize the files to receive data. This call must precede calls to FMSPUT which transfer data.
For symmetric matrices, you may either define the terms in the upper **OR** lower triangle, but not both. If you use the first form to initialize FMSPUT with IROW1=0, FMSPUT will only transfer terms in the upper triangle and diagonal. Any term in array DATA which falls in the lower triangle is ignored. If you use the second form to initialize FMSPUT with IROW1=1, FMSPUT will only transfer terms in the lower triangle and diagonal. Any term in array DATA which falls in the upper triangle is ignored.

FMS stores data in matrix files in a direction proceeding toward the diagonal. Therefore if your DATA array is not square, one of the options for defining symmetric matrix data will be more efficient than the other. If NROWS>NCOLS, it is more efficient to populate the upper triangle. If NCOLS>NROWS, it is more efficient to populate the lower triangle.

The second step is to call FMSPUT as many times as you want to define the data. ***FMS*** does not initialize the file to zero or check if all the data on the file is defined. It is your responsibility to define all the data.

The final step is to direct ***FMS*** to release any storage. This is accomplished by making the following call:

```
CALL FMSPUT (LUFMS, 0, 0, NUMEQ+1, 0, 0, 0)
```

where the value of IROW1=NUMEQ+1 is greater than the number of equations.

At this point the data has been transferred to the ***FMS*** file. For vector files and incore matrix files, the operation is complete. For disk-based matrix files, however, the data is not standard ***FMS*** file format. This conversion will happen automatically during matrix assembly/factoring when subroutine RSDAF, RNDAF, CHDAF, CSDAF ,CNDAF is called. When calling the assemble/factor subroutine, you specify this matrix file LUFMS as an input matrix file.

You may call FMSPUT from subroutines you are running in parallel (which you started by calls to FMSPAR). This can reduce the time required to fill the matrix and vector arrays. To use this feature, you must direct ***FMS*** to open the file LUFMS for access by multiple processes. Use the ***FMS*** Parameter IOKIDS before the call which opens file LUFMS. The first and last calls to FMSPUT, which are used for setup and completion, must be called by the parent process before and after the calls to FMSPAR respectively.

Examples

Example 13 further illustrates how to use FMSPUT to populate a matrix and vectors.

Copyright © Multipath Corporation

C.3.5. SUBROUTINE FMSCPY

PURPOSE

To make a copy of a ***FMS*** file and optionally open or create the output file.

SYNOPSIS

```
CALL FMSCPY (LUA, NAME, LUB)
```

INPUT PARAMETERS

- LUA(25) = Integer array.
  Matrix, Vector or Submatrix file attribute list for the existing file.
- NAME = Character string.
  A 1 to 119 character string that contains a unique alphanumeric name for the new file. If the output file exists and is already open, this name is not used.

OUTPUT PARAMETERS

- LUB(25) = Integer array.
  Matrix, Vector or Submatrix file attribute list for the output file. This file must be the same type and have the same record structure as the input file LUA.

FMS PARAMETERS

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IEXIST	Use existing file

DESCRIPTION:

This subroutine makes a copy of the file described by LUA and places the contents into the file described by LUB. After this subroutine exits, the two files LUA and LUB are identical.
If file LUB is not opened, this routine will create and/or open it, using the name specified.

If file LUB is already opened, the contents of LUA are copied to LUB. When LUB is open, it must have the same

structure as the input file LUA. Specifically the following must match:

- File type (matrix, vector, submatrix)
- Matrix format, where appropriate (profile, block, slab)
- Record length
- Number of records
- File location (incore, disk)

If any of these conditions are not met, an error results.

Copyright © Multipath Corporation

C.3.6. SUBROUTINES RSUBLK, RNUBLK, CHUBLK, CSUBLK, CNUBLK

PURPOSE

To provide a user interface for defining or modifying matrix data.

This is a subroutine you provide.

SYNOPSIS

```
SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2, JCOL1, JCOL2,
IJSTEP)
SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2, JCOL1, JCOL2,
IJSTEP)
SUBROUTINE CHUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2, JCOL1, JCOL2,
IJSTEP)
SUBROUTINE CSUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2, JCOL1, JCOL2,
IJSTEP)
SUBROUTINE CNUBLK (A, D, LOWEQ, LOCEQ, IROW1, IROW2, JCOL1, JCOL2,
IJSTEP)
```

INPUT PARAMETERS

- LOWEQ(*) = Integer array.

  Matrix profile vector that specifies the lowest coupled equation for each equation. For the lower triangular matrix (RSUBLK, CHUBLK and CSUBLK), this is the column number of the first nonzero term in each row. For the upper triangular matrix (RNUBLK, CNUBLK), this is the row number of the first nonzero term in each column.

- LOCEQ(*) = Integer array.

  Base address for each equation (rows for RSUBLK, CHUBLK and CSUBLK; columns for RNUBLK and CNUBLK)

- IROW1 = First row which can be changed
- IROW2 = Last row which can be changed
- JROW1 = First column which can be changed
- JROW2 = Last column which can be changed
- IJSTEP = Stride toward diagonal.

OUTPUT PARAMETERS

- A(0:1) = Real array (RSUBLK and RNUBLK).
  A(0:1) = Complex array (CHUBLK, CSUBLK and CNUBLK).
  Matrix data array which is addressed by the **{LOCEQ}** vector.
- D(*) = Real array (RSUBLK, RNUBLK and CHUBLK).
  D(*) = Complex array (CSUBLK and CNUBLK).
  Matrix diagonals.

FMS PARAMETERS

The following ***FMS*** Parameters are especially important to this routine:

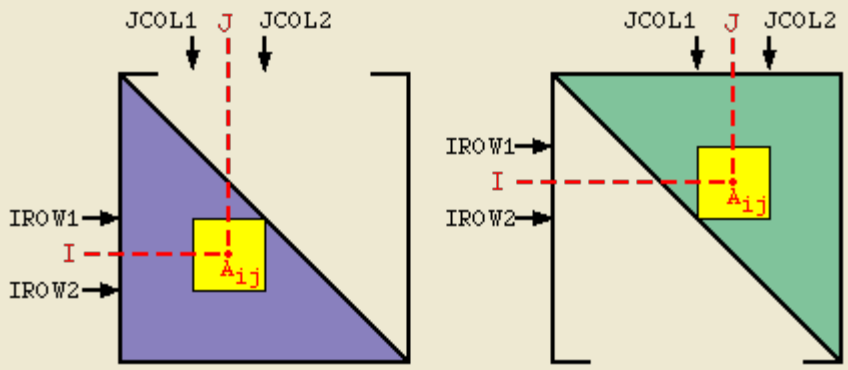
Parameter	Description
MDATAU	Call this subroutine to define or modify matrix data

DESCRIPTION:

These subroutines, which are written by you, provide an interface to the ***FMS*** global matrix file. The RSUBLK, CHUBLK and CSUBLK subroutines define terms in the lower triangle and diagonal. The RNUBLK and CNUBLK subroutines define terms in the upper triangle and diagonal. For nonsymmetric problems (RN and CN), both the lower and upper subroutines are required.
To use this feature, ***FMS*** must be directed to call your subroutines. The MDATAU parameter provides this function. The default value of 0 results in skipping all calls to these subroutines.

When MDATAU is 1, ***FMS*** calls these subroutines during the assembly process. When MDATAU is 2, ***FMS*** calls these subroutines during the factoring process. If MDATAU is 1 or 2, these subroutines are called during the assembly-factor process.

If the entire matrix does not fit in memory, these subroutines are called several times. Each call defines a window of the matrix, bound by rows IROW1 through IROW2 and columns JCOL1 through JCOL2 as shown in the figure below. The diagonal can be defined during either or both calls.



The location vector **{LOCEQ}** is used to address matrix data in the array **A**. For lower triangular terms (RSUBLK, CHUBLK, CSUBLK), the matrix coefficients are addressed as

```
A(I,J) = A(LOCEQ(I) + IJSTEP*J)
```

For the upper triangle terms (RNUBLK, CNUBLK), the matrix coefficients are addressed as

```
A(I,J) =  A(LOCEQ(J) + IJSTEP*I)
```

The row I ranges from IROW1 through IROW2 and the column J ranges from JCOL1 through JCOL2. The entire diagonal D, which is addressed as a single vector, is available during all calls.

The profile vector **{LOWEQ}** is passed to this subroutine so that you can check that no coefficients outside the matrix profile are being defined.

In general, these subroutines provide the best interface for finite difference, or general matrix formats. For finite element programs, the matrix assembly interface is more convenient.

FMS is distributed with user-supplied subroutines for populating test matrices with integer and random numbers. If your program supplies its own subroutines, they must be explicitly named on the link command to avoid using the subroutines supplied in the library.

Examples

As an example, suppose your program has a real nonsymmetric matrix C(100,100) in the common block MATRIX. The user supplied subroutines RSUBLK and RNUBLK would be as follows:

```
      SUBROUTINE RSUBLK (A, D, LOWEQ, LOCEQ,
     1  IROW1, IROW2, JCOL1, JCOL2, IJSTEP)
      COMMON/MATRIX/  C(100,100)
      INTEGER IROW1, IROW2, JCOL1, JCOL2
      INTEGER LOWEQ(IROW2), LOCEQ(IROW2)
      DOUBLE PRECISION A(0:0), D(1), C
      DO 20 IROW  =  IROW1, IROW2
         D(IROW) =  C(IROW,IROW)
         IF(IROW .LE. 1) GO TO 20
         LIEQ   =  LOCEQ(IROW)
         J2     = MIN0(JCOL2,IROW-1)
         DO 10 JCOL  =  JCOL1, J2
            A(LIEQ+IJSTEP*JCOL)  =  C(IROW,JCOL)
10       CONTINUE
20    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE RNUBLK (A, D, LOWEQ, LOCEQ,
     1  IROW1, IROW2, JCOL1, JCOL2, IJSTEP)
      COMMON/MATRIX/C(100,100)
      INTEGER IROW1, IROW2, JCOL1, JCOL2
      INTEGER LOWEQ(JCOL2), LOCEQ(JCOL2)
      DOUBLE PRECISION A(0:0), D(1), C
      DO 20 JCOL  =  JCOL1,JCOL2
         IF(JCOL .LE. 1)  GO TO 20
         LJEQ   =  LOCEQ(JCOL)
         I2     = MIN0(IROW2,JCOL-1)
         DO 10 IROW  =  IROW1, I2
            A(LJEQ+IJSTEP*IROW)  =  C(IROW,JCOL)
10       CONTINUE
20    CONTINUE
      RETURN
      END
```

Note that for this example the profile vector **{LOWEQ}** is not used. It is assumed that the matrix is full.

Copyright © Multipath Corporation

C.3.7. SUBROUTINES RNUSLB, CNUSLB

PURPOSE

To provide a user interface for defining or modifying matrix data for SLAB format matrices.

This is a subroutine you provide.

SYNOPSIS

```
SUBROUTINE RNUSLB (A, LOCI, LOCJ, LUFLAG, JCOL1, JCOL2, NUMEQ)
SUBROUTINE CNUSLB (A, LOCI, LOCJ, LUFLAG, JCOL1, JCOL2, NUMEQ)
```

INPUT PARAMETERS

- LOCI(NUMEQ) = Integer array.

Row location vector.

- LOCJ(JCOL1:JCOL2,2) = Integer array.

Column location vector.

- LUFLAG(NUMEQ) = Integer array.

LU flag vector.

=1 for rows in the upper triangle,

=2 for rows in the lower triangle.

- JCOL1 = Integer.

First column in the slab

- JCOL2 = Integer.

Last column in the slab

- NUMEQ = Integer.

Number of equations.

OUTPUT PARAMETERS

- A(0:1) = Real array (RNUSLB).

A(0:1) = Complex array (CNUSLB).

Matrix data array which is addressed by

```
A( LOCI(I) + LOCJ(J,LUFLAG(I)) ) = AIJ
```

FMS PARAMETERS

The following ***FMS*** Parameters are especially important to this routine:

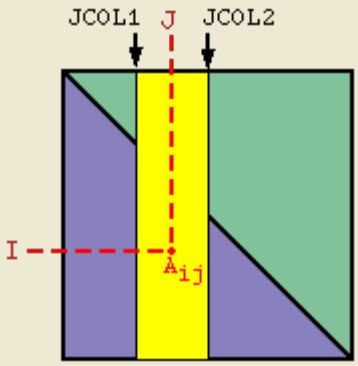
Parameter	Description
MDATAU	Call this subroutine to define or modify matrix data
MFMAT	Matrix format (PROFILE, BLOCK, SLAB)

DESCRIPTION:

These subroutines, which are written by you, provide an interface to the ***FMS*** global matrix file when the data is stored in SLAB format. The RNUSLB subroutine is used for real data and CNUSLB for complex data.
To use this feature, ***FMS*** must be directed to call your subroutines. The MDATAU parameter provides this function. The default value of 0 results in skipping all calls to these subroutines.

When MDATAU is 1, ***FMS*** calls these subroutines during the assembly process. When MDATAU is 2, ***FMS*** calls these subroutines during the factoring process. If MDATAU is 1 or 2, these subroutines are called during the assembly-factor process.

If the entire matrix does not fit in memory, these subroutines are called several times. Each call defines a slab of the matrix, bound by rows 1 through NUMEQ and columns JCOL1 through JCOL2 as shown in the figure below.



The location vectors **{LOCI}**, **{LOCJ}** and **{LUFLAG}** are used to address matrix data in the array **A** as follows:

```
A(I,J) = A(LOCI(I) + LOCJ(J,LUFLAG(I))
```

The row I ranges from 1 through NUMEQ and the column J ranges from JCOL1 through JCOL2.

FMS is distributed with user-supplied subroutines for populating test matrices with integer and random numbers. If your program supplies its own subroutines, they must be explicitly named on the link command to avoid using the subroutines supplied in the library.

Copyright © Multipath Corporation

C.4. STEP4: PERFORM MATRIX ALGEBRA

Index	NAME	Function
C.4.1	RSDAF RNDAF CHDAF CSDAF CNDAF	Assemble and Factor a Matrix
C.4.2	RSDA RNDA CHDA CSDA CNDA	Assemble a Matrix
C.4.3	RSDAV CSDAV	Assemble Vectors
C.4.4	RSDF RNDF CHDF CSDF CNDF	Factor a Matrix
C.4.5	RSUPIV CSUPIV	Handle Zero Pivots.
C.4.6	RSDS RNDS CHDS CSDS CNDS	Solve
C.4.7	RSDEX RNDEX CHDEX CSDEX CNDEX	Extract a Submatrix

C.4.8	FMSMM	Out-of-Core Multiply **[C] = [D] + [A][B], [C] = [A][B], or [C] = [D] - [A][B].**
C.4.9	RSDMVM RNDMVM CHDMVM CSDMVM CNDMVM	Multiply **{Y} = [A]{X}** (FMSMM preferred)
C.4.10	RSDSVM CHDSVM CSDSVM	Multiply **{Y} = SUM([Si]{X})** (FMSMM preferred)
C.4.11	RNDVMM CNDVMM	Multiply **{Y} = {X}[F]**
C.4.12	RSDVVM CHDVVM CSDVVM	Multiply **[F] = {X}^{T}**{Y}, Symmetric **[F]**
C.4.12	RNDVVM CNDVVM	Multiply **[F] = {X}^{T}{Y}**, General **[F]**
C.4.13	RSDDVM CHDDVM CSDDVM	Multiply **[F] = {X}^{T}[D]{X}**, Symmetric
C.4.13	RNDDVM CNDDVM	Multiply **[F] = {X}^{T}[D]{Y}**, General
C.4.14	FMSVAN	Vector Norms, Add, Subtract, Move
C.4.15	RSDWTP	Accumulate **[S] = [S] + w[B]^{T}[D][B]**, Symmetric
C.4.15	RNDWTP	Accumulate **[S] = [S] + w[B]^{T}[D][A]**, General

Copyright © Multipath Corporation

C.4.1. SUBROUTINES RSDAF, RNDAF, CHDAF, CSDAF, CNDAF

PURPOSE

Assemble a global matrix and factor it into triangular form. Optionally perform forward reduction (and diagonal scaling) on right-hand side vectors.

SYNOPSIS

```
CALL RSDAF (LUAI, ALPHA, NUMAI, LUS, NUMSF, LUA, LUF, LUB, LUX, NUMRHS)
CALL RNDAF (LUAI, ALPHA, NUMAI, LUS, NUMSF, LUA, LUF, LUB, LUX, NUMRHS)
CALL CHDAF (LUAI, ALPHA, NUMAI, LUS, NUMSF, LUA, LUF, LUB, LUX, NUMRHS)
CALL CSDAF (LUAI, ALPHA, NUMAI, LUS, NUMSF, LUA, LUF, LUB, LUX, NUMRHS)
CALL CNDAF (LUAI, ALPHA, NUMAI, LUS, NUMSF, LUA, LUF, LUB, LUX, NUMRHS)
```

INPUT PARAMETERS

- LUAI(25, NUMAI) = Integer array.

  File attribute lists for input matrices used to initialize **[A]**.

- ALPHA(NUMAI) = Real array (RS, RN, CH).

  ALPHA(NUMAI) = Complex array (CS, CN).

  Scale factors for input matrices.

- NUMAI = Integer.

  Number of input matrices. Specify 0 and dummy arguments for LUAI and ALPHA to initialize **[A]** to **[O]**.

- LUS(25, NUMSF) = Integer array.

  File attributes of each submatrix file.

- NUMSF = Integer.

  Number of submatrix files. If no submatrix files are used, specify 0 for NUMSF and a dummy argument for LUS.

- LUA(25) = Integer array.

  File attributes for the assembled matrix **[A]**. If the assembled matrix **[A]** is not being saved, specify 0.

- LUF(25) = Integer array.

  File attributes for the matrix factors. If no matrix factoring is required, specify 0.

- LUB(25) = Integer array.

File attributes for right-hand side vectors. If no forward reduction is performed during factoring, a dummy argument should be specified.

- LUX(25) = Integer array.

File attributes for the file where reduced vectors are to be written. This parameter is usually the same as LUB to overlay reduced vectors on the original right-hand side vectors. If no forward reduction is performed during factoring, a dummy argument should be specified.

- NUMRHS = Integer.

Number of right-hand side vectors to reduce during the factoring process. If NUMRHS is specified zero, no forward reduction (and diagonal scaling for symmetric matrices) is performed.

OUTPUT PARAMETERS:

None.

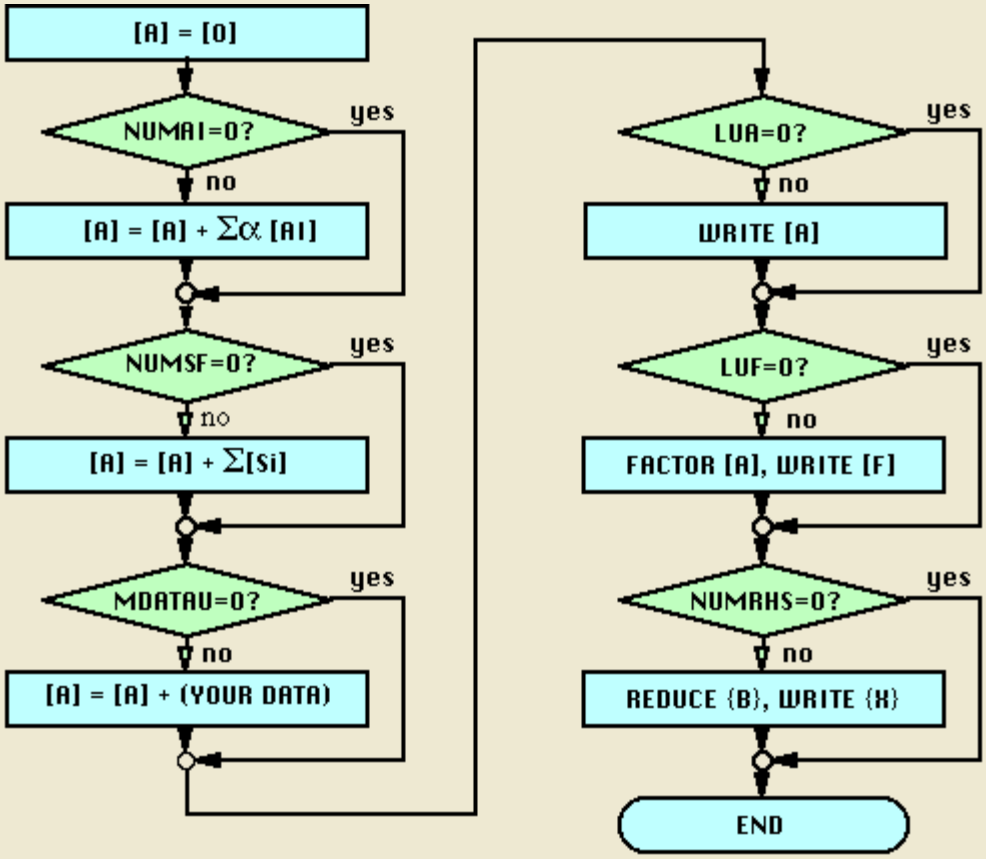
FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
	`Input Parameters`
IPRA	Assembly print code
IPRF	Factoring print code
IPRS	Solution print code
LOWASM	First changed equation
MDATAU	Call your subroutine to define or modify matrix data
NEQSUB	First equation of substructure
LUOK	Make a nonsymmetric matrix symmetric
MPOSDF	Positive definite flag
MZERO	Zero pivot flag
NBITSZ	Number of lost bits on diagonal to call zero
NBITSP	Number of lost bits on diagonal to print a warning
RPIVOT	Value to set zero pivot reciprocal
PIVMIN	Absolute value of minimum pivot

IALGOR	Optional factoring algorithms
MINDIM	Minimum dimension to terminate Strassen's algorithm
IPOLLD	CPU synchronization on diagonal elements
	Output Parameters
NUMSCG	Number of diagonal sign changes during factoring
NPIVOT	Number of equations pivoted (SLAB format)
CONINV	Inverse condition number
ATERMS	Number of terms in matrix
AZEROS	Number of zero's in matrix
FZEROS	Number of zero's in matrix factor
SAVOPS	Number of floating point operations saved

DESCRIPTION:



This subroutine performs the following steps:

1. **[A] = [0]**

   The matrix **[A]** is initialized to **[0]**. Therefore only the nonzero coefficients need to be defined.

2. **[A] = [A] +** SUM(ALPHA(i)***[AI]**(i)), m

Each of the initialization matrices LUAI(25,i) are scaled by their scale factor ALPHA(i) and added to the matrix **[A]**. The initialization matrices **[AI]** must have the same profile, symmetry, data type and record length as matrix **[A]**. This feature may be used for nonlinear or optimization analysis to initialize **[A]** to a previous value. For eigenvalue analysis, you may use the scale factor to shift the matrix. If you do not want to use initialization matrices, specify a 0 value for NUMAI and dummy arguments for LUAI and ALPHA. The ***FMS*** Parameter LUOK allows you to specify nonsymmetric matrices as input to the symmetric subroutines RSDAF and CSDAF. ***FMS*** averages the upper and lower triangle parts of the nonsymmetric matrix to make it symmetric.

3. **[A]** = **[A]** + SUM(**[S]**i), i=1,NUMSF

All of the submatrices on the files LUS(25,NUMSF) are then added to **[A]**. This assembly is typically used in finite element programs. You may skip submatrix addition by specifying a 0 value for NUMSF and a dummy argument for LUS.

4. Call your Subroutines

If the ***FMS*** Parameter MDATAU is set, one or more of the following subroutines you provide is called to define or modify **[A]**.

Matrix Format	Data Type	Matrix Symmetry	Subroutines Required
PROFLIE or BLOCK	Real	Symmetric	RSUBLK
PROFLIE or BLOCK	Real	Nonsymmetric	RSUBLK, RNUBLK
PROFLIE or BLOCK	Complex	Hermitian	CHUBLK
PROFLIE or BLOCK	Complex	Symmetric	CSUBLK
PROFLIE or BLOCK	Complex	Nonsymmetric	CSUBLK, CNUBLK
SLAB	Real	Nonsymmetric	RNUSLB
SLAB	Complex	Nonsymmetric	CNUSLB

5. Write **[A]** to file LUA.

If LUA(1) is not 0, the assembled matrix **[A]** is written to file LUA. If you plan to factor the matrix and do not need to save **[A]**, specify a zero value for LUA.

6. Factor **[A]**

The completed matrix **[A]** is then factored into one of the following forms:

[L][D][L]T = [A] for symmetric **[A]**

or

[L][U] = [A] for nonsymmetric **[A]**.

Here, **[L]** is a lower triangular matrix with Lii = 1 and Lij = 0 for i<j. The matrix **[D]** is diagonal with Dij = 0 for i not equal to j. The upper triangular matrix **[U]** has nonzero values on and above the diagonal with Uij = 0 for i>j. The symbol "T" is used to denote the transpose for symmetric problems.

If you do not want to perform matrix factoring, specify zero values for LUF(1) and NUMRHS and dummy arguments for LUB and LUX. ***FMS*** detects LUF(1)=0 and skips the rest of this subroutine.

7. Write the factored matrix to file LUF.

FMS writes the matrix factors to the file you provide described by the attributes LUF. You may overwrite the matrix factors on one of the input matrices by specifying the same file attributes for LUF that you specified for one of the LUAI's.

8. Reduce **{B}**

If you specified NUMRHS>0, ***FMS*** performs forward reduction (and diagonal scaling for symmetric problems) on the vectors in file LUB. This option saves reading the matrix factor **[L]** at the expense of rereading the right-hand side vectors **{B}** several times. If the number of solution vectors is small, or the solution vectors are in memory, this option should be used. However, if there are a large number of disk-based solution vectors, forward reduction should be performed using the solution subroutines RSDS, RNDS, CHDS, CSDS, CNDS). To skip this option, specify a zero value for NUMRHS and dummy arguments for LUB and LUX.

If you are only performing matrix assembly and are not using initialization matrices (NUMAI = 0, LUF = 0, NUMRHS = 0), you may use the assembly subroutines RSDA, RNDA, CHDA, CSDA, CNDA.

If you are only performing matrix factoring and forward reduction (NUMAI = 0, NUMSF = 0) and you have not specific the matrix data with FMSROW or FMSCOL you may use the factoring subroutines RSDF, RNDF, CHDF, CSDF, CNDF.

These subroutines provide reduced argument lists when assembly and factoring are performed as separate steps.

Copyright © Multipath Corporation

C.4.2. SUBROUTINES RSDA, RNDA, CHDA, CSDA, CNDA

PURPOSE

Assemble a global system matrix from submatrices and/or define the matrix coefficients by a subroutine you provide.

SYNOPSIS

```
CALL RSDA (LUS, NUMSF, LUA)
CALL RNDA (LUS, NUMSF, LUA)
CALL CHDA (LUS, NUMSF, LUA)
CALL CSDA (LUS, NUMSF, LUA)
CALL CNDA (LUS, NUMSF, LUA)
```

INPUT PARAMETERS

- LUS(25,NUMSF) = Integer array.
  File attribute list for each submatrix file.
- NUMSF = Integer.
  Number of submatrix files. If the matrix is populated entirely by the user-supplied subroutine, specify a 0 value for NUMSF and a dummy argument for LUS.
- LUA(25) = Integer array.
  File attribute list for matrix **[A]** that was computed by the initialization subroutine (RSDI, RNDI, CHDI, CSDI, CNDI).

OUTPUT PARAMETERS:

None.

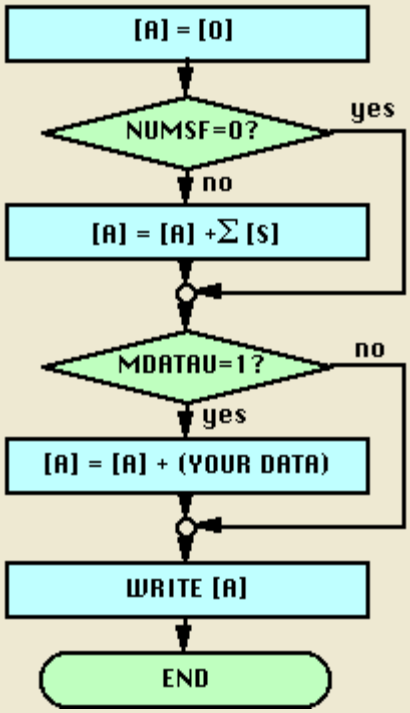
FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description

IPRA	Assembly print code
LOWASM	First changed equation
MDATAU	Call your subroutine to define matrix data

DESCRIPTION:



This subroutine populates the matrix **[A]** with data by the following steps:

1. **[A]** is initialized to **[0]**. Therefore, you only need to define the nonzero terms in **[A]**.
2. Submatrices are added to **[A]**. For finite element programs, submatrix addition provides the most efficient interface. You may skip submatrix addition by specifying a zero value for the number of submatrix files, NUMSF.
3. Your subroutine(s) are called to define or modify the matrix data. To use this option, the ***FMS*** Parameter MDATAU must be set. In addition, you must provide one or more of the following subroutines:

Matrix Format	Data Type	Matrix Symmetry	Subroutines Required
PROFLIE or BLOCK	Real	Symmetric	RSUBLK
PROFLIE or BLOCK	Real	Nonsymmetric	RSUBLK, RNUBLK
PROFLIE or BLOCK	Complex	Hermitian	CHUBLK
PROFLIE or BLOCK	Complex	Symmetric	CSUBLK
PROFLIE or BLOCK	Complex	Nonsymmetric	CSUBLK, CNUBLK
SLAB	Real	Nonsymmetric	RNUSLB

SLAB	Complex	Nonsymmetric	CNUSLB

The assembly process begins with the equation LOWASM, which is an ***FMS*** parameter. The default value of 1 should always be used the first time a matrix is assembled. For nonlinear analysis, you may set LOWASM to the first changed equation and ***FMS*** will only reassemble the part of the matrix beginning with equation LOWASM.

This subroutine is a subset of the more general assemble- factor subroutine. It is equivalent to one of the following calls:

CALL RSDAF(IDUM, RDUM, 0, LUS, NUMSF, LUA, 0, IDUM, IDUM, 0)
exit CALL RNDAF(IDUM, RDUM, 0, LUS, NUMSF, LUA, 0, IDUM, IDUM, 0)
CALL CHDAF(IDUM, RDUM, 0, LUS, NUMSF, LUA, 0, IDUM, IDUM, 0)
CALL CSDAF(IDUM, RDUM, 0, LUS, NUMSF, LUA, 0, IDUM, IDUM, 0)
CALL CNDAF(IDUM, RDUM, 0, LUS, NUMSF, LUA, 0, IDUM, IDUM, 0)

where IDUM and RDUM are dummy arguments. If you want to perform assembly as a separate step and your matrix is initialized to `[0]`, you may use this subroutine. Otherwise the more general assemble-factor subroutines should be used.

Copyright © Multipath Corporation

C.4.3. SUBROUTINES RSDAV, CSDAV

PURPOSE

Assemble one or more global vectors from submatrix vectors.

SYNOPSIS

```
CALL RSDAV (LUBI, LUS, NUMSF, LUB)
CALL CSDAV (LUBI, LUS, NUMSF, LUB)
```

INPUT PARAMETERS

- LUBI (25) = Integer array.

  File attribute list for initialization vector **{BI}**. To initialize **{B}** to **{0}** and compute **{B}**=Sum**{Vi}**, specify a value of 0.

- LUS(25, NUMSF) = Integer array.

  File attribute list for each submatrix file.

- NUMSF = Integer.

  Number of submatrix files.

- LUB(25) = Integer array.

  File attribute list for vector **{B}**. This may be the same as LUBI to compute **{B}**=**{B}**+Sum**{Vi}**.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IPRA	Assembly print code

DESCRIPTION:

This subroutine builds one or more global vectors in ***FMS*** format by adding together submatrix vectors in the

following steps:

1. The vector **{B}** is initialized to **{BI}**. If you have specified a 0 value for LUBI, the vector **{B}** is initialized to **{0}**.

2. Each submatrix vector **{Vi}** is assembled into the global vector **{B}** according to the equation number vector **{IEQSUB}** supplied on the corresponding submatrix integer record. The actual assembly process is equivalent to the following FORTRAN statements:

```
DO NV = 1, NUMVEC
   D0  I = 1, M
      IGLOB = IEQSUB(I)
      IF (IGLOB .GT. O) B(IGLOB, NV) = B(IGLOB, NV) + V(I,NV)
   END DO
END DO
```

The number of vectors assembled, NUMVEC, is the minimum of the number of vectors on the vector file and the number of vectors on the submatrix file.

The vector assembly process always begins with equation 1, and is not controlled by the ***FMS*** parameter LOWASM.

Copyright © Multipath Corporation

C.4.4. SUBROUTINES RSDF, RNDF, CHDF, CSDF, CNDF

PURPOSE

Factor a matrix into triangular form. Optionally perform forward reduction (and diagonal scaling) on right-hand side vectors.

SYNOPSIS

```
CALL RSDF (LUA, LUF, LUB, LUX, NUMRHS)
CALL RNDF (LUA, LUF, LUB, LUX, NUMRHS)
CALL CHDF (LUA, LUF, LUB, LUX, NUMRHS)
CALL CSDF (LUA, LUF, LUB, LUX, NUMRHS)
CALL CNDF (LUA, LUF, LUB, LUX, NUMRHS)
```

INPUT PARAMETERS

- LUA(25) = Integer Array.

  File attributes for matrix **[A]**.

- LUF(25) = Integer array.

  File attributes for the file where the matrix factors are to be written. These file attributes are usually the same as LUA to overlay the factored matrix on the original matrix to conserve space. If the original matrix is to be preserved, the FMSOM subroutine should be called to open the files for storing the matrix factors and create the LUF attribute list.

- LUB(25) = Integer array.

  File attributes for right-hand side vectors. If no forward reduction is performed during factoring, a dummy argument should be specified.

- LUX(25) = Integer array.

  File attributes for the file where reduced vectors are to be written. This parameter is usually the same as LUB to overlay reduced vectors on the original right-hand side vectors. If no forward reduction is performed during factoring, a dummy argument should be specified.

- NUMRHS = Integer.

  Number of right-hand side vectors to reduce during the factoring process. If NUMRHS is specified zero, no

forward reduction (and diagonal scaling for symmetric matrices) is performed during factoring.

OUTPUT PARAMETERS:

None.

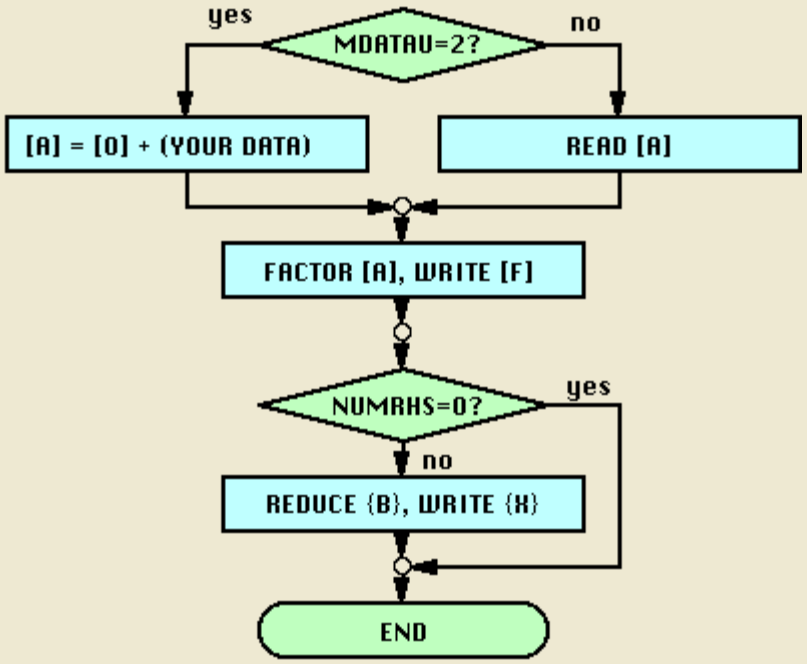
FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
	Input Parameters
IPRF	Factoring print code
IPRS	Solution print code
MDATAU	Call your subroutine to define or modify matrix data
NEQSUB	First equation of substructure
MPOSDF	Positive definite flag
MZERO	Zero pivot flag
NBITSZ	Number of lost bits on diagonal to call zero
NBITSP	Number of lost bits on diagonal to print a warning
RPIVOT	Value to set zero pivot reciprocal
PIVMIN	Absolute value of minimum pivot
IALGOR	Optional factoring algorithms
MINDIM	Minimum dimension to terminate Strassen's algorithm
IPOLLD	CPU synchronization on diagonal elements
	Output Parameters
NUMSCG	Number of diagonal sign changes during factoring
NPIVOT	Number of equations pivoted (SLAB format)
CONINV	Inverse condition number
ATERMS	Number of terms in matrix
AZEROS	Number of zero's in matrix

FZEROS	Number of zero's in matrix factor
SAVOPS	Number of floating point operations saved

DESCRIPTION:



This subroutine performs the following steps:

1. Obtain **[A]**

If the ***FMS*** parameter MDATAU is 2, the matrix data is obtained by initializing **[A]** to **[0]** and calling one or more of the subroutines you provided to define the matrix data:

Matrix Format	Data Type	Matrix Symmetry	Subroutines Required
PROFLIE or BLOCK	Real	Symmetric	RSUBLK
PROFLIE or BLOCK	Real	Nonsymmetric	RSUBLK, RNUBLK
PROFLIE or BLOCK	Complex	Hermitian	CHUBLK
PROFLIE or BLOCK	Complex	Symmetric	CSUBLK
PROFLIE or BLOCK	Complex	Nonsymmetric	CSUBLK, CNUBLK
SLAB	Real	Nonsymmetric	RNUSLB
SLAB	Complex	Nonsymmetric	CNUSLB

If MDATAU has any value other than 2, the matrix is read from file LUA and your subroutine(s) are not called.

2. Factor **[A]**

The completed matrix **[A]** is then factored into one of the following forms:

[L][D][L]T = [A] for symmetric **[A]**

or

[L][U] = [A] for nonsymmetric **[A]**.

Here, **[L]** is a lower triangular matrix with Lii = 1 and Lij = 0 for i<j. The matrix **[D]** is diagonal with Dij = 0 for i not equal to j. The upper triangular matrix **[U]** has nonzero values on and above the diagonal with Uij = 0 for i>j. The symbol "T" is used to denote the transpose for symmetric problems.

3. Write the factored matrix to file LUF.

FMS writes the matrix factors to the file you provide described by the attributes LUF. You may overwrite the matrix factors on the input matrix file LUA by specifying the same file attributes for LUF that you specified for LUA.

4. Reduce **{B}**

If you specified NUMRHS>0, ***FMS*** performs forward reduction (and diagonal scaling for symmetric problems) on the vectors in file LUB. This option saves reading the matrix factor **[L]** at the expense of rereading the right-hand side vectors **{B}** several times. If the number of solution vectors is small, or the solution vectors are in memory, this option should be used. However, if there are a large number of disk-based solution vectors, forward reduction should be performed using the solution subroutines RSDS, RNDS, CHDS, CSDS, CNDS. To skip this option, specify a zero value for NUMRHS and dummy arguments for LUB and LUX.

This subroutine is a subset of the more general assemble- factor subroutines RSDAF, RNDAF, CHDAF, CSDAF, CNDAF. It is equivalent to one of the following calls: `CALL RSDAF (LUA,1.0D0,NUMAI,IDUM,0,0,LUF,LUB,LUX,NUMRHS)`

`CALL RNDAF (LUA,1.0D0,NUMAI,IDUM,0,0,LUF,LUB,LUX,NUMRHS)`

`CALL CHDAF (LUA,1.0D0,NUMAI,IDUM,0,0,LUF,LUB,LUX,NUMRHS)`

`CALL CSDAF (LUA,1.0D0,NUMAI,IDUM,0,0,LUF,LUB,LUX,NUMRHS)`

`CALL CNDAF (LUA,1.0D0,NUMAI,IDUM,0,0,LUF,LUB,LUX,NUMRHS)` where IDUM is a dummy argument. The value of NUMAI is 0 if MDATAU = 2. Otherwise, NUMAI is 1. If you want to perform matrix factoring as a separate step, you may use this subroutine. Otherwise, you should use the more general assemble-factor subroutine.

Copyright © Multipath Corporation

C.4.5. SUBROUTINES RSUPIV, CSUPIV

PURPOSE

To provide a user interface for processing zero diagonal pivots.

This is a subroutine you provide.

SYNOPSIS

```
CALL RSUPIV (*, IEQN, A, S, D)
CALL CSUPIV (*, IEQN, A, S, D)
```

INPUT PARAMETERS

- IEQN = Integer.

  Diagonal equation number.

- A = Real (RSUPIV).

  A = Complex (CSUPIV).

  Diagonal of original matrix.

- S = Real (RSUPIV).

  S = Complex (CSUPIV).

  Inner product to be subtracted, <L(IEQN, k), U(k, IEQN)>

OUTPUT PARAMETERS:

- * = Alternate return.

  Return to your program where you called the ***FMS*** factoring subroutine.

- D = Real (RSUPIV).

  D = Complex (CSUPIV).

  Diagonal factor reciprocal, D=1/(A-S).

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
MZERO	Zero pivot flag

NBITSZ	Number of lost bits on diagonal to call zero
PIVMIN	Absolute value of minimum pivot

DESCRIPTION:

This subroutine provides you with low level access to the ***FMS*** zero pivot handler. You direct ***FMS*** to call this subroutine by specifying a value of 2 for the ***FMS*** Parameter MZERO.
When ***FMS*** determines that a zero pivot has occurred, based on the tolerance values of NBITSZ and PIVMIN, and MZERO is 2, this subroutine is called. ***FMS*** passes the equation number, original diagonal and sum to be subtracted. You are required to return the diagonal pivot reciprocal.

If you want to abandon the factoring process but continue execution, you may use the alternate return. Executing the FORTRAN statement

```
RETURN 1
```

in your subroutine will return processing to your program where you called the ***FMS*** factoring subroutine. Naturally the factoring calculation will be incomplete, requiring you to take alternate action.

The Pivot Precision Monitoring section describes how ***FMS*** determines if the pivot is zero and what ***FMS*** normally does with zero pivots. You should make every effort to use standard ***FMS*** features before developing this subroutine.

Copyright © Multipath Corporation

C.4.6. SUBROUTINES RSDS, RNDS, CHDS, CSDS, CNDS

PURPOSE

Solve a system of equations having one or more right-hand side vectors using a matrix **[A]** that has previously been factored.

SYNOPSIS

```
CALL RSDS (LUF, LUB, LUX, NUMRHS, ISKIP)
CALL RNDS (LUF, LUB, LUX, NUMRHS, ISKIP)
CALL CHDS (LUF, LUB, LUX, NUMRHS, ISKIP)
CALL CSDS (LUF, LUB, LUX, NUMRHS, ISKIP)
CALL CNDS (LUF, LUB, LUX, NUMRHS, ISKIP)
```

INPUT PARAMETERS

- LUF(25) = Integer array.

  File attributes for the factored matrix.

- LUB(25) = Integer array.

  File attributes for the right-hand side vectors. This parameter is not used if ISKIP=1.

- LUX(25) = Integer array.

  File attributes for the file where solution vectors are to be written. This can be the same parameter as LUB to overlay the solution vectors on the right-hand side vectors. If the parameter ISKIP=1, LUX specifies the file containing reduced vectors.

- NUMRHS = Integer.

  Number of right-hand side vectors.

- ISKIP = Integer.

  Operations to skip.

  The following options are available:

  - 0

    Perform forward reduction, diagonal scaling (symmetric) and back substitution.

  - 1

Skip forward reduction and diagonal scaling, perform back substitution.

- 2

Perform forward reduction and diagonal scaling, skip back substitution.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IPRA	Assembly print code
IPRS	Solution print code
NEQSUB	First equation of substructure
LUTRAN	Solve **[A]T{X}={B}** instead of **[A]{X}={B}**
IPBACK	Use dot products during back substitution
IALGOR	Optional factoring algorithms
MINDIM	Minimum dimension to terminate Strassen's algorithm
SAVOPS	Number of floating point operations saved

DESCRIPTION:

This subroutine performs forward reduction, diagonal scaling (symmetric), and back substitution on one or more systems of simultaneous equations. The solution uses the triangular and diagonal matrix factors **[L]**, **[D]**, and **[U]** computed in the factoring subroutine.

The forward reduction process consists of solving the system as follows:

[L]{Y} = {B}

for the intermediate vectors **{Y}** using the right-hand side vectors **{B}** and the lower- triangular matrix factor **[L]**.

The same forward reduction calculation is performed for symmetric and nonsymmetric matrices.

For symmetric problems, the second step is to scale the reduced vectors **{Y}** by the diagonal factor reciprocals **[1/D]** to obtain the following:

{Z} = [1/D] {Y}

The final step in the solution process is back substitution. For symmetric problems, the system:

```
[L] T {X} = {Z}
```

is solved for **{X}**, while for nonsymmetric problems

```
[U]{X} = {Y}
```

is solved for **{X}**.

This data structures used provide maximum performance when several solution vectors are processed together.

The ISKIP parameter can be used to skip forward reduction, diagonal scaling, or back substitution as shown in the following table:

ISKIP	Functions Skipped
0	None
1	Forward Reduction, Diagonal Scaling
2	Back Substitution

When ISKIP = 0, right-hand side vectors are read from the file LUB, and solution vectors are written to the file LUX.

When ISKIP = 1, reduced and scaled (symmetric) vectors are read from the file LUX and solution vectors are written to the file LUX. The file LUB is not used. This option should be selected if forward reduction was performed during factoring or if the solution process is being restarted after substructuring.

When ISKIP = 2, right-hand side vectors are read from the file LUB and reduced and scaled (symmetric) vectors are written to the file LUX. This calculation is equivalent to performing forward reduction during factoring. It is used to halt the solution process after forward reduction when substructuring is being performed.

The ***FMS*** substructuring parameter NEQSUB controls where the forward reduction stops and where back substitution begins. When the NEQSUB parameter is used, this subroutine is called the first time with ISKIP=2 to stop the solution after the partial forward reduction and diagonal scaling. After the substructure solution is computed and placed in file LUX, this subroutine is called a second time with ISKIP=1 to complete the solution.

Copyright © Multipath Corporation

C.4.7. SUBROUTINES RSDEX, RNDEX, CHDEX, CSDEX, CNDEX

PURPOSE

Extract a full incore matrix A(N,N) from the lower-right corner of a ***FMS*** matrix.

SYNOPSIS

```
CALL RSDEX (LUA, A, LDA, N)
CALL RNDEX (LUA, A, LDA, N)
CALL CHDEX (LUA, A, LDA, N)
CALL CSDEX (LUA, A, LDA, N)
CALL CNDEX (LUA, A, LDA, N)
```

INPUT PARAMETERS

- LUA(25) = Integer array.

  File attribute list.

- LDA = Integer.

  Leading dimension of array A. Usually N, but must at least as large as N.

- N = Integer.

  Number of equations to extract.

OUTPUT PARAMETERS:

- A(LDA,N) = Real array (RSDEX, RNDEX).

  A(LDA,N) = Complex array (CHDEX, CSDEX, CNDEX).

  Full matrix containing the N by N submatrix from the lower- right corner of the ***FMS*** matrix.

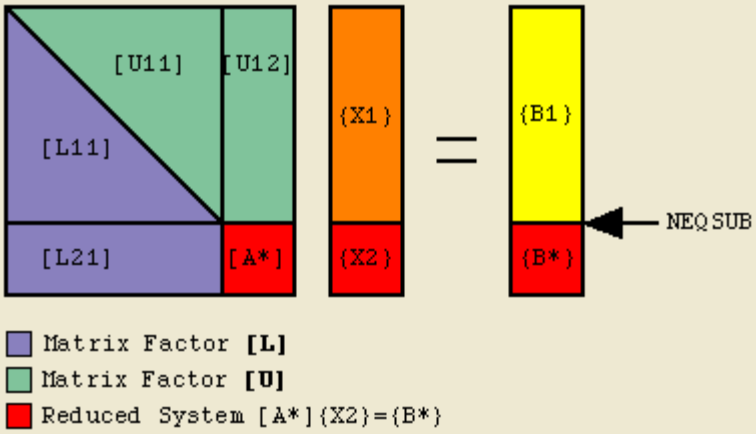
FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IPRF	Factoring print code

NEQSUB	First equation of substructure

DESCRIPTION:



This subroutine is used during substructuring to extract the reduced submatrix **[A*]** from the lower-right corner of the global matrix. On input, the file attribute list LUA(25) describes an ***FMS*** format matrix which has been partially factored by setting the NEQSUB parameter. The submatrix terms beginning with row and column NEQSUB have been partially reduced but not completely factored. The size of the submatrix, N, can be computed from the NEQSUB parameter and the number of equations NUMEQ as follows:

```
CALL FMSIGT ('NEQSUB', NEQSUB)
NUMEQ = LUA(8)
N = NUMEQ - NEQSUB + 1
```

Calling this subroutine stores the N by N submatrix in the array **[A]**. The leading dimension of **[A]**, LDA, can be larger than the number of equations to extract, N. The submatrix is stored in A(1:N,N). Terms A(N+1:LDA,N) are not used.

The array **A** must be aligned on a natural address boundary.

Copyright © Multipath Corporation

C.4.8. SUBROUTINE FMSMM

PURPOSE

Perform one of the matrix multiply operations

[C] = [D] - [A]T{B}, IACCUM = -1, TRANSA='T'

[C] = [D] - [A] {B}, IACCUM = -1, TRANSA='N'

[C] = [A]T{B}, IACCUM = 0, TRANSA='T'

[C] = [A] {B}, IACCUM = 0, TRANSA='N'

[C] = [D] + [A]T{B}, IACCUM = +1, TRANSA='T'

[C] = [D] + [A] {B}, IACCUM = +1, TRANSA='N'

SYNOPSIS

```
CALL FMSMM (LUC, LUD, IACCUM, TRANSA, LUA, LUB)
```

INPUT PARAMETERS

- LUC(25) = Integer array.

  Vector or matrix file attributes for the output array **[C]**.

- LUD(25) = Integer array.

  Vector or matrix file attributes for the input array **[D]**.

  This array is used to initialize the output array **[C]** when IACCUM is -1 or 1.

  This parameter must be the same type (matrix, vector) and size as LUC.

  This parameter may be the same as LUC.

- IACCUM = Integer.

  Determines if the matrix product of **[A]** and **[B]** is subtracted, stored into or added to **[C]**.

  This value overrides the ***FMS*** Parameter IACCUM.

  Possible values are -1, 0 and 1.

- TRANSA = Character*1.

  Determines if the Vector file **{A}** is transposed.

Possible values are 'N' and 'T'.

NOTE: TRANSA is always 'N' when **[A]** is a matrix or submatrix file.

- LUA(25) = Integer array.

Vector, matrix or submatrix file attributes for the input array **[A]**.

- LUB(25) = Integer array.

Vector file attributes for the input array **{B}**.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
MMROW	Starting row in **[C]** to store the product of **[A]** and **[B]**. Default value = 0
MMCOL	Starting column in **[C]** to store the product of **[A]** and **[B]**. Default value = 0
MMKA	Number of [0] columns to add to the beginning of **[A]** Default value = 0
IPRMV	Matrix-vectors multiply print code
IPRVV	Vectors-vectors multiply print code

DESCRIPTION:

This subroutine performs one of the following multiply operations:

CASE	LUC and LUD	IACCUM	TRANSA	LUA	LUB	Compute
1	Vector File	-1 0 1	'N'	Matrix File	Vector File	{C} = {D} - [A]{B} {C} = [A]{B} {C} = {D} + [A]{B}
2	Vector File	-1 0 1	'N'	Submatrix File	Vector File	{C} = {D} - [Si]{B} {C} = [Si]{B} {C} = {D} + [Si]{B}

3	Vector File	-1 0 1	'N'	Vector File	Vector File	{C} = {D} - {A}{B} {C} = {A}{B} {C} = {D} + {A}{B}
4	Vector File	-1 0 1	'T'	Vector File	Vector File	{C} = {D} - {A}T{B} {C} = {A}T{B} {C} = {D} + {A}T{B}
5	Matrix File	-1 0 1	'T'	Vector File	Vector File	[C] = [D] - {A}T{B} [C] = {A}T{B} [C] = [D] + {A}T{B}

The calculation proceeds in two steps:

1. First, the output array **[C]** is initialized to the input array **[D]** when IACCUM is -1 or 1. When IACCUM=0, **[C]** is initialized to **[0]**.
2. The matrix product **[A][B]** or **[A]T[B]** is then subtracted from **[C]**, stored into **[C]** or added to **[C]**, according to the value of IACCUM (-1, 0, or 1). The matrix **[A]** may be transposed (TRANSA='T') or not (TRANSA='N').

Normally the product of **[A]** and **[B]** is stored in **[C]** starting at the first row and column of matrix **[C]**. However, the ***FMS*** Parameters MMROW and MMCOL may be used to shift the storage to start at C(1+MMROW,1+MMCOL). The default values of MMROW and MMCOL are 0. This option only applies when [C] is a matrix file in block or slab format.

If any part of the product of **[A]** and **[B]**, including any shifting by MMROW and MMCOL, falls outside of the matrix **[C]**, those values are not computed. If the product of **[A]** and **[B]** is smaller than **[C]**, only the appropriate terms in **[C]** are modified.

The ***FMS*** Parameter MMKA may be used to shift the matrix **[A]** by adding MMKA columns of zero's. The shifted matrix **[A]** then becomes **[0|A]**. The default value of MMKA is 0.

EXAMPLE_15 illustrates how to use this subroutine and the ***FMS*** Parameters MMROW, MMCOL and MMKA.

Copyright © Multipath Corporation

C.4.9. SUBROUTINES RSDMVM, RNDMVM, CHDMVM, CSDMVM, CNDMVM

PURPOSE

Multiply a group of vectors **{X}** by a matrix **[A]** and place the results in vectors **{Y}**, according to the ***FMS*** Parameter IACCUM

{Y} = {Z} - [A]{X}, IACCUM = -1,

{Y} = [A]{X}, IACCUM = 0, (default)

{Y} = {Z} + [A]{X}, IACCUM = +1,

SYNOPSIS

```
CALL RSDMVM (LUA, LUX, LUY, NUMVEC, LUZ)
CALL RNDMVM (LUA, LUX, LUY, NUMVEC, LUZ)
CALL CHDMVM (LUA, LUX, LUY, NUMVEC, LUZ)
CALL CSDMVM (LUA, LUX, LUY, NUMVEC, LUZ)
CALL CNDMVM (LUA, LUX, LUY, NUMVEC, LUZ)
```

INPUT PARAMETERS

- LUA(25) = Integer array.
  File attributes for matrix **[A]**.
- LUX(25) = Integer array.
  File attributes for the file containing **{X}** vectors.
- LUY(25) = Integer array.
  File attributes for the file where the **{Y}** vectors are to be written. This parameter may be the same as LUX to overlay the **{Y}** vectors on the **{X}** vectors.
- NUMVEC = Integer.
  Number of vectors to multiply.
- LUZ(25) = Integer array.
  File attributes for the file containing **{Z}** vectors. This parameter is only used if the ***FMS*** Parameter IACCUM is set to a nonzero value. This parameter may be the same as LUY to accumulate the results in **{Y}**.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IACCUM	Product accumulation flag = -1, **{Y} = {Z} - {X}T[F]** = 0, **{Y} = {X}T[F]** (default) = +1, **{Y} = {Z} - {X}T[F]**
IPRMV	Matrix-vectors multiply print code

DESCRIPTION:

This subroutine multiplies a group of vectors **{X}** stored on file LUX by a matrix stored in ***FMS*** format on file LUA. The resulting vectors **{Y}** are written to file LUY, which can be the same as file LUX.
The calculation proceeds in two steps:

1. First, the **{X}** vectors are scaled by the diagonals on file LUA(2) to obtain the following:

   {Y} = [D]{X}

2. For symmetric problems, the second step consists of computing the following:

   {Y} = {Y} + [AL]{X} + [AL] T {X}

   while for nonsymmetric problems the following two calculations are performed:

   {Y} = {Y} + [AL]{X}

   {Y} = {Y} + [AU]{X}

The matrix-vectors calculation can be used to check the solution of the system of equations
[A]{X} = {B}

or as the first step in projecting a matrix **[A]** into a subspace **[A*]**,

[A*] = {X} T [A]{X}

The data structures used are designed to obtain maximum performance when several **{X}** vectors are processed in parallel.

Copyright © Multipath Corporation

C.4.10. SUBROUTINES RSDSVM, RNDSVM, CHDSVM, CSDSVM, CNDSVM

PURPOSE

Multiply a group of vectors **{X}** by submatrices **[S]i** and place the results in vectors **{Y}**, according to the ***FMS*** Parameter IACCUM

{Y} = {Z} - SUM([S]i{X}), IACCUM = -1

{Y} = SUM([S]i{X}), IACCUM = 0 (default)

{Y} = {Z} + SUM([S]i{X}), IACCUM = +1

SYNOPSIS

```
CALL RSDSVM (LUS, NUMSF, LUX, LUY, NUMVEC, LUZ)
CALL CHDSVM (LUS, NUMSF, LUX, LUY, NUMVEC, LUZ)
CALL CSDSVM (LUS, NUMSF, LUX, LUY, NUMVEC, LUZ)
```

INPUT PARAMETERS

- LUS(10,NUMSF) = Integer array.

  File attributes for each submatrix file.

- NUMSF = Integer.

  Number of submatrix files.

- LUX(25) = Integer array.

  File attributes for the file containing **{X}** vectors.

- LUY(25) = Integer array.

  File attributes for the file where the **{Y}** vectors are to be written. This parameter may be the same as LUX to overlay the **{Y}** vectors on the **{X}** vectors.

- NUMVEC = Integer.

  Number of vectors to multiply.

- LUZ(25) = Integer array.

  File attributes for the **{Z}** vectors. This parameter is only used if the ***FMS*** Parameter IACCUM is set to a nonzero value. This parameter may be the same as LUY to accumulate the results in **{Y}**.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IACCUM	Product accumulation flag = -1, **{Y} = {Z} - SUM([S]i{X})**, = 0, **{Y} = SUM([S]i{X})**, (default) = +1, **{Y} = {Z} + SUM([S]i{X})**
IPRMV	Matrix-vectors multiply print code

DESCRIPTION:

This subroutine multiplies a group of vectors **{X}** stored on file LUX by the submatrices stored on files LUS. The resulting vectors **{Y}** are written to file LUY, which may be the same as file LUX.

This subroutine is equivalent to assembling the matrix **[A]** with subroutines RSDA, RNDA, CHDA, CSDA, CNDA and then performing matrix-vectors multiply **{Y}=[A]{X}** using RSDMVM, RNDMVM, CHDMVM, CSDMVM, CNDMVM. If the assembled matrix **[A]** contains a large number of zero terms within the profile, this subroutine should be used. If the matrix is densely populated and several vectors are to be multiplied, the matrix **[A]** should be assembled as a separate step.

Copyright © Multipath Corporation

C.4.11. SUBROUTINES RSDVMM, RNDVMM, CHDVMM, CSDVMM, CNDVMM

PURPOSE

Premultiply a full incore matrix **[F]** by a group of vectors **{X}** and place the results in vectors **{Y}**, according to the ***FMS*** Parameter IACCUM

{Y} = {Z} - {X}[F], IACCUM = -1,

{Y} = {X}[F], IACCUM = 0, (default)

{Y} = {Z} + {X}[F], IACCUM = +1,

SYNOPSIS

```
CALL RNDVMM (LUX, F, LUY, NUMX, NUMY, LUZ)
CALL CNDVMM (LUX, F, LUY, NUMX, NUMY, LUZ)
```

INPUT PARAMETERS

- LUX(25) = Integer array.
  File attributes for the **{X}** vectors.
- F(NUMX, NUMY) = Real array (RNDVMM).
  F(NUMX, NUMY) = Complex array (CNDVMM).
  Full incore matrix in standard FORTRAN format.
- LUY(25) = Integer array.
  File attributes for the file where the **{Y}** vectors are to be written. This parameter may be the same as LUX to overlay the **{Y}** vectors on the **{X}** vectors.
- NUMX = Integer.
  Number of **{X}** vectors.
- NUMY = Integer.
  Number of **{Y}** vectors.
- LUZ(25) = Integer array.
  File attributes for the **{Z}** vectors. This parameter is only used if the ***FMS*** Parameter IACCUM is set to a nonzero value. This parameter may be the same as LUY to accumulate the results in **{Y}**.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IACCUM	Product accumulation flag = -1, **{Y} = {Z} - {X}[F],** = 0, **{Y} = {X}[F],** (default) = +1, **{Y} = {Z} - {X}[F],**
IPRMV	Vectors-matrix multiply print code

DESCRIPTION:

This subroutine multiplies a full matrix **[F]** stored in memory by a group of vectors **{X}** stored on file LUX. The resulting vectors **{Y}** are written to file LUY, which may be the same as file LUX.

The array **[F]** must be aligned on a natural address boundary.

Copyright © Multipath Corporation

C.4.12. SUBROUTINES RSDVVM, RNDVVM, CHDVVM, CSDVVM, CNDVVM

PURPOSE

Multiply a group of vectors **{Y}** by the transpose of a group of vectors **{X}** and store the results in matrix **[F]** according to the ***FMS*** Parameter IACCUM

[F] = [F] - {X}T{Y}, IACCUM = -1,

[F] = {X}T{Y}, IACCUM = 0, (default)

[F] = [F] + {X}T{Y}, IACCUM = +1,

SYNOPSIS

```
CALL RSDVVM (LUX, LUY, F, NUMXY)
CALL RNDVVM (LUX, LUY, F, NUMX, NUMY)
CALL CHDVVM (LUX, LUY, F, NUMXY)
CALL CSDVVM (LUX, LUY, F, NUMXY)
CALL CNDVVM (LUX, LUY, F, NUMX, NUMY)
```

INPUT PARAMETERS

- LUX(25) = Integer array.
  File attributes for the **{X}** vectors.
- LUY(25) = Integer array.
  File attributes for the **{Y}** vectors. This parameter can be the same as LUX to compute **[F]={X}T{X}**.
- NUMXY = Integer. (RSDVVM, CHDVVM, CSDVVM only)
  Number of **{X}** or **{Y}** vectors.
- NUMX = Integer. (RNDVVM, CNDVVM only)
  Number of **{X}** vectors.
- NUMY = Integer. (RNDVVM, CNDVVM only)
  Number of **{Y}** vectors.

OUTPUT PARAMETERS:

- F(NUMXY*(NUMXY+1)/2) = Real array (RSDVVM).

F(NUMX, NUMY) = Real array (RNDVVM).

F(NUMXY*(NUMXY+1)/2) = Complex array (CHDVVM, CSDVVM).

F(NUMX, NUMY) = Complex array (CNDVVM).

This array contains the matrix formed from the inner products of the **{X}** and **{Y}** vectors. For symmetric problems (RSDVVM, CHDVVM, CSDVVM) this array is stored in lower triangular form, starting from column 1 through the diagonal by increasing row number. For nonsymmetric problems (RNDVVM, CNDVVM) this array is stored in standard FORTRAN storage.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IACCUM	Product accumulation flag = -1, **[F] = [F] - {X}T{Y},** = 0, **[F] = {X}T{Y},** (default) = +1, **[F] = [F] + {X}T{Y},**
IPRVV	Vectors-vectors multiply print code

DESCRIPTION:

This subroutine forms a matrix Fij from the inner products of vectors **{X}i** and **{Y}j** . The vector groups **{X}** and **{Y}** are supplied on files LUX and LUY. The matrix **[F]** is formed and left in memory.

For symmetric problems, only the lower triangle is computed. Data is stored in a packed triangular format similar to submatrix type 4 described in the call to FMSOS. The calculation performed is equivalent to the following FORTRAN statements:

```
L = 0
DO I = 1,NUMXY
   DO J = l,I
      L = L + 1
      F(L) = 0.
      DO K = 1,NUMEQ
         F(L) = F(L) + X(K,I)*Y(K,J)
      END DO
   END DO
END DO
```

In general, the inner products of **{X}** and **{Y}** vectors do not produce a symmetric matrix **[F]**. However, if the **{Y}** vectors were obtained from the matrix- vectors multiply operation **{Y}=[A]{X}** and **[A]** is a symmetric

matrix, then **[F]** is symmetric.

For nonsymmetric problems, **[F]** may have nonsymmetric values as well as a different number of rows and columns. The storage used for **[F]** is the standard FORTRAN storage for F(NUMX, NUMY). The calculation performed is equivalent to the following FORTRAN statements:

```
DO I  = 1,NUMX
   DO J = 1,NUMY
      F(I,J) = 0 .
      DO K = 1,NUMEQ
         F(I,J) = F(I,J) + X(K,I)*Y(K,J)
      END DO
   END DO
END DO
```

This vectors-vectors multiply calculation is frequently used with the matrix-vectors multiply calculation for computing quadratic forms and projecting the matrix **[A]** into a subspace. The data structure used provides the maximum speed when several **{X}** and **{Y}** vectors are processed in parallel.

The array **[F]** must be aligned on a natural address boundary.

Copyright © Multipath Corporation

C.4.13. SUBROUTINES RSDDVM, RNDDVM, CHDDVM, CSDDVM, CNDDVM

PURPOSE

Compute the quadratic form **{X}T[D]{X}** (symmetric) or **{X}T[D]{Y}** (nonsymmetric) where **[D]** is a diagonal matrix. Store the result in matrix **[F]** according to the ***FMS*** Parameter IACCUM

[F] = [F] - {X}T[D]{Y}, IACCUM = -1,

[F] = {X}T[D]{Y}, IACCUM = 0, (default)

[F] = [F] + {X}T[D]{Y}, IACCUM = +1.

For symmetric problems, **{Y} = {X}.**

SYNOPSIS

```
CALL RSDDVM (LUX, LUD, F, NUMXY)
CALL RNDDVM (LUX, LUD, LUY, F, NUMX, NUMY)
CALL CHDDVM (LUX, LUD, F, NUMXY)
CALL CSDDVM (LUX, LUD, F, NUMXY)
CALL CNDDVM (LUX, LUD, LUY, F, NUMX, NUMY)
```

INPUT PARAMETERS

- LUX(25) = Integer array.
  File attributes for the **{X}** vectors.
- LUD(25) = Integer array.
  File attributes for the diagonal **[D]**, stored as a vector file.
- LUY(25) = Integer array. (RNDDVM, CNDDVM only)
  File attributes for the **{Y}** vectors. If LUY is the same as LUX, the symmetric version of this subroutine should be called to compute **[F]** more rapidly.
- NUMXY = Integer. (RSDDVM, CHDDVM, CSDDVM only)
  Number of **{X}** vectors.
- NUMX = Integer. (RNDDVM, CNDDVM only)
  Number of **{X}** vectors.

NUMY = Integer. (RNDDVM, CNDDVM only)

Number of **{Y}** vectors.

OUTPUT PARAMETERS:

- F(NUMXY*(NUMXY+1)/2) = Real array (RSDDVM).

  F(NUMX, NUMY) = Real array (RNDDVM).

  F(NUMXY*(NUMXY+1)/2) = Complex array (CHDDVM, CSDDVM).

  F(NUMX, NUMY) = Complex array (CNDDVM).

  This array contains the matrix formed from the quadratic form **{X}T[D]{X}** (symmetric) or **{X}T[D]{Y}** (nonsymmetric). For symmetric problems (RSDDVM, CHDDVM, CSDDVM) this array is stored in lower triangular form, starting from column 1 through the diagonal by increasing row number. For nonsymmetric problems (RNDDVM, CNDDVM) this array is stored in standard FORTRAN storage.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IACCUM	Product accumulation flag = -1, **[F] = [F] - {X}T[D]{Y},** = 0, **[F] = {X}T[D]{Y},** (default) = +1, **[F] = [F] + {X}T[D]{Y},**
IPRDV	Vectors-diagonal-vectors multiply print code

DESCRIPTION:

For symmetric problems, this subroutine forms the lower triangular matrix **[F]** from the quadratic form **{X}T[D]{X}**. The **{X}** vectors are supplied on file LUX and the diagonal on file LUD. The matrix **[F]** is stored in a packed triangular format similar to submatrix type 4 described in the call to FMSOS. The calculation performed is equivalent to the following FORTRAN statements:

```
L = 0
DO I = 1,NUMXY
   DO J = 1,I
      L = L + 1
      F(L) = 0.
      DO K = 1,NUMEQ
         F(L) = F(L) + X(K,I)*D(K,K)*Y(K,J)
      END DO
```

```
      END DO
   END DO
```

For nonsymmetric problems, this subroutine forms a full matrix **[F]** from the quadratic form **{X}T[D]{Y}**. The vector groups **{X}** and **{Y}** are supplied on the files LUX and LUY and the diagonal on file LUD. The matrix **[F]** is stored in standard FORTRAN storage for F(NUMX, NUMY). The calculation performed is equivalent to the following FORTRAN statements:

```
   DO I  = 1,NUMX
      DO J = 1,NUMY
         F(I,J) = 0 .
         DO K = 1,NUMEQ
            F(I,J) = F(I,J) + X(K,I)*D(K,K)*Y(K,J)
         END DO
      END DO
   END DO
```

The vectors-diagonal-vectors calculation provides an efficient and convenient way to evaluate quadratic forms when the matrix is diagonal.

The array **[F]** must be aligned on a natural address boundary.

Copyright © Multipath Corporation

C.4.14. SUBROUTINE FMSVAN

PURPOSE

This subroutine provides a collection of vector utilities primarily used in iterative algorithms. Several processing options are included, as determined by the parameter IOPT. Generally **{X}** represents the solution values and **{Y}** the change in solution values, or the solution at a different iteration. The norm RNORM is the ratio of solution increment to solution value.

The following options are implemented:

IOPT	IALPHA	{X} on Output	RNORM
0	Not Used	Unchanged	|| {X} ||
1	-1	**{X}={X}-{Y}**	|| Y(i) / (X(i)-Y(i)) ||
1	+1	**{X}={X}+{Y}**	|| Y(i) / (X(i)+Y(i)) ||
2	Not Used	Unchanged	|| (X(i)-Y(i)) / X(i) ||
3	Not Used	**{X}={Y}**	|| (Y(i)-X(i)) / Y(i) ||

SYNOPSIS

```
CALL FMSVAN (LUX, IALPHA, LUY, INORM, RNORM, IOPT)
```

INPUT PARAMETERS

- LUX(25) = Integer array.

  File attributes for the **{X}** vectors.

- IALPHA = Integer.

  Scale factor for **{Y}** vectors.

- LUY(25) = Integer array.

  File attributes for the **{Y}** vectors.

- INORM = Integer.

  Number of the norm to compute:

```
RNORM = ( SUM( ABS(X(i))**INORM ) )**(1/INORM)
```

The exception being when INORM=0, the infinity norm is computed.

The implemented norms are:

INORM	RNORM
0	MAX(ABS(X(i)))
1	SUM(ABS(X(i)))
2	SQRT(SUM(X(i)**2))

- IOPT = Integer.

Processing options.

OUTPUT PARAMETERS:

- RNORM = Real.

Vector norm.

DESCRIPTION:

This subroutine is a collection of vector utilities that are typically used in iterative algorithms. The computed norm RNORM may be used to test for convergence.
The parameter IOPT determines if the vector **{X}** is modified and how the norm is computed. The following options are implemented:

- IOPT=0

Compute the norm of **{X}**. On exit, **{X}** is unchanged. The parameters IALPHA and LUY are not used.

- IOPT=1

{X} is the solution and **{Y}** is the increment, which may be added (IALPHA=1) to, or subtracted (IALPHA=-1) from **{X}**. The norm RNORM is the ratio of the increment to the new solution value.

- IOPT=2

{X} is the new solution and **{Y}** is the previous solution. The norm RNORM is the ratio of the change in solution to the new solution value. On exit, **{X}** is unchanged. The parameter IALPHA is not used.

- IOPT=3

{X} is the previous solution and **{Y}** is the new solution. The norm RNORM is the ratio of the change in solution to the new solution value. On exit, the new solution value **{Y}** is moved to the previous solution **{X}**. The parameter IALPHA is not used.

Copyright © Multipath Corporation

C.4.15. SUBROUTINES RSDWTP, RNDWTP

PURPOSE

Compute the matrix triple product, multiply it by a scalar and add it to a matrix:

[S] = [S] + W{B}T[D]{B} (Symmetric)

[S] = [S] + W{B}T[D]{A} (Nonsymmetric)

SYNOPSIS

```
CALL RSDWTP (W, B, D, S, N, M)
CALL RNDWTP (W, B, D, A, S, N, M)
```

INPUT PARAMETERS

- W = Real.

  Scale Factor.

- B(N,M) = Real array.

  Matrix stored by columns.

- D(N,N) = Real array.

  Full matrix stored by columns. For symmetric problems (RSDWTP), **[D]** must be symmetric.

- A(N,M) = Real array (RNDWTP only).

  Matrix stored by columns.

- N = Integer.

  Number of rows in matrix **[B]** and **[A]** and the size of the square matrix **[D]** . The maximum value of N must be less than or equal to 32. This restriction on N results from the use of high-speed internal registers that store intermediate values for this calculation. For most problems, N is in the range 3 to 6.

- M = Integer.

  Number of columns in matrix **[B]** and **[A]**.

OUTPUT PARAMETERS:

- S(M*(M+1)/2) = Real array (RSDWTP).

  S(M,M) = Real array (RNDWTP).

  For symmetric problems (RSDWTP), **[S]** is a lower triangular matrix stored by rows. For nonsymmetric

problems, standard FORTRAN storage is used for **[S]**.

DESCRIPTION:

This subroutine performs the matrix triple product that is typically used in finite element matrix generation. Before the first call in each element, the matrix values in **[S]** must be initialized to **[0]**. The matrix **[B]** is assumed full to provide for formulations including large deformation. If possible, the columns of matrix **[B]** should be ordered by increasing global equation number. This ordering produces a matrix **[S]** that is also ordered by increasing equation number and may be assembled into the global system with minimal computation. The matrix **[D]** is also full, which provides for orthotropic and rotated material relationships.

For symmetric problems (RSDWTP), the calculation performed is equivalent to the following FORTRAN statements.

```
LS = 0
DO I=1,M
   DO J=1,I
      LS = LS  +  1
      DO K=1,N
         DO L=1,N
            S(LS) = S(LS) + W*B(K,I)*D(K,L)*B(L,J)
         END DO
      END DO
   END DO
END DO
```

For nonsymmetric problems (RNDWTP), the calculation performed is equivalent to the following FORTRAN statements:

```
DO I=1,M
   DO J=1,I
      DO K=1,N
         DO L=1,N
            S(I,J) = S(I,J) + W*B(K,I)*D(K,L)*A(L,J)
         END DO
      END DO
   END DO
END DO
```

The actual algorithm implemented uses an intermediate vector to reduce the number of operations performed and take advantage of high-speed hardware.

Copyright © Multipath Corporation

C.5. STEP5: READ DATA FROM FMS FILES

Index	NAME	Function
C.5.1	FMSRED FMSRD8 FMSSRD	Direct Access Read of a Vector File Allows large disk addresses on 32-bit machines Sequential Read of a Vector File
C.5.2	FMSGET	Read a Block of Data from a Vector File

Copyright © Multipath Corporation

C.5.1. SUBROUTINES FMSRED, FMSRD8, FMSSRD

PURPOSE

To read data from a ***FMS*** file.

SYNOPSIS

```
CALL FMSRED (LOGUNT, LOCUNT, ARRAY, NUMWRD)
CALL FMSRD8 (LOGUNT, LOCUNT, ARRAY, NUMWRD)
CALL FMSSRD (LOGUNT, ARRAY, NUMWRD)
```

INPUT PARAMETERS

- LOGUNT = Integer.

  The ***FMS*** file number, which is stored by ***FMS*** in the file attribute list.

- LOCUNT = Integer. (FMSRED only)

  LOCUNT = REAL*8. (FMSRD8 only)

  The file address (in words) where you want to start to read. The first word on the file is located at LOCUNT=1. . For FMSRED, this is an INTEGER. For FMSRD8, this is a REAL*8 Floating Point Number. FMSRD8 allows the addressing of files greater than 2 Gbytes on systems supporting only 32-bit integers.

- ARRAY(NUMWRD) = Complex, Real or Integer array.

  Memory array where you want to receive data.

- NUMWRD = Integer.

  Number of WORDS to transfer. For complex data, count each real and imaginary pair as two words.

OUTPUT PARAMETERS:

None.

DESCRIPTION:

These subroutines provide read access to ***FMS*** files.

Subroutine FMSRED provides random access, subroutine FMSSRD provides sequential access. You may reset the file location used by FMSSRD by calling FMSEEK, FMSRED, FMSWRT or FMSSWR. When using FMSSRD to

read a fixed-length record file, you must specify the record length for NUMWRD.

These subroutines are typically used to read vectors after ***FMS*** processing.

Copyright © Multipath Corporation

C.5.2. SUBROUTINES FMSGET

PURPOSE

To read a block of data from a ***FMS*** Vector File.

SYNOPSIS

```
CALL FMSGET (LUFMS, NROWS, NCOLS, IROW1, JCOL1, A(NROWS,NCOLS), LDA)
```

INPUT PARAMETERS

- LUFMS(25) = Integer array.

FMS Vector File Attribute List which defines the file where the data is to be read.

- NROWS = Integer.

The number of rows to transfer.

- NCOLS = Integer.

The number of columns to transfer.

- IROW1 = Integer.

The starting global row where you want to read the block.

- JCOL1 = Integer.

The starting global column where you want to read the block.

- LDA = Integer.

The leading dimension of the DATA array. This must be at least as big as NROWS.

OUTPUT PARAMETERS:

- DATA(NROWS,NCOLS) = Real or Complex array.

This is the array of data you want to transfer from the file, in normal FORTRAN storage.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IOKIDS	Allow children to access a file

DESCRIPTION:

This subroutine provides read access to ***FMS*** Vector files. The array DATA(NROWS,NCOLS) is transferred directly from the file LUFMS without buffering.

To use this subroutine, the file LUFMS must be opened and populated with data. Using this subroutine involves the following 3 steps:

1. Initialize

```
CALL FMSGET (LUFMS, 0, 0, 0, 0, 0, 0)
```

The 0 value arguments direct ***FMS*** to initialize the files for read access. This call must precede calls to FMSGET which transfer data.

2. Transfer data

```
CALL FMSGET (LUFMS, NROWS, NCOLS, IROW1, JCOL1, A(NROWS,NCOLS), LDA)
```

The second step is to call FMSGET as many times as you want to read the data.

3. End

```
CALL FMSGET (LUFMS, 0, 0, NUMEQ+1, 0, 0, 0)
```

where the value of IROW1=NUMEQ+1 is greater than the number of equations. This call directs ***FMS*** to release any storage.

You may call FMSGET from subroutines you are running in parallel (which you started by calls to FMSPAR). To use this feature, you must direct ***FMS*** to open the file LUFMS for access by multiple processes. Use the ***FMS*** Parameter IOKIDS before the call which opens file LUFMS.

Examples

Example 13 further illustrates how to use FMSGET to populate a matrix and vectors.

Copyright © Multipath Corporation

C.6. STEP6: CLOSE FMS FILES AND END

Index	NAME	Function
C.6.1	FMSCV	Close a Vector File
C.6.2	FMSCS	Close a Submatrix File
C.6.3	FMSCM	Close a Matrix File
C.6.4	FMSEND	End FMS - Last Subroutine Called

Copyright © Multipath Corporation

C.6.1. SUBROUTINES FMSCV

PURPOSE

To close and optionally delete a file used to store vectors.

SYNOPSIS

```
CALL FMSCV (LUX)
```

INPUT PARAMETERS

- LUX(25) = Integer array.

FMS Vector File Attribute List for the vector file to be closed.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IFKEEP	Keep files

DESCRIPTION:

This subroutine closes the file associated with a group of vectors. If the ***FMS*** Parameter IFKEEP is 0 (default), the file is also deleted.

Copyright © Multipath Corporation

C.6.2. SUBROUTINE FMSCS

PURPOSE

To close and optionally delete a file used to store submatrices.

SYNOPSIS

```
CALL FMSCS (LUS)
```

INPUT PARAMETERS

- LUS(25) = Integer array.

FMS Submatrix File Attribute List for the submatrix file to be closed.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IFKEEP	Keep files

DESCRIPTION:

This subroutine closes the submatrix file. If multiple submatrix files are used, this subroutine must be called for each file. If the ***FMS*** Parameter IFKEEP is 0 (default), the file is also deleted.

Copyright © Multipath Corporation

C.6.3. SUBROUTINE FMSCM

PURPOSE

To close and optionally delete the files associated with a matrix.

SYNOPSIS

```
CALL FMSCM (LUA)
```

INPUT PARAMETERS

- LUA(25) = Integer array.

FMS Matrix File Attribute List for the matrix file to be closed.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
IFKEEP	Keep files

DESCRIPTION:

This subroutine closes the files associated with a matrix. If the ***FMS*** Parameter IFKEEP is 0 (default), the files are also deleted.

Copyright © Multipath Corporation

C.6.4. SUBROUTINES FMSEND

PURPOSE

To terminate ***FMS*** processing and print performance reports.

SYNOPSIS

```
CALL FMSEND
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS:

None.

DESCRIPTION:

This is the last ***FMS*** subroutine called and should be the last executable statement in your application.

This subroutine performs the following tasks:

- Ends the WEB page reporting.
- Ends the I/O processes.
- Ends the GPU processes.
- Ends the CPU children.
- Deallocates the ***FMS*** memory pool.
- Deletes the semaphores and locks used for synchronizing ***FMS*** tasks.
- Prints reports on subroutine and file usage and performance.

If FMSINI is the first statement in your application and FMSEND is the last, the non- ***FMS*** timing data printed by FMSEND will have more meaning.

Copyright © Multipath Corporation

C.7. FMS MEMORY MANAGEMENT

Index	NAME	Function
C.7.1	FMSIMG	Get Integer Memory
	FMSRMG	Get Real*8 Memory
	FMSCMG	Get Complex*16 Memory
C.7.2	FMSIMR	Return Integer Memory
	FMSRMR	Return Real*8 Memory
	FMSCMR	Return Complex*16 Memory
C.7.3	FMSILG	Get Local Integer Memory
	FMSRLG	Get Local Real*8 Memory
	FMSCLG	Get Local Complex*16 Memory
C.7.4	FMSILR	Return Local Integer Memory
	FMSRLR	Return Local Real*8 Memory
	FMSCLR	Return Local Complex*16 Memory

Copyright © Multipath Corporation

C.7.1. SUBROUTINES FMSIMG, FMSRMG, FMSCMG

PURPOSE

To allocate memory for data storage.

SYNOPSIS

```
CALL FMSIMG (IMD, LOC, LEN)
CALL FMSRMG (RMD, LOC, LEN)
CALL FMSCMG (CMD, LOC, LEN)
```

INPUT PARAMETERS

- IMD(*) = Integer array(FMSIMG)

  RMD(*) = Real array (FMSRMG)

  CMD(*) = Complex array (FMSCMG).

  This is the array you use in your program to reference the allocated memory. This array must be defined by a POINTER or aligned on a natural address boundary as explained below.

- LEN = Integer.

  The number of words of the appropriate data type to allocate.

OUTPUT PARAMETERS:

- LOC = Integer.

  Starting address of allocated memory in IMD, RMD or CMD, relative to the address you provided. If you have dimensioned IMD, RMD and CMD starting at 0 in your program, LOC will be the subscript of these arrays.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
MAXMD	Amount of memory to use
MDUSED	Amount of memory already used

SHOW	Display reports

DESCRIPTION:

These subroutines provide your program with access to the ***FMS*** memory manager.

The first step is to declare an array in your program that you will use to reference the allocated memory. There are currently two mechanisms used to do this.

1. Machines supporting FORTRAN pointers

   If your machine supports the FORTRAN POINTER statement, the reference arrays are obtained by including the POINTER statements for the arrays and obtaining the pointer values by calling FMSIGT, as follows:

```
POINTER (CMD_PTR, CMD)
POINTER (RMD_PTR, RMD)
POINTER (IMD_PTR, IMD)
COMPLEX*16 CMD(0:1)
REAL*8     RMD(0:1)
INTEGER    IMD(0:1)
...
CALL FMSIGT ('MEMPTR',CMD_PTR)
CALL FMSIGT ('MEMPTR',RMD_PTR)
CALL FMSIGT ('MEMPTR',IMD_PTR)
```

2. Machines not supporting FORTRAN pointers

   If your machine does not support the FORTRAN POINTER statement, the reference arrays are obtained as follows:

```
COMMON CMD
EQUIVALENCE (IMD, RMD, CMD)
COMPLEX*16 CMD(0:1)
REAL*8     RMD(0:1)
INTEGER    IMD(0:1)
```

   On most machines, ***FMS*** automatically aligns BLANK COMMON on a 16-byte boundary. You should not be concerned if you use BLANK COMMON for other purposes. ***FMS*** does not store any data in CMD(0:1). It simply uses the arrays IMD, RMD and CMD as reference points for the allocated memory.

Once the reference arrays are established in your subroutine, you call FMSIMG to allocate LEN integer words starting at IMD(LOC), FMSRMG to allocate LEN REAL*8 words starting at RMD(LOC), or FMSCMG to allocate LEN COMPLEX*16 words starting at CMD(LOC).

The value of LOC returned is relative to the array value you provide on the call. For example, if you make the call:

```
CALL FMSIMG (IMD(10), LOC, LEN)
```

the allocated memory will start at IMD(10+LOC).

You may make the value of LOC be an actual subscript of the array by dimensioning the array to start at 0,

```
INTEGER IMD (0:1)
```

Then when you call the memory allocation subroutine

```
CALL FMSIMG (IMD, LOC, LEN)
```

the allocated memory will start at IMD(LOC).

Examples

As an example, to allocate storage for the ***FMS*** profile vector LOWEQ(NUMEQ) call FMSIMG as follows:

```
CALL FMSIMG (IMD, LLOWEQ, NUMEQ)
```

You may address LOWEQ(I) as

```
IMD(LLOWEQ-1+I)
```

You may also pass the LOWEQ array to a subroutine as

```
CALL RSDI (IMD(LLOWEQ), NUMEQ, 'LUA', LUA)
```

When storage for LOWEQ(NUMEQ) is no longer required, you may return it to ***FMS***.

```
CALL FMSIMR (IMD, LLOWEQ, NUMEQ)
```

You may allocate storage for several arrays with a single call by requesting the total length. For example, to allocate storage for A(N), B(N) and C(N) use

```
CALL FMSRMG (RMD, LA, 3*N)
LB = LA + N
LC = LB + N
```

Copyright © Multipath Corporation

C.7.2. SUBROUTINES FMSIMR, FMSRMR, FMSCMR

PURPOSE

To return allocated memory to ***FMS***

SYNOPSIS

```
CALL FMSIMR (IMD, LOC, LEN)
CALL FMSRMR (RMD, LOC, LEN)
CALL FMSCMR (CMD, LOC, LEN)
```

INPUT PARAMETERS

- IMD(*) = Integer array(FMSIMR)

  RMD(*) = Real array (FMSRMR)

  CMD(*) = Complex array (FMSCMR).

  This is the reference array you use in your program to allocate the memory by calling FMSIMG, FMSRMG or FMSCMG.

- LOC = Integer.

  Starting address in IMD, RMD or CMD of memory to return.

- LEN = Integer.

  The number of words of the appropriate data type to return.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
MAXMD	Amount of memory to use
MDUSED	Amount of memory already used

SHOW	Display reports

DESCRIPTION:

Subroutine FMSIMR returns LEN integer words starting at IMD(LOC). Subroutine FMSRMR returns LEN REAL*8 words starting at RMD(LOC). Subroutine FMSCMR returns LEN COMPLEX*16 words starting at CMD(LOC). All memory returned to ***FMS*** must first have been allocated by calling FMSIMG, FMSRMG or FMSCMG.

Copyright © Multipath Corporation

C.7.3. SUBROUTINES FMSILG, FMSRLG, FMSCLG

PURPOSE

To allocate local memory on all the processors.

SYNOPSIS

```
CALL FMSILG (IMD, LOC, LEN)
CALL FMSRLG (RMD, LOC, LEN)
CALL FMSCLG (CMD, LOC, LEN)
```

INPUT PARAMETERS

- IMD(*) = Integer array(FMSILG)

  RMD(*) = Real array (FMSRLG)

  CMD(*) = Complex array (FMSCLG).

  This is the array you use in your program to reference the allocated memory. This array must be defined by a POINTER or aligned on a natural address boundary as explained below and in the README.TXT file for your machine.

- LEN = Integer.

  The number of words of the appropriate data type to allocate on each processor.

OUTPUT PARAMETERS:

- LOC = Integer.

  Starting address of allocated memory in IMD, RMD or CMD, relative to the address you provided. If you have dimensioned IMD, RMD and CMD starting at 0 in your program, LOC will be the subscript of these arrays. This is the address where the memory starts for the parent (CPU 0). The starting address for process ICPU is L = LOC + LEN*(ICPU-1).

- LEN = Number of words allocated per processor. The input value is rounded up to be of page length.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
MAXMD	Amount of memory to use
MAXCPU	The number of processors on which memory is allocated.
NUMAFL	Use NUMA directive
MDINIT	How memory is initialized
MDWHEN	When memory is allocated.
MDUSED	Amount of memory already used
SHOW	Display reports

DESCRIPTION:

These subroutines provide your program with access to the ***FMS*** memory manager.

The first step is to declare an array in your program that you will use to reference the allocated memory. There are currently two mechanisms used to do this.

1. Machines supporting FORTRAN pointers

   If your machine supports the FORTRAN POINTER statement, the reference arrays are obtained by including the POINTER statements for the arrays and obtaining the pointer values by calling FMSIGT, as follows:

```
POINTER (CMD_PTR, CMD)
POINTER (RMD_PTR, RMD)
POINTER (IMD_PTR, IMD)
COMPLEX*16 CMD(0:1)
REAL*8     RMD(0:1)
INTEGER    IMD(0:1)
...
CALL FMSIGT ('MEMPTR',CMD_PTR)
CALL FMSIGT ('MEMPTR',RMD_PTR)
CALL FMSIGT ('MEMPTR',IMD_PTR
```

2. Machines not supporting FORTRAN pointers

   If your machine does not support the FORTRAN POINTER statement, the reference arrays are obtained as follows:

```
COMMON CMD
EQUIVALENCE (IMD, RMD, CMD)
COMPLEX*16 CMD(0:1)
REAL*8     RMD(0:1)
```

```
      INTEGER    IMD(0:1)
```

On most machines, ***FMS*** automatically aligns BLANK COMMON on a 16-byte boundary. You should not be concerned if you use BLANK COMMON for other purposes. ***FMS*** does not store any data in CMD(0:1). It simply uses the arrays IMD, RMD and CMD as reference points for the allocated memory.

Once the reference arrays are established in your subroutine, you call FMSILG to allocate LEN integer words on each processor, starting at IMD(LOC) on processor 0 (Parent). Similarly you may call FMSRMG to allocate LEN REAL*8 words starting at RMD(LOC), or FMSCMG to allocate LEN COMPLEX*16 words starting at CMD(LOC). The value of LOC returned is relative to the array value you provide on the call. For example, if you make the call:

```
      CALL FMSILG (IMD(10), LOC, LEN)
```

the allocated memory will start at IMD(10+LOC).

You may make the value of LOC be an actual subscript of the array by dimensioning the array to start at 0,

```
      INTEGER IMD (0:1)
```

Then when you call the memory allocation subroutine

```
      CALL FMSILG (IMD, LOC, LEN)
```

the allocated memory will start at IMD(LOC).

Examples

As an example, to allocate a vector for each processor X(NUMEQ) call FMSRLG as follows:

```
      LEN = NUMEQ
      CALL FMSRLG (RMD, LCPU1, LEN)
```

For processor N (where N=0 is the parent) the allocated memory is addressed as

```
      L = LCPU1 + LEN*(N-1)
      X(I) = RMD(L-1+I)
```

The memory will be local to processor N.

These routines are similar to FMSIMG, FMSRMG, and FMSCMG with the following differences:

1. LEN is rounded up to be page length,
2. LEN words are allocated on each processor,
3. The allocated memory is placed locally next to the processor

Copyright © Multipath Corporation

C.7.4. SUBROUTINES FMSILR, FMSRLR, FMSCLR

PURPOSE

To return allocated local memory to ***FMS***

SYNOPSIS

```
CALL FMSILR (IMD, LOC, LEN)
CALL FMSRLR (RMD, LOC, LEN)
CALL FMSCLR (CMD, LOC, LEN)
```

INPUT PARAMETERS

- IMD(*) = Integer array(FMSILR)

  RMD(*) = Real array (FMSRLR)

  CMD(*) = Complex array (FMSCLR).

  This is the reference array you use in your program to allocate the memory by calling FMSILG, FMSRLG or FMSCLG.

- LOC = Integer.

  Starting address in IMD, RMD or CMD of memory to return. This is the value returned by FMSILG, FMSRLG or FMSCLG.

- LEN = Integer.

  The number of words per processor of the appropriate data type to return. This is the value returned by FMSILG, FMSRLG or FMSCLG which has been rounded up to page length.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description

MAXMD	Amount of memory to use
MAXCPU	Number of processors
MDUSED	Amount of memory already used
SHOW	Display reports

DESCRIPTION:

Subroutine FMSILR returns LEN integer words per processor, starting at IMD(LOC) on processor 0 (parent). Subroutine FMSRLR returns LEN REAL*8 words starting at RMD(LOC). Subroutine FMSCLR returns LEN COMPLEX*16 words starting at CMD(LOC). All memory returned to ***FMS*** must first have been allocated by calling FMSILG, FMSRLG or FMSCLG.

The total memory returned is MAXCPU * LEN words.

Copyright © Multipath Corporation

C.8. FMS PRINTING SUBROUTINES

Index	NAME	Function
C.8.1	FMSTIM	Obtain CPU and Elapsed Time
C.8.2	FMSPSH FMSPOP	Place Subroutine on Call Stack Remove Subroutine from Call Stack
C.8.3	FMSPRV	Print Vectors from Vector File
C.8.4	FMSPRF	Print Matrix from Memory
C.8.5	FMSPRM	Print Matrix from Matrix File

Copyright © Multipath Corporation

C.8.1. SUBROUTINE FMSTIM

PURPOSE

To obtain the CPU time, the elapsed (wall) time and the time spent waiting for I/O to complete since FMSINI was called. The CPU time includes the parent and all children.

SYNOPSIS

```
CALL FMSTIM (TCPU, TWALL, TIO)
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS:

- TCPU = Real

  CPU time in seconds used by the parent and all children since FMSINI was called.

- TWALL = Real

  Elapsed (wall) time in seconds used since FMSINI was called.

- TIO = Real

  Time spent waiting for I/O to complete, in seconds since FMSINI was called.

DESCRIPTION:

This subroutine returns the CPU, elapsed and I/O wait times since FMSINI was called. You may use it to obtain and print timing information within your application.

EXAMPLE:

To compute the time required by your routine FOO,

```
      REAL*8 TCPU,  TCPU1,  TCPU2
      REAL*8 TWALL, TWALL1, TWALL2
      REAL*8 TIO,   TIO1,   TIO2
      REAL*8 PSPEED
      ...
      CALL FMSINI
```

```
      ...
      CALL FMSTIM (TCPU1, TWALL1, TIO1)
      CALL FOO
      CALL FMSTIM (TCPU2, TWALL2, TIO2)
      TCPU   = TCPU2  - TCPU1
      TWALL  = TWALL2 - TWALL1
      TIO    = TIO2   - TIO1
      PSPEED = TCPU/(TWALL - TIO)
      PRINT *,'CPU  time used by FOO........(Sec.)=', TCPU
      PRINT *,'Wall time used by FOO........(Sec.)=', TWALL
      PRINT *,'Time spent waiting for I/O...(Sec.)=', TIO
      PRINT *,'Parallel speedup in FOO.............=', PSPEED
```

Copyright © Multipath Corporation

C.8.2. SUBROUTINES FMSPSH AND FMSPOP

PURPOSE

To trace calls to your subroutines. FMSPSH adds your subroutine to the list of internal subroutines maintained by ***FMS***. FMSPOP removes your subroutine from the list of called subroutines. The benefits include CPU and WALL timings, number of times called by parent and children and an optional graphical trace of your subroutine calls.

SYNOPSIS

```
CALL FMSPSH (NAME)
CALL FMSPOP (NAME)
```

INPUT PARAMETERS

- NAME = CHARACTER*18
  Name of the subroutine

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
LOGTIM	Turns on timing and tracing

DESCRIPTION:

These subroutines add or remove an entry to the ***FMS*** call stack for subroutine NAME. The action taken is different when NAME is called by the parent or child.

- Parent
  FMSPSH places subroutine NAME on the ***FMS*** call stack. Timers are started to measure the CPU and WALL time used. The number of calls by the parent is incremented.
  FMSPOP removes subroutine NAME from the ***FMS*** call stack and stops the timers.;

- Child

FMSPSH increments the number of times NAME was called by a child process.

FMSPOP checks NAME and returns.

You MUST follow a call to FMSPSH with a call to FMSPOP at each point in subroutine NAME where a return is possible.

Copyright © Multipath Corporation

C.8.3. SUBROUTINE FMSPRV

PURPOSE

To print one or more vectors from a ***FMS*** vector file.

SYNOPSIS

```
CALL FMSPRV (LUX, NUMV1, NUMVEC, NMSG)
```

INPUT PARAMETERS

- LUX(25) = Integer array.

Attribute list for the vector file.

- NUMV1 = Integer.

First vector (record) to print.

- NUMVEC = Integer.

Number of vectors (records) to print.

- NMSG = Integer.

Heading message number.

```
= 1, 'VECTOR ( ) BEFORE FORWARD REDUCTION.'
= 2, 'VECTOR ( ) AFTER FORWARD REDUCTION.'
= 3, 'VECTOR ( ) AFTER DIAGONAL SCALING.'
= 4, 'VECTOR ( ) AFTER BACK SUBSTITUTION.'
= 5, 'VECTOR ( ) BEFORE MATRIX MULTIPLY.'
= 6, 'VECTOR ( ) AFTER DIAGONAL MULTIPLICATION.'
= 7, 'VECTOR ( ) AFTER MATRIX MULTIPLICATION.'
= 8, 'VECTOR ( 1) MATRIX DIAGONAL.'
= 9, 'VECTOR ( 1) DIAGONAL FACTOR RECIPROCALS.'
= 10, 'VECTOR ( ).'
= 11, 'VECTOR ( ),X-VECTORS.'
= 12, 'VECTOR ( ),Y-VECTORS.'
```

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
LUPR	Output unit number
NEQPR1	Lower limit on equation to print
NEQPR2	Upper limit on equation to print
NROWPG	Number of rows per page
NCOLPG	Number of columns per page
IFMAT	Format for printing integers
RFMAT	Format for printing real numbers
CFMAT	Format for printing complex numbers

DESCRIPTION:

This subroutine prints the specified vectors from file LUX(1). The ***FMS*** parameters NEQPR1 and NEQPR2 may be used to set lower and upper limits on the range of equations printed.

Copyright © Multipath Corporation

C.8.4. SUBROUTINE FMSPRF

PURPOSE

To print a full matrix in memory.

SYNOPSIS

```
CALL FMSPRF (A, NROWS, NCOLS, ISTYPE, IDTYPE)
```

INPUT PARAMETERS

- A(NROWS*(NROWS+1)/2) = Real array (ISTYPE=1, IDTYPE=1).

  A(NROWS*(NROWS+1)/2) = Complex array (ISTYPE=1, IDTYPE=2).

  A(NROWS,NCOLS) = Real array (ISTYPE=2, IDTYPE=1).

  A(NROWS,NCOLS) = Complex array (ISTYPE=2, IDTYPE=2).

  When ISTYPE=1, this is the lower triangular matrix stored by rows from column 1 through the diagonal.

  When ISTYPE=2, this is a full matrix in normal FORTRAN storage.

- NROWS = Integer.

  Number of rows in the matrix.

- NCOLS = Integer.

  Number of columns in the matrix.

  When ISTYPE=1, this must be the same value as NROWS.

- ISTYPE = Integer.

  Symmetry and data structure type.

  = 1, Symmetric matrix stored in lower triangular format.

  = 2, Full matrix stored in normal FORTRAN storage.

- IDTYPE = Integer.

  Data type.

  = 1, Real.

  = 2, Complex.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
LUPR	Output unit number
NEQPR1	Lower limit on equation to print
NEQPR2	Upper limit on equation to print
NROWPG	Number of rows per page
NCOLPG	Number of columns per page
IFMAT	Format for printing integers
RFMAT	Format for printing real numbers
CFMAT	Format for printing complex numbers

DESCRIPTION:

This subroutine prints a full matrix that is stored in memory. The matrix may be in lower triangular format (ISTYPE=1) or a full matrix (ISTYPE=2).

The ***FMS*** parameters NEQPR1 and NEQPR2 may be used to set lower and upper limits on the range of equations printed. The effect of these parameters on printing the full matrix **[A]** is to print the following submatrix:

```
A(NEQPR1:NEQPR2,NEQPR1:NEQPR2)
```

Copyright © Multipath Corporation

C.8.5. SUBROUTINE FMSPRM

PURPOSE

To print a matrix stored in ***FMS*** format.

SYNOPSIS

```
CALL FMSPRM (LUA, NMSG)
```

INPUT PARAMETERS

- LUA(25) = Integer array.

  File attribute list for the matrix.

- NMSG = Integer.

  Heading message number.

```
= 1,
'LOWER TRIANGLE MATRIX BEFORE FACTORING.'
'MATRIX DIAGONAL.'
'UPPER TRIANGLE MATRIX BEFORE FACTORING.'
= 2,
'LOWER TRIANGLE MATRIX AFTER FACTORING.'
'DIAGONAL FACTOR RECIPROCALS.'
'UPPER TRIANGLE MATRIX AFTER FACTORING.'
```

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
LUPR	Output unit number
NEQPR1	Lower limit on equation to print

NEQPR2	Upper limit on equation to print
NROWPG	Number of rows per page
NCOLPG	Number of columns per page
IFMAT	Format for printing integers
RFMAT	Format for printing real numbers
CFMAT	Format for printing complex numbers

DESCRIPTION:

This subroutine prints the lower triangular, diagonal, and upper triangular (nonsymmetric) parts of a matrix. Terms that fall outside the matrix profile are printed as blanks. This subroutine may be used to print the original matrix **`[A]`** or the matrix factors **`[L]`**, **`[1/D]`**, and **`[U]`**. For symmetric matrices, no upper triangle data is printed.

The ***FMS*** parameters NEQPR1 and NEQPR2 may be used to set lower and upper limits on the range of equations printed. For the lower triangular matrix, the range is on rows. For the upper triangular matrix, the range is on columns. Only the diagonals between NEQPR1 and NEQPR2 are printed.

Copyright © Multipath Corporation

C.9. FMS PARALLEL PROCESSING SUBROUTINES

Index	NAME	Function
C.9.1	FMSPAR	Queue Parallel Tasks
	FMSRUN	Start Parallel Tasks Running
	FMSYNC	Wait for Parallel Tasks to Complete
C.9.2	FMSONE	Acquire Mutual Exclusive Lock
	FMSALL	Release Mutual Exclusive Lock

Copyright © Multipath Corporation

C.9.1. SUBROUTINES FMSPAR, FMSRUN, FMSYNC

PURPOSE

To perform parallel processing

SYNOPSIS

```
CALL FMSPAR (NUMPAR, SNAME, P1, P2, ...)
CALL FMSRUN
CALL FMSYNC
```

INPUT PARAMETERS

- NUMPAR = Integer

  (FMSPAR only) Specifies the number of parameters to be passed to SNAME.

  Maximum value is 16.

- SNAME = Address of entry point

  (FMSPAR only) Specifies the subroutine to be called in parallel.

  This subroutine must be declared EXTERNAL in your program.

- P1, P2,... = Integer, Real or Complex

  (FMSPAR only) Parameters to be passed to SNAME. There must be NUMPAR parameters specified. These parameters must be address pointers to shared memory.

OUTPUT PARAMETERS:

None.

FMS PARAMETERS:

The following ***FMS*** Parameters are especially important to this routine:

Parameter	Description
MAXCPU	Number of processors to use.
MYNODE	Your process number.

DESCRIPTION:

These subroutines provide a way for you to run part of your application in parallel. To use this feature, you place the work to be performed into a subroutine, which we will call USERSUB for this example. This subroutine should be designed to be called once for each processor, with each call performing an equal piece of work.
For example, we will assume subroutine USERSUB has 3 arguments, A1, A2 and A3. You use the ***FMS*** parallel processing utilities to call USERSUB in parallel as follows:

```
EXTERNAL USERSUB
...
CALL FMSIGT ('MAXCPU', MAXCPU)
DO MYCPU = 2,MAXCPU
   CALL FMSPAR (3, USERSUB, A1, A2, A3)
ENDDO
IF(MAXCPU .GT. 1) CALL FMSRUN
CALL USERSUB (A1, A2, A3)
IF(MAXCPU .GT. 1) CALL FMSYNC
```

The EXTERNAL statement lets the compiler know that USERSUB is a subroutine name (global symbol) and not a local variable. You must declare all subroutines passed to FMSPAR as EXTERNAL.
You may call FMSIGT to obtain MAXCPU, the number of processors actually running.

You then call FMSPAR to place the work for the children in a queue. After all of the work has been queued, you call FMSRUN to start the children running.

After the children are running, USERSUB is called by the parent to do its share of the work. The parent then calls FMSYNC to wait for the completion of the children.

Note that if there is only one process (MAXCPU=1), FMSPAR , FMSRUN and FMSYNC are not called.

In subroutine USERSUB you may include the statement

```
CALL FMSIGT('MYNODE', MYNODE)
```

to obtain the process number (0 to MAXCPU-1; 0 is parent) of the subroutine. You may then use the value of MYNODE to determine your piece of work to compute.

Copyright © Multipath Corporation

C.9.2. SUBROUTINES FMSONE AND FMSALL

PURPOSE

To acquire and release a mutual exclusive lock.

SYNOPSIS

```
CALL FMSONE
CALL FMSALL
```

INPUT PARAMETERS

None.

OUTPUT PARAMETERS:

None.

DESCRIPTION:

For some applications, the children or parent may need to obtain exclusive use of the data for a short period of time. For example, if the calls to SNAME all passed the same parameters (P1, P2, ...) then the individual copies of SNAME will need to determine what work to do. They might do this by incrementing a shared variable NTASKS to determine their task, MYTASK.

FMS includes a mutual exclusive lock that can be used as follows:

```
SUBROUTINE SNAME(P1, P2, ...,NTASKS, MAXTSK)
INTEGER MYTASK, MAXTSK
...
CALL FMSONE
   MYTASK = NTASKS
   NTASKS = NTASKS + 1
CALL FMSALL
IF(MYTASK .GT. MAXTSK) RETURN
```

The parent initializes NTASKS to 1. Each child then obtains a unique value of MYTASK that can be use to perform its part of the work. The ***FMS*** mutual exclusive lock FMSONE prevents two tasks from reading and incrementing NTASKS at the same time.

If the number of tasks to be performed by subroutine SNAME far exceeds MAXCPU, this method can provide improved load balancing, compared to pre-allocating the work before each call to SNAME. As each task finishes its piece of work, it checks out another piece by incrementing NTASKS. When there is no more work left, it returns.

Copyright © Multipath Corporation

D. FMS PARAMETERS

- D1. Hardware Parameters
- D2. GPU Parameters
- D3. NUMA Parameters
- D4. File Parameters
- D5. Matrix File Initialization Parameters
- D6. Matrix Assembly Parameters
- D7. Matrix Factoring Parameters
- D8. Solving Parameters
- D9. Multiply Parameters
- D10. Printing Parameters
- D11. **Dashboard** Performance Report Parameters
- D12. Error Handling Parameters
- D13. Application Program Parameters

Copyright © Multipath Corporation

D.1. FMS HARDWARE PARAMETERS

Hardware Parameters

Index	NAME	Function
D.1.1	MAXCPU	Number of CPU processing cores to use
D.1.2	MYNODE	Your process number (0 to MAXCPU-1) (Read Only)
D.1.3	MDHW	Amount of memory installed (8-byte words)(Read Only)
D.1.4	MAXMD MAXMB MEMPCT	Amount of memory to use (8-byte words) Amount of memory to use (Megabytes) Amount of memory to use (% of MDHW)
D.1.5	MDUSED	Amount of memory already used (8-byte words)(Read Only)
D.1.6	MEMPTR	Pointer to ***FMS*** managed memory (Read Only)
D.1.7	MDHOW	How memory is allocated
D.1.8	MDINIT	Memory initialization options
D.1.9	MALIGN	Memory alignment (8-byte words)
D.1.10	LPAGE	Page size (bytes)
D.1.11	NOPAGE	Page file location
D.1.12	ISYNC	Type of CPU synchronization
D.1.13	ISCOPE	Scope and priority of threads
D.1.14	MDPADD	Append prefetch area to memory allocations (8-byte words)
D.1.15	LCACHE	Size of cache (bytes)
D.1.16	LSTACK	Size of stack (bytes)
D.1.17	ITPCPU	Processor type (1=scalar, 2=vector)
D.1.18	LENVRG	Vector register length (8-byte words)
D.1.19	NOCORE	Do NOT use the specified CPU core

Copyright © Multipath Corporation

D.1.1. FMS HARDWARE PARAMETER MAXCPU

Data Type

Integer

Default Value

Number of reported CPU cores.

Description

This is the number of compute tasks that ***FMS*** runs in parallel. By default, this value is set to the number of CPU cores in the system. This allows a single ***FMS*** job to use all system resources and complete as fast as possible.

The value of MAXCPU is loosely coupled to the number of CPU cores in the system. MAXCPU controls the number of tasks that ***FMS*** places in the queue for execution. This may be greater or less than the number of CPU cores.

If you run multiple ***FMS*** jobs simultaneously, you may want to decrease the default value of MAXCPU to reduce contention for CPU cores. What counts is the sum of the values of MAXCPU in all concurrent ***FMS*** jobs. If this value is less than the number of CPU cores, the system is under-subscribed. If this value is greater than the number of CPU cores, the system is oversubscribed.

By default most desktop systems (PC's, Workstations) have hyperthreading turned on. This causes the system to state more CPU cores than are physically present. The intention is to allow scheduling of multiple tasks for each CPU core, thereby increasing overall throughput for random work loads. Hyperthreading offers no performance benefit for ***FMS***, because each thread is designed to use 100% of the CPU core resources. Scheduling multiple ***FMS*** threads on a single CPU core actually decreases performance as the threads compete for shared resources. Hyperthreading can be turned off, which is usually an option in the BIOS. If this is not possible, setting MAXCPU to the number of physical CPU cores will provide most of the benefit.

Setting MAXCPU to a value greater than the number of CPU cores will always degrade performance.

Copyright © Multipath Corporation

D.1.2. FMS HARDWARE PARAMETER MYNODE

Data Type

Integer

Default Value

None. This is a read-only parameter.

Description

This parameter returns the ***FMS*** process number of the calling process. Process number 0 is the parent. The children are numbered from 1 through (MAXCPU-1).

Typically the call

```
CALL FMSIGT ('MYNODE',MYNODE)
```

is placed in a subroutine which is running in parallel (started by a call to FMSPAR). The integer variable MYNODE should be a local variable (not in a common block where it will be shared).

This value may be used to identify the process or determine which piece of work it is to perform.

Copyright © Multipath Corporation

D.1.3. FMS HARDWARE PARAMETER MDHW

Data Type

Integer

Units

REAL*8 (8-byte) words

Default Value

The amount of memory installed in the system,

or

the amount of memory available to each user.

Description

This parameter returns the amount of memory installed in the system. It is a read-only parameter.

Copyright © Multipath Corporation

D.1.4. FMS HARDWARE PARAMETERS MAXMD, MAXMB AND MEMPCT

Data Type

Integer

Units

MAXMB: REAL*8 (8-byte) words
MAXMB: Megabytes
MEMPCT: Percent

Default Value

MEMPCT:
50 on machines where MDHW is the amount of physical memory
90 on machines where MDHW is the amount of memory for each user
MAXMD: 0.01 * MEMPCT * MDHW

Description

These parameters control the amount of memory used by ***FMS***. You may specify the amount of memory in any of the following ways:

1. MAXMD specifies the memory size in REAL*8 words (8-bytes)
2. MAXMB specifies the memory size in Megabytes,
3. MEMPCT specifies the memory size as a percentage of the installed hardware MDHW.

These Parameters are usually specified in the license file. However they may also be specified during program execution by calling FMSIST.

FMS includes a memory management system for scratch storage of work arrays. When you call FMSINI, a pool of memory is allocated. Individual ***FMS*** tasks check memory out of this pool as needed. When the memory is no longer required, it is returned back to the pool.

All ***FMS*** routines which use scratch memory check the memory pool prior to execution to determine the amount of memory available. They then adjust their algorithm and storage requirements to operate within the available space.

Your application may also use memory from the ***FMS*** memory pool by calling routines FMSIMG, FMSRMG or

FMSCMG. When you no longer need the memory you may return it to the pool by calling FMSIMR, FMSRMR and FMSCMR. This allows the non-***FMS*** part of your application to share the same memory used by ***FMS***.

FMS is designed to operate over a wide range of MAXMD values. In general, increasing MAXMD will reduce the amount of data ***FMS*** rereads from disk. If you notice that ***FMS*** is spending excessive time waiting for I/O to complete, you should try increasing the value of MAXMD. However MAXMD should not be increased beyond MDHW (the amount of physical memory in the machine), or excessive page faults will result.

On machines without virtual memory there is a hard upper limit to the value of MAXMD. Trying to set MAXMD above this limit will result in an ***FMS*** system error.

On virtual memory machines it is possible to set MAXMD to a value that is larger than the amount of physical memory. However this will result in page faults and degrade performance.

If you run multiple ***FMS*** jobs simultaneously, you may want to decrease the default value of MAXMD to reduce contention for the memory. What counts is the sum of the values of MAXMD in all concurrent ***FMS*** jobs. If this value is less than the amount of physical memory, the memory is under-subscribed. If this value is greater than the amount of physical memory, the system is oversubscribed.

Some oversubscription of memory will usually increase throughput, especially if the non-***FMS*** part of your application has not been converted to run in parallel. A few tests will usually show the best combination of MAXMD for the concurrent ***FMS*** jobs. Note that each individual ***FMS*** job can have a different value of MAXMD.

Copyright © Multipath Corporation

D.1.5. FMS HARDWARE PARAMETER MDUSED

Data Type

Integer

Units

REAL*8 (8-byte) words

Default Value

None. This is a read-only parameter.

Description

This parameter returns the amount of memory currently allocated in the ***FMS*** memory pool. The units of MDUSED are REAL*8 words (8-bytes).

You may use this parameter to determine the amount of memory left in the pool. For example:

```
CALL FMSIMG ('MAXMD' , MAXMD )
CALL FMSIMG ('MDUSED', MDUSED)
MDLEFT = MAXMD - MDUSED
```

will compute the amount of memory left in the pool, MDLEFT.

If you try to allocate MDLEFT words of memory and fail, the ***FMS*** memory pool may have become fragmented. You may correct this condition by altering the order that you allocate and free memory from the pool. The longest arrays should be allocated first, and so on. Try using the call

```
CALL FMSCST ('SHOW', 'MEMORY')
```

to display the current contents of the ***FMS*** memory pool.

Copyright © Multipath Corporation

D.1.6. FMS HARDWARE PARAMETER MEMPTR

Data Type

Integer

Default Value

None. This is a read-only parameter.

Description

This parameter returns the pointer which defines the base address for arrays in your application used to reference ***FMS*** managed memory. The first step is to declare the following arrays in your application:

```
POINTER (CMD_PTR, CMD)
POINTER (RMD_PTR, RMD)
POINTER (IMD_PTR, IMD)
COMPLEX*16 CMD(0:1)
REAL*8     RMD(0:1)
INTEGER    IMD(0:1)
```

At this point the arrays IMD, RMD and CMD exist in your application, but they start at address zero.

The next step is to map these arrays directly to ***FMS*** managed memory using the following calls:

```
CALL FMSIGT ('MEMPTR', CMD_PTR)
CALL FMSIGT ('MEMPTR', RMD_PTR)
CALL FMSIGT ('MEMPTR', IMD_PTR)
```

The arrays IMD, RMD and CMD now share the same addresses with the ***FMS*** managed memory. Note that all 3 arrays share the same address space.

Before you can use the memory allocated by ***FMS***, you must allocate it from the ***FMS*** memory pool. The routines FMSIMG, FMSRMG, FMSCMG perform that task. When the memory is no longer required, you may return it to the pool by calling FMSIMR, FMSRMR, FMSCMR.

Copyright © Multipath Corporation

D.1.7. FMS HARDWARE PARAMETER MDHOW

Data Type

Integer

Default Value

Usually 0, but may be different on some machines.

Description

This parameter determines the system service used for allocating ***FMS*** memory. Not all options are available on all machines. The following options are available:

- 0, mmap

This is the preferred method and is used when available. It provides for a paging file in case the pages are faulted. This page file may be the system page file (NPOAGE=-1) or a private page file (NPOAGE=0).

- 1, malloc

Where mmap is not available, this service is used. The amount of memory available may be limited.

- 2, shm

On some systems, this service can provide special features, including large page sizes. Usually the amount of shared memory available must be created prior to the run. Memory allocated using shm is usually not swapped to disk.

- 3, cudaHostAlloc

Provides for page locked memory that cannot be swapped to disk. The cuda runtime library takes advantage of this to do DMA transfers to the cuda devices, resulting in significantly improved transfer rates. Applicable only to systems having supported cuda devices and the cuda runtime library.
You may need to increase the memlock limit, which determines amount of memory a user can pin. To find the limit in Kbytes on Unix systems enter the following command (note the argument is lower case L):

```
ulimit -l
```

You may change this limit (may need to be SU) by

```
ulimit -l unlimited
```

or by editing the /etc/security/limits.conf file.

- 4, fixed array of 5 Mbytes

This option is mostly used for testing or debugging.

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.8. FMS HARDWARE PARAMETER MDINIT

Data Type

Integer

Default Value

0, except when NUMA is used.

4, when NUMA is used. NUMAFL > 0

Description

This parameter provides the following options for initializing the ***FMS*** memory. The options are bit flags. You may choose any combination. The following options are available:

- 0, Allocate only

Memory is allocated but not touched, placed, or set to any value.

- +1, Set to 0

Memory values are initialized to 0.

- +2, Touched

Each page of memory is touched but the contents are not altered. On NUMA systems, each page is touched by the processors on the local node.

- +4, Placed

Memory is placed on the nodes using NUMA directives.

- +8, Locked

Memory is locked and cannot be swapped. If you fail using this option you may need to increase your resource limits. To check the current value of limits use one of the commands:

```
ulimit -a     (bash)
limit         (csh)
```

The value you are interested in is "max locked memory" (bash) or "memorylocked" (csh). If the limits are too low, you can set them to unlimited by editing the file /etc/security/limits.conf as SU by adding the following 2 lines:

```
*    soft memlock unlimited
*    hard memlock unlimited
```

Because the memory in the ***FMS*** memory pool is treated as scratch memory, it should not be necessary to initialize its value to zero. However, if you encounter problems with your use of memory from the ***FMS*** pool, you can try setting MDINIT to 1.

When MDINIT is not zero, the memory is touched during FMSINI. From the system point of view, this involves setting the page table entries for each page in the memory pool. This may take some time, especially on large memory systems. The time required for this task also depends on the state of the machine. On a freshly booted system, where the page table entries are mostly zero, the process proceeds more rapidly than on a system that has been running, where the old page contents must first be flushed to disk.

If you are doing benchmarking and want to obtain consistent measurements, you should set MDINIT to 1 or 2. This will force the page table into a known state before computation begins. This avoids delays in setting page table entries during computation, which may vary from run to run.

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.9. FMS HARDWARE PARAMETER MALIGN

Data Type

Integer

Units

REAL*8 (8-byte) words

Default Value

One page, in units of REAL*8 (8-byte) words.

Description

When memory is allocated from the ***FMS*** memory pool, the starting address and length are rounded up to fall on boundaries specified by MALIGN. The default value of one page results in all memory allocated from the memory pool starting on a page boundary and being page length. ***FMS*** may change this value internally, if other alignment is required. Direct I/O is one option that may result in this value being increased.

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.10. FMS HARDWARE PARAMETER LPAGE

Data Type

Integer

Units

Bytes

Default Value

Hardware page size.

Description

On some systems, it is possible to increase the page size for allocated memory. This has the benefit of increasing the range of addresses stored in the translation look aside buffer (TLB). This reduces the overhead of the virtual memory system. On SGI, the following options are available:

LPAGE	Page Size
16,384	16K(default)
65,536	64K
262,144	256K
1,048,576	1M
4,194,304	4M
16,777,216	16M

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.11. FMS HARDWARE PARAMETER NOPAGE

Data Type

Integer

Default Value

-1

Description

The following options are available:

- -1, Use the system page file.
- 0, Use a page file created by ***FMS***.
FMS opens its own page file in the directory pointed to by FMSPAGE. This option is not recommended. Most UNIX systems will attempt to keep this page file consistent with the ***FMS*** memory, resulting in excessive page file activity.
- 1, Virtual to Real memory (VMS only)
Use virtual to real memory (no page file required). Applies to Digital systems running VMS only. The memory is locked in the system and cannot be paged. Physical and virtual addresses are the same.
- 2, Option 1 if it fits, otherwise Option 0.
Use virtual to real memory if it fits (Option 1). If it does not fit or the virtual to real memory is busy, use the ***FMS*** page file (Option 0).

On virtual memory machines, memory must be backed up by a page file. Because ***FMS*** may use more memory than typical applications, an option is provided to have ***FMS*** create its own page file.

When using the system page file, you do not need to create a directory for the ***FMS*** page file or define the environment variable FMSPAGE which points to it.

If you want ***FMS*** to create a private page file, you must create a directory FMSPAGE where the page file will be placed. In addition you must define the environment variable FMSPAGE to point to that directory. ***FMS*** automatically generates unique file names for the page files in directory FMSPAGE. When a job terminates normally, its page file is deleted.

VMS systems contain a range of addresses which are ignored by the virtual memory system. Data addressed in this range provide improved access performance for both processors and I/O systems. You may use these addresses by specifying NOPAGE=1.

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.12. FMS HARDWARE PARAMETER ISYNC

Data Type

Integer

Default Value

0

Description

This parameter controls how ***FMS*** compute tasks are synchronized. The following options are available:

- 0, Semaphores

When a task completes, it is put to "sleep", releasing its processor back to the system. This processor then becomes available to service ***FMS*** I/O tasks or other jobs. This value of ISYNC results in the maximum throughput of the system.

- 1, Spin-locks

When a task completes, it goes into a polling loop waiting for more work to do. Because the operating system cannot distinguish between polling and useful work, the task continues to tie up the processor. On some systems this can produce a deadlock condition if the number of tasks (***FMS*** or other jobs) exceeds the number of processors. For this reason, this option is not implemented on all systems. Typically this feature is only used on special hardware dedicated to a single ***FMS*** application.

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.13. FMS HARDWARE PARAMETER ISCOPE

Data Type

Integer

Default Value

0

Description

This parameter is used to adjust the scheduling and priority of the threads. The values are platform-specific. Generally you should not adjust these values. The values are bit flags. You may select any appropriate combination. The following options are available:

- 0, all threads same priority
  All threads have the same priority, run in process scope, are scheduled by SCHED_OTHER and are JOINABLE.
- +1 Decrease CPU thread priority (SGI)
  The priority of the CPU children is decreased by 1. The objective is to run them at a lower priority than the I/O threads.
- +2 Increase I/O thread priority (SGI)
  Run the I/O threads at a real time priority of 41. This assures that they are scheduled whenever an I/O operation completes. This requires limited superuser privileges.
- 1 to 63 Set I/O thread priority (Except SGI)
  Increase the priority of the I/O threads by the value specified.
- +64 Set thread scope to system (Except SGI)
  Run the threads in system scope. (Default is process scope).
- +128 Schedule with SCHED_FIFO (Except SGI)
  Use SCHED_FIFO for scheduling. (Default is SCHED_OTHER).
- +256 Schedule with SCHED_RR (Except SGI)
  Use SCHED_RR for scheduling. (Default is SCHED_OTHER).
- +512 Run threads DETACHED (Except SGI)
  Run threads DETACHED. (Default is JOINABLE).

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.14. FMS HARDWARE PARAMETER MDPADD

Data Type

Integer

Units

REAL*8 (8-byte) words

Default Value

0 (Except Compaq and SGI)
1023 (Compaq: one page)
400000 (SGI; 200K complex equations)

Description

This parameter appends a guard area to each memory allocation. It is used to accommodate special matrix multiply routines, which use aggressive prefetch. The value is in units of REAL*8 (8-byte) word. For SGI, the value should be as large as the largest array processed. For ***FMS***, this usually is the size of the R.H.S. or solution vector. The default value of 400000 provides for complex problems having 200,000 equations.

For Compaq, the default value of one page will work for all cases.

Because this parameter affects actions during FMSINI, it must be set in the License File.

Copyright © Multipath Corporation

D.1.15. FMS HARDWARE PARAMETER LCACHE

Data Type

Integer

Units

Bytes

Default Value

System dependent

Description

FMS adjusts its order of processing data to maximize the reuse of data, once it has been loaded into cache. It is initialized to the appropriate value of the hardware. You should not need to adjust this value.

Copyright © Multipath Corporation

D.1.16. FMS HARDWARE PARAMETER LSTACK

Data Type

Integer

Units

Bytes

Default Value

System dependent

Description

Threads allocate private storage dynamically when they are executed from a memory area called the stack. This includes any local variables or arrays dimensioned in the program. It does not include static arrays that appear in common blocks or the memory managed by ***FMS***. If you exceed the stack size, unpredictable results occur.

FMS sets the default value during the call to FMSINI. Generally this is the value specified in the shell limit, or the amount required by ***FMS***.

If you are using the ***FMS*** parallel processing utilities FMSPAR to run your subroutines in parallel, than the amount of stack size required may need to be increased, depending on the amount of local storage required by your application.

Because the maximum stack size is set during thread creation, this parameter must be set in the License File.

Copyright © Multipath Corporation

D.1.17. FMS HARDWARE PARAMETER ITPCPU

Data Type

Integer

Default Value

Machine dependent

Description

This Parameter sets the type of processor according to the following:

- 1, Scalar
- 2, Vector

Copyright © Multipath Corporation

D.1.18. FMS HARDWARE PARAMETER LENVRG

Data Type

Integer

Units

REAL*8 (8-byte) words

Default Value

System dependent

Description

This Parameter sets the length of the vector register on vector machines. It is only used when ITPCPU is 2. This value is used for selecting matrix block sizes that maximize the use of vector registers.

Copyright © Multipath Corporation

D.1.19. FMS CPU PARAMETER NOCORE

Data Type

Integer

Default Value

Use all the COREs

Description

Each use of this Parameter masks out the use one CPU core. Values range from 0 to the maximum number of processors used by ***FMS*** - 1. MAXCPU. You may use this option multiple times to mask out several CPU cores. It is mostly used for testing.

The default value uses all the CPU coress.

Examples

```
To NOT use cores 1 and 2,

NOCORE=1
NOCORE=2
```

Copyright © Multipath Corporation

D.2. FMS GPU PARAMETERS

GPU Parameters

Index	NAME	Function
D.2.1	GPUFL	Master flag to control what calculations are performed on the GPU
D.2.2	GPUPR	Print output level
D.2.3	CUPATH	Path to nvidia-smi program
D.2.4	MAXGPU	Number of GPU processors to use
D.2.5	GPUCPU	Bind GPU thread to CPU cores
D.2.6	NOGPU	Do not use specified GPU
D.2.7	IMMPCT ITRPCT ITMPCT CSAFE	Percent of work done by CPUs
D.2.8	GPUPCT	Percent of GPU free memory to use
D.2.9	ICHUNK ITRCHK ITMCHK	Transfer block size
D.2.10	CGTEST CGMPCT CGLOOP	Test CPU to GPU transfer rates
D.2.11	MNCORE	Only use GPUs with MNCORE cores
D.2.12	MAXEVT	Maximum recorded events on a call (read-only)
D.2.13	GPUTHI GPUTLO GPUTWS	GPU temperature to stop processing for cooling

Copyright © Multipath Corporation

D.2.1. FMS GPU PARAMETER GPUFL

Data Type

Integer

Default Value

0

Description

This is the master flag for activating the GPU processors. If you want to change the default values, you must set this parameter in the ***FMS*** License File. By default, GPU devices are used for all appropriate calculations if they are present. This parameter is mainly included for software development and testing.

The following values are bit flags. To obtain the value for GPUFL, add the selected options. The following options are available:

> The following flags control the use of GPUs during computation. By default, the GPUs are used. Setting these flags results in the computation being performed by CPUs only.
>
> - +1, Do NOT use the GPUs for performing matrix multiplies.
> - +2, Do NOT use the GPUs for performing triangle solves.
> - +4, Do NOT use the GPUs for diagonal block factoring.
>
> The following flags control how the GPUs are interfaced to the system.
>
> - +8, Do NOT use asynchronous streams for transfers to/from the GPUs. Asynchronous streams overlap transfers to and from the GPUs with GPU processing. Selecting this option results in these transfers being performed synchronously.
> - +16, Allocate GPU memory on each call (inefficient). The default is to allocate the memory once when the GPU threads are started and then use memory from this pool (more efficient).
>
> The following are flags passed to cudaSetDeviceFlags when each GPU is initialized:
>
> - +32; cudaDeviceMapHost
> The default is to map pinned host memory for access by the device. Setting this flag causes the memory not to be mapped.
> - +64; cudaDeviceLmemResizeToMax

The default is to instruct CUDA to not reduce local memory after resizing local memory for a kernel. This can prevent thrashing by local memory allocations when launching many kernels with high local memory usage at the cost of potentially increased memory usage. Setting this flag causes the memory to be reduced.

- +128; cudaDeviceScheduleYield
  Yield the CPU processors after queuing GPU work. On systems with powerful CPU processors, this might provide a net improvement in performance at the expense of having the GPUs wait. The default is to yield if there are less CPU processors than GPUs. If there are more CPU processors than GPUs, the default is to have a CPU spin for each GPU to minimize latency.
- +256, Do NOT pin host memory.
  Pinning host memory improves transfer rates to/from the GPUs. It allows the transfers to occur without locking and unlocking each page during the transfers. For this reason it is selected as the default. Setting this flag causes host memory NOT to be pinned. If a machine is being heavily used for other tasks, it may not be possible to pin the required memory. Under these conditions this Parameter may be used to skip memory pinning

The following flag will allow running with ECC off on ECC capable devices:

- +512, Do not use ECC.
  This flag will use GPUs that have the ECC mode turned off or have recorded double-bit errors.
  CAUTION: Setting this bit can result in **WRONG** answers. It should only be used for testing hardware.
- +1024, Use all the GPUs installed.
  This flag will use all the GPUs in the system, regardless of their performance level. By default, ***FMS*** determines the most powerful GPU available and uses all GPUs that have that same performance. This allows a weaker GPU to be used to drive a display and more powerful GPU to be used for computation.

  Setting this flag on a system with unequal performing GPUs will result in a significant degredation of performance. ***FMS*** allocates an equal amount of work to each GPU, so the performance will degrade to the slowest GPU in the system.

Copyright © Multipath Corporation

D.2.2. FMS GPU PARAMETER GPUPR

Data Type

Integer

Default Value

1

Description

This parameter controls how much information is printed about each GPU. The following values are bit flags. To obtain the value for GPUPR, add the selected options. The following options are available:

- 0, Print no information.
- +1, Print general GPU hardware configuration at startup for each device.
  Indicate any devices not used by ***FMS*** due to insufficient compute capablilty. The information printed includes the model, capability, compute mode, number of multiprocessors, number of cores, amount of memory, driver and runtime versions and the board serial number.
- +2, Print all remaining information for each GPU.
  This includes the information available from nvndia-smi at startup including temperature, fan speed, and error correcting logs.
- +4, Print each task as the GPU is initialized.
- +8, Print the percentage of work done by the CPU's and GPU's as they are adjusted.
  This requires the ***FMS*** Parameters IMMPCT and ITRPCT to be in the range 1 to 99.

The following flags are primarily used for software development and testing.

- +16, Record timing events at the start and end of each call to the GPU's.
  This option is required to use any of the following options.
- +32, Print wall time used and the percentage for major functions.
  This includes matrix multiply, triangle solve, synchronous transfers and other for each GPU function call.
- +64, Print information about each event as it is being recorded.
- +128, Print internal debugging information used for development.

```
    May require the routine to be compiled with the debug option.
```

Copyright © Multipath Corporation

D.2.3. FMS GPU PARAMETER CUPATH

Data Type

Character

Default Value

(Linux) /usr/bin
(Windows) C:\Program Files\NVIDIA Corporation\NVSMI

Description

This is the name of the directory containing the nvidia-smi program. If nvidia-smi is installed in a location other than this default value, this specifies the directory location. It should be placed in the License File between the FMSSET and RETURN lines.

Copyright © Multipath Corporation

D.2.4. FMS GPU PARAMETER MAXGPU

Data Type

Integer

Default Value

Number of installed GPU processors.

Description

This is the number of GPU tasks that ***FMS*** runs in parallel. By default, this value is set to the number of GPU processors in the system. This allows a single ***FMS*** job to use all system resources and complete as fast as possible.

For each GPU processor being used, ***FMS*** starts a separate host thread that is dedicated to managing that GPU processor. You may set this parameter to a value less than the number of installed GPU processors, including a value of 0 to perform all the work in the host. If you try to set this parameter to a value greater than the number of installed GPU processors, a fatal error will occur.

Copyright © Multipath Corporation

D.2.5. FMS GPU PARAMETER GPUCPU

Data Type

Integer

Default Value

GPU threads are bound to groups of CPU cores in order.
For example, a system having 32 CPU cores and 4 GPUs, the GPU threads would be bound as follows:

GPU	CPUs used
1	1 - 8
2	9 - 16
3	17 - 24
4	25 - 32

Description

This parameter binds the thread controlling a GPU to run on a specified range of CPU cores. The bus where each GPU is interfaced will be controlled by a single CPU Processor. Binding the thread controlling that GPU to the cores in the processor where it is interfaced can reduce memory traffic.

The form of this command is:

```
GPUCPU=GPU#,CPU1:CPU2
```

where GPU# is the GPU number (starting at 1) and CPU1:CPU2 is the range of CPU cores (starting at 1). For example, to bind GPU 2 to CPU cores 9 through 16 you would enter

```
GPUCPU=2,9:16
```

This command should be placed in the license file between the FMSSET and RETURN lines.

You should not include hyperthreads when making these bindings. A list of hyperthreads is given on the CPU WEB page.

To determine what GPUs are attached to what processors and if binding the threads would make any difference, run the CPU-GPU memory tests described on CGTEST.

Copyright © Multipath Corporation

D.2.6. FMS GPU PARAMETER NOGPU

Data Type

Integer

Default Value

Use all the GPUs

Description

Each use of this Parameter masks out the use one GPU. Values range from 0 to the maximum number of GPUs - 1. You may use this option multiple times to mask out several GPUs. It is mostly used for testing.

The default value will use all the GPUs.

Examples

```
To NOT use GPUs 1 and 2,

NOGPU=1
NOGPU=2
```

Copyright © Multipath Corporation

D.2.7. FMS GPU PARAMETERS IMMPCT, ITRPCT, ITMPCT AND CSAFE

Data Type

Integer

Default Value

IMMPCT, ITRPCT, ITMPCT=0, but you should set your own value in the License File.
CSAFE=90

Description

These parameters control how the work is divided between the CPU and GPU processors. The values are expressed as an integer percentage of the amount of work done by the CPUs, ranging from 0 to 100 as follows:

- `0, Do not use the CPUs.`
`Perform all work using the GPUs only.`
- `1-99, Use the CPUs for the specified percentage of work.`
`The remaining work is performed by the GPUs.`
- `100, Perform all the work with the CPUs.`
`The GPUs do not do any work.`

The following ***FMS*** Parameters control the CPU / GPU work split during different parts of the computation.

FMS Parameters

Parameter	Type of Computation
IMMPCT	Matrix Multiply
ITMPCT	Triangle Solve
ITRPCT	Triangle Multiply Symmetric Only

A nonzero value for a Parameter will cause the workload of the CPUs to automatically adjust so the time spent by the CPUs will be CSAFE percent less than the timne spent by the GPUs.

Since the GPUs are more powerful than the CPU's, excees work added to the CPUs may significantly degrade performance. The CSAFE Parameter provides for a safe margin so the CPU time will remain below the GPU time. Even when the CPUs are not performing computation they play an important role in transferring data from main memory to and from the GPUs. If the CPUs become oversubscribed, these transfers may be delayed, degrading overall performance. For large GPU systems the best value will usually be the default 0, since the CPUs will be busy transferring data.

The ***FMS*** Parameter GPUPR contains an option (+8) to print how the workload is adjusted during processing. This split does not account for any loss in performance due to oversubscribing the CPUs.

While ***FMS*** will eventually achieve a balanced between CPU and GPU work for your machine and problem, improved performance may be achieved by changing the default of these Parameters to a value closer to your configuration and problem. After some testing you should set the correct default values in the License File. The **Matrix Warrior** Parameters WEB page will offer suggestions on how these values should be set.

Copyright © Multipath Corporation

D.2.8. FMS GPU PARAMETER GPUPCT

Data Type

Integer

Default Value

99

Description

This Parameter is used to limit the amount of GPU memory used by ***FMS***. The units are percent of free memory in the GPU when it is initialized by ***FMS***. The range is 1 to 100. If the GPU is also being used for display, enough memory must be left for the display to operate. By default it is assumed that the GPU is used entirely for computation, so 99% of GPU memory available is allocated.

Copyright © Multipath Corporation

D.2.9. FMS GPU PARAMETERS ICHUNK, ITRCHK AND ITMCHK

Data Type

Integer

Default Values

ICHUNK = 192

ITRCHK = 192

ITMCHK = 192.

Description

These parameters control how data is divided to stream to/from the GPU in parallel with processing. The computation influenced by these values are

FMS Parameter

Parameter	Type of Computation
ICHUNK	Matrix Multiply
ITRCHK	Triangle Solve
ITMCHK	Triangle Multiply Symmetric Only

If you change the default value you should consider the following:

- Low values:
  - (-) Smaller transfer to/from the GPUs,
  - (-) Less reuse of data on the GPUs,
  - (-) Less work performed with each kernel execution,
  - (+) Quicker startup and shutdown.
- High values:
  - (+) Larger transfers to/from the GPUs,
  - (+) Increased reuse of data on the GPUs,

- (+) More work performed with each kernel execution,
- (-) Longer startup and shutdown.

Values that are not an integer multiple of CUDA cores may result in decreased performance.

This parameter also influences the selection of matrix block size. ***FMS*** will try to select blocks NEQBIO that are an integer multiple of ICHUNK * MAXGPU.

Copyright © Multipath Corporation

D.2.10. FMS GPU PARAMETERS CGTEST, CGMPCT, CGLOOP

Data Type

Integer

Default Value

CGTEST = 0, No test is performed,
CGMPCT = 25,
CGLOOP = 3.

Description

These Parameters tests the speed of data transferrs from memory to GPUs. GPU threads are bound to one or more CPU cores and the transfer rate to the GPU is measured.
The following tests are available:

- = 0 (default)
No test performed.
- = 1
GPU threads are left bound to the range of CPU cores specified by GPUCPU. The first test runs the GPUs one at a time, measuring the transfer rate (single threded). The second test runs them in parallel. The time required by each GPU is independently measured and the transfer rates computed. This second test gives an indication of any paths that may saturate when all the GPUs are transferring in parallel.
- = 2
The transfer rate from each CPU core to each GPU is measured independently. The result is a table listing the transfer rates where GPUs are the columns and CPUs are the rows.
- = 3
Perform both tests (1) and (2).

CGMPCT specifies the size of the data transfers, as a percentage of memory on the GPU.

CGLOOP specified the number of times the test is repeated.

The results are available on the **MatrixWarrior** CPU-GPU WEB page.

This command should be placed in the license file between the FMSSET and RETURN lines. For example

```
FMSSET
CGTEST=3
RETURN
```

Copyright © Multipath Corporation

D.2.11. FMS GPU PARAMETER MNCORE

Data Type

Integer

Default Value

0

Description

This parameter allows selection of GPUs based on the number of cores available. The default value of 0 will use all GPUs with the largest number of cores. GPUs with fewer cores will not be used. ***FMS*** divides the work among the GPUs evenly so the processing time will be determined by the slowest GPU. This Parameter is mostly used for testing.

Only GPUs with compute capability of 1.3 or higher will be used.

Copyright © Multipath Corporation

D.2.12. FMS GPU PARAMETER MAXEVT

Data Type

Integer

Default Value

Read-only

Description

This is a read-only parameter which returns the maximum number of events that were recorded during a single call to a ***FMS*** GPU routine. When the ***FMS*** Parameter GPUPR includes the 16 bit, ***FMS*** records the start and stop of each GPU call, including transfers to/from the GPU and kernel executions. The recorded information is used for various reports which are selected by the parameter GPUPR.

You may obtain the value of this parameter by including

```
CALL FMSIGT('MAXEVT', MAXEVT)
```

in your program or entering MAXEVT at the FMSSET> prompt.

FMS will record a maximum of 40 events on each call. Additional events are not recorded. You may check this Parameter to determine if the maximum number of recorded events was reached.

Copyright © Multipath Corporation

D.2.13. FMS GPU PARAMETERS GPUTHI, GPUTLO AND GPUTWS

Data Type

Integer

Default Value

GPUTHI = 95 (deg c)
GPUTLO = 90 (deg c)
GPUTWS = 0 seconds (do not perform test)

Description

This optional test monitors the GPU temperatures to prevent an automatic GPU shutdown. Most modern GPUs perform this test automatically. However if you are using an older GPU, or experience an unexpected shutdown, you should select this test by specifying a nonzero value for GPUTWS (120 would be a good value).

The GPU Parameter GPUTHI specify a high temperature (deg. c) that the GPU should not exceed.

The more GPUs are used, the warmer they get. Under some circumstances they may reach a temperature (usually 100 deg. c) that will cause a shutdown. This automatically occurs to protect the GPU. When this happens the application will eventually terminate with an unpredictable error condition. In order to prevent unwanted shutdown and job termination, ***FMS*** monitors the GPU temperature. If the temperature reaches GPUTHI deg. C, a cooling loop is initiated that prevents any additional tasks being run on the GPU. During this loop the GPU remains idle, allowing it to cool to a safe temperature, as specified with GPUTLO Parameter. When the safe temperature is reached, processing resumes.

If after MAXTWS seconds the safe lower temperature has not been reached, processing will resume. A value of MAXTWS=0 is used to skip this test if you know your GPU performs it automatically.

Copyright © Multipath Corporation

D.2.19. FMS GPU PARAMETER MAXTWS

Data Type

Integer

Default Value

120

Description

This Parameter specifies the maximum number of seconds a GPU will spend idle in a single cooling loop.

The more GPUs are used, the warmer they get. Under some circumstances they may reach a temperature (usually 100 deg. c) that will cause shutdown. This automatically occurs to protect the GPU. When this happens the application will eventually terminate with an unpredictable error condition. In order to prevent unwanted shutdown and job termination, ***FMS*** monitors the GPU temperature. If the temperature reaches GPUTHI deg. C, a cooling loop is initiated that prevents any additional tasks being run on the GPU. During this loop the GPU remains idle, allowing it to cool to a safe temperature, as specified with the GPUTLO Parameter. When the safe temperature is reached, processing resumes.

If after MAXTWS seconds the safe lower temperature has not been reached, processing will resume.

\

Copyright © Multipath Corporation

D.3. FMS NUMA PARAMETERS

Non-Uniform Memory Access (NUMA) Parameters

Index	NAME	Function
D.3.1	NUMAFL	Master flag. Must be set to 1 for any of the following parameters to take effect.
D.3.2	NUMAPR	Print output level
D.3.3	MYCPU1	Starting processor to use. 1 is first processor.
D.3.4	NPNODE	Number of processors per node
D.3.5	NUMNOD	Number of nodes (read-only)
D.3.6	NUMAFX	Thread binding to processor options.
D.3.7	MDWHEN	When memory is allocated
D.3.8	MAXLMD	Stride between nodes for placing memory (bytes)
D.3.9	NUMATP	Topology (SGI only)
D.3.10	MMGLUE	Combine blocks during matrix multiply

Copyright © Multipath Corporation

D.3.1. FMS NUMA PARAMETER NUMAFL

Data Type

Integer

Default Value

0

Description

This is the master flag for activating all the NonUniform Memory Access (NUMA) directives. You must set this parameter in the **FMS** License File in order to activate any of the NUMA technology. The following values are available:

- 0, Do not use NUMA technology.
- 1, Use NUMA technology for large problems.
- 2, Use NUMA technology for all problems.
  This option will use NUMA technology even if the problem is so small that it results in using less than the number of processors. This is mostly useful for debugging.

The NUMA directives discussed below are a powerful set of programming tools for achieving maximum performance. These are the directives which ***FMS*** uses internally to achieve peak performance. However, you may also use these directives in your own application.

Many of the NUMA features used by ***FMS*** work best when the machine is dedicated to ***FMS*** applications. This may be a single application or multiple applications where each one is explicitly placed. Running more than one application on a group of processors, or letting the operating system schedule other tasks on top of a ***FMS*** NUMA job may significantly degrade performance.

The fundamental building block of a NUMA machine is the "node". Within the node, all processing cores share a common memory with high-speed uniform access. This may be a single multi-core processor or multiple processors each having multiple cores. Programming at this level is similar to Symmetric MultiProcessor (SMP) programming techniques. The ***FMS*** Parameter NPNODE defines the number of processors per node. It defaults to a value appropriate for your machine.

The number of nodes NUMNOD is obtained by dividing the total number of processing cores being used MAXCPU

by the number of processors per node NPNODE. This parameter is automatically computed by ***FMS***.

In order to achieve peak performance on a NUMA machine, the software must maximize references to local memory and minimize references to remote memory. This requires close coordination between which processor runs the thread and where the memory is placed. In addition, the software must be designed to distribute the data in a fashion similar to programming techniques used on distributed memory computers.

To achieve peak performance, ***FMS*** implements the following features:

Memory Placement

When ***FMS*** places memory using NUMA directives, the requested memory is dealt out round robin among the nodes using a stride that is specified by the ***FMS*** Parameter MAXLMD. For example, the records used to hold matrix data use a stride that evenly distributes the record among the number of nodes being used for the problem. If you allocate memory for your application using one of the ***FMS*** memory management routines FMSIMG, FMSRMG or FMSCMG, and make the call from the parent, your memory will be distributed according to the specified value of MAXLMD. If you call the ***FMS*** memory management routines to allocate memory from a child thread, all the memory requested will be allocated on the node where that thread is running. You may also allocate memory and place it on each node with a single call from the parent using FMSILG, FMSRLG or FMSCLG. This provides a simple, yet effective way for you to control where your data resides on the NUMA machine.

For some applications, a single distribution of data will be optimal for the entire job. Most ***FMS*** applications, however, go through different phases, with each phase requiring a different data distribution. For example, an application may form matrix data, factor, solve and process results as distinct phases. ***FMS*** includes the Parameter MDWHEN that controls when the memory is placed. When the NUMA flag is set, memory is placed as it is required. When the NUMA flag is not set, all the memory is allocated once at the beginning.

Thread Binding

FMS includes options for attaching each thread to an individual processing core or any processing core on the node. The NUMAFX Parameter controls these options. When the NUMA flag is set, this parameter automatically defaults to the optimum value for each machine.

The threads are placed in increasing order, starting on processing core MYCPU1 (1 is the first CPU core in the machine), and continuing for MAXCPU processing cores. You may obtain the thread number MYCPU of a subroutine you have running in parallel by calling FMSIGT('MYNODE', MYCPU). Knowing the thread number and how the memory was distributed, you can determine what calculations should be performed on the local data.

One simple case is filling a matrix. First, you call the routine that fills the matrix in parallel using FMSPAR. At the beginning of the subroutine, you find out what thread you are by calling FMSIGT ('MYNODE', MYCPU). Next, you call one or more of the ***FMS*** memory management routines FMSIMG, FMSRMG or FMSCMG to allocate memory for your data. Because these routines are being called from a thread, they will automatically allocate the memory

on the local node. After performing your part of the computation, you return the memory and return from your subroutine.

Copyright © Multipath Corporation

D.3.2. FMS NUMA PARAMETER NUMAPR

Data Type

Integer

Default Value

1

Description

This parameter controls how much information is printed about the NUMA placement. The following values are available:

- 0, No information.
- 1, Information about the node topology.
This includes the relationship between CPU cores and nodes and the amount of memory on each node.
- 2, Information on the binding of threads to nodes or CPU cores
- 4, Information on memory placement before each major function.
- 8, Information on memory placement after each memory allocation.
- 16, Internal details on NUMA calls.
- 32, Debug information on NUMA calls.

Copyright © Multipath Corporation

D.3.3. FMS NUMA PARAMETER MYCPU1

Data Type

Integer

Default Value

1

Description

This parameter specifies the starting processing core for NUMA placement. Processing cores are numbered starting at 1 and continuing through the number of processing cores requested MAXCPU. If the default value is to be changed, this Parameter must be specified in the License File.

This parameter may be used to run multiple ***FMS*** NUMA jobs concurrently using different sections of the machine. For example, if you want to run eight jobs on a 32 processor system, you would specify values of MYCPU1 equal to 1, 5, 9, 13, 17, 21, 25, and 29 in each of the License Files. Note that you would run each of these jobs from a different directory, and each job would have its own License File.

Copyright © Multipath Corporation

D.3.4. FMS NUMA PARAMETER NPNODE

Data Type

Integer

Default Value

Machine dependent. Computed by ***FMS***.

Description

This parameter specifies the number of processing cores per node. You should not need to change this parameter.

Copyright © Multipath Corporation

D.3.5. FMS NUMA PARAMETER NUMNOD

Data Type

Integer

Default Value

MAXCPU / NPNODE, rounded up to an integer.

Description

This parameter specifies the number of nodes. You should not need to change this parameter.

Copyright © Multipath Corporation

D.3.6. FMS NUMA PARAMETER NUMAFX

Data Type

Integer

Default Value

0

Description

This parameter specifies how the compute and I/O threads are bound to processors. The following values are bit flags. To obtain the value for NUMAFX, add the selected options. The following options are available:

- 0, Do not bind threads.
- +1, Bind compute threads to individual processing cores.
- +2, Bind I/O threads to individual processing cores.
- +4, Place the parent on the target processor before starting the thread.
  On some systems, this is required to locate the stack and data segments of the new thread on the target CPU.
- +8, Request is mandatory (SGI only).
- +16, Bind the compute threads to their appropriate node.
- +32, Bind the I/O threads to their appropriate node.

These options provide an essential part of the NUMA technology. They provide the link between what memory is addressed by each thread and where the memory is located.

Two choices are available for binding threads. One choice binds the threads to specific processing cores. This has the advantage of assuring that maximum reuse of cache will occur, even if a thread is interrupted by an I/O process.

The second choice binds the threads to the node. This allows the threads to be scheduled on any processing core on the node. This allows the I/O and CPU threads to make maximum use of the compute resources on the node. However, if the thread looses its affinity for the processing core and is rescheduled on another processing core, additional cache loads will result.

FMS distributes the I/O threads uniformly among the nodes.

Copyright © Multipath Corporation

D.3.7. FMS NUMA PARAMETER MDWHEN

Data Type

Integer

Default Value

0 when NUMAFL is not set

1 when NUMAFL is set

Description

This parameter specifies when memory is allocated from the system. The following options are available:

- `0, Once at the beginning.`
- `1, As requested by` ***FMS*** `or your application.`

When NUMA technology is not used, ***FMS*** allocates MAXMD words of memory when FMSINI is called. Requests by ***FMS*** or your application are allocated from this memory pool.

When NUMA technology is used, the memory is allocated from the system on each request by ***FMS*** or your application. This provides for a different distribution of memory among the nodes for each memory request.

Copyright © Multipath Corporation

D.3.8. FMS NUMA PARAMETER MAXLMD

Data Type

Integer

Units

Bytes

Default Value

Hardware page size

Description

This parameter specifies the stride by which memory is allocated among the nodes. The allocation begins by placing MAXLMD bytes on the first node, followed by MAXLMD bytes on the second node. This process continues until all the requested memory has been placed. If not all the memory has been placed at the end of the last node, the process is repeated starting over at the first node.

By default, if the process requesting memory placement is not the parent, all the memory is placed on the node on which that process is running.

To place all the requested memory on the parent node, specify MAXLMD=0.

Copyright © Multipath Corporation

D.3.9. FMS NUMA PARAMETER NUMATP

Data Type

Integer

Default Value

3

Description

This parameter specifies which hardware nodes are to be used. When multiple ***FMS*** jobs are being run, you want the nodes clustered close together. Currently this parameter only applies to the SGI Origin series, which has multiple levels of remote nodes. On the Compaq GS series, there is only level, a node is local or not.

All possible options are listed below. However, for this ***FMS*** implementation, option 3 provides the most control and is recommended.

The following options are available:

- 0, TOPOLOGY_FREE:
- 1, TOPOLOGY_CUBE:
- 2, TOPOLOGY_CUBE_FIXED:
- 3, TOPOLOGY_PHYSNODES:
  The nodes are explicitly placed by name.
- 4, TOPOLOGY_CPUCLUSTER:
- 5, TOPOLOGY_LAST:

The default option 3 will explicitly place the memory and threads on specific hardware nodes This is implemented by creating a list of hardware node names in file FMSNOD. The order of these names is the order in which the nodes will be placed.

You may use standard UNIX tools to create file FMSNOD. First, create a list of hardware node names as follows:

```
find /hw -name node -print > FMSNOD1
```

On an 8-processor Origin 3000 system, this file would typically contain the following:

```
/hw/module/001c07/node
/hw/module/001c07/Ibrick/xtalk/15/pci/6/ohci/0/scsi_ctlr/0/node
```

```
/hw/module/001c07/Ibrick/xtalk/15/pci/6/ohci/0/node
/hw/module/001c10/node
```

The first and last lines are compute nodes, the second and third lines are not compute nodes. The next step is to use an editor on file FMSNOD1 and delete those lines that are not compute nodes. In this example, that would leave

```
/hw/module/001c07/node
/hw/module/001c10/node
```

The final step is to number the nodes in a format expected by ***FMS***. You may use the UNIX tool awk as follows:

```
awk '{printf("%3d %s\n",NR,$1)}' FMSNOD1 > FMSNOD
```

The resulting file FMSNOD will contain

```
  1 /hw/module/001c07/node
  2 /hw/module/001c10/node
```

The ***FMS*** hardware node number is in columns 1-3 right-justified, and the hardware node name starts in column 5, left-justified.

You may change the topology used by changing the order of hardware node names in this file.

Only one FMSNOD file is required for each system. If you change the system configuration by adding or removing nodes, you will need to make a new FMSNOD file.

Copyright © Multipath Corporation

D.3.10. FMS NUMA PARAMETER MMGLUE

Data Type

Integer

Default Value

0, Most machines

1, NUMA machines

Description

This parameter directs ***FMS*** to redistribute the work among threads on a node during matrix multiply to reduce the amount of nonlocal memory references. Where possible, this is accomplished by computing blocks that are more square in shape.

Copyright © Multipath Corporation

D.4. FMS FILE PARAMETERS

File Parameters

Index	NAME	Function
D.4.1	INCORE	File location (disk or memory)
D.4.2	IEXIST	Use existing file
D.4.3	IFKEEP	Keep files
D.4.4	IREST	Write restart information
D.4.5	IOLAST	Write to end of file on open
D.4.6	IOKIDS	Open for access by children
D.4.7	NUMIOQ	Number of I/O queues
D.4.8	NUMSTP	Number of stripes per queue
D.4.9	NUMVOL	Number of volumes per stripe
D.4.10	MBYTES	Number of megabytes per volume
D.4.11	IOTYPE	Perform raw or direct I/O
D.4.12	IOMASK	Bind FMSDISK thread to CPU cores
D.4.13	IOWARN	Special I/O warning options
D.4.14	IOSYNC	Perform synchronous I/O
D.4.15	LSECTR	Disk sector length (8-byte words)
D.4.16	NOOPEN	Do not physically open files
D.4.17	LBUFSI	Buffer size for submatrix integer records
D.4.18	LBUFSR	Buffer size for submatrix real records
D.4.19	LBUFSV	Buffer size for submatrix vector records
D.4.20	FREEDU	Free disk space

Copyright © Multipath Corporation

Copyright © Multipath Corporation

D.4.1. FMS FILE PARAMETER INCORE

Data Type

Integer

Default Value

0

Description

This parameter determines whether ***FMS*** stores the file in memory or on disk. The following options are available:

- 0, Store the file on disk.
- 1, Store the file in memory. If the file does not fit, exit with an error condition.
- 2, Store the file in memory if it fits. Otherwise store the file on disk.

With today's large memory machines, it is possible for some smaller problems to fit entirely in memory. In this case ***FMS*** creates "memory files" and eliminates all read/write operations to and from disk.

When using the option INCORE=2, the order in which the files are opened must be considered. When subroutine RSDI, RNDI, CHDI, CSDI, or CNDI is called to create a matrix file, it makes a decision on the record size based on the amount of memory available in the **FMS** memory pool. If that memory is removed by creating an incore file before the matrix is factored, it may result in an insufficient memory error during factoring.

This condition can be avoided by opening the vector and submatrix files first, then calling RSDI, RNDI, CHDI, CSDI, or CNDI to open the matrix file. If the matrix file fits in memory, all is fine. If it doesn't, it will be correctly blocked for the memory that remains.

In an extreme condition, it is possible for the submatrix and vector files to use all the memory pool, leaving nothing for the matrix. If this happens, simply change INCORE back to 0.

Copyright © Multipath Corporation

D.4.2. FMS FILE PARAMETER IEXIST

Data Type

Integer

Default Value

0

Description

This parameter determines whether ***FMS*** creates a new file or uses an existing file. The following options are available:

- 0, Create a new file. This is similar to the FORTRAN directive STATUS='NEW' on the OPEN statement..
- 1, Use an existing file. Fail with a file not found error if the file does not exist. This is similar to the FORTRAN directive STATUS='OLD' on the OPEN statement.
- 2, Use an existing file if it can be found. Otherwise create a new file. This is similar to the FORTRAN directive STATUS='UNKNOWN' on the OPEN statement.

This option allows you to reopen ***FMS*** files that have been previously saved with the IFKEEP parameter.

When you reopen a file, you must supply the file attribute list that was used for the file when it was created. If the file was created in a separate job, this list must be written to and reread from a restart file.

Copyright © Multipath Corporation

D.4.3. FMS FILE PARAMETER IFKEEP

Data Type

Integer

Default Value

0

Description

This parameter determines whether ***FMS*** keeps or deletes the file when it is closed. The following options are available:

- `0, Delete the file.`
- `1, Keep the file.`

This option can be used to create permanent ***FMS*** files. These files can later be reopened in the same or different job using the ***FMS*** parameter IEXIST.

By default ***FMS*** files are considered scratch files and deleted when the file is closed. By setting IFKEEP=1 prior to calling FMSCM, FMSCV or FMSCS, the file will not be deleted.

If you want to keep the file, it is strongly recommended that you set IFKEEP=1 before the file is opened. On some systems this will save the file if the job terminates abnormally.

Copyright © Multipath Corporation

D.4.4. FMS FILE PARAMETER IREST

Data Type

Integer

Default Value

0

Description

This parameter determines whether ***FMS*** flushes all writes, and waits on write operations during factoring. The following options are available:

> - 0, Do not flush writes or wait on writes during factoring. Provides for the maximum asynchronous I/O.
> - 1, Flush all writes to disk. These flush operations are performed by the I/O threads and operate in parallel with processing. They assure that data written to disk is actually placed on the disk, rather than remaining in memory in the buffer cache. If you have specified IOTYPE=2 for direct I/O (available on some machines), this happens automatically.
> In addition, at the end of factoring each segment, **FMS** waits for any outstanding write operations and saves any additional information which may be required to restart at that point.

This option can be used when factoring a large matrix to save intermediate results should the factoring process stop prematurely. To enable a job to be restarted during factoring, should the job not complete, the following steps are required on the initial job:

- Set the ***FMS*** Parameter IFKEEP=1 to save the files.
- Set the ***FMS*** Parameter IREST=1 to keep the files current.
- Save the matrix file attributes lists to a file of your choice.
- Organize your application so partially factored results do not corrupt unfactored matrix data. Some options are:
  - Use separate files for the input and output matrices,
  - Generate the matrix data during the factoring process (MDATAU=2).

If you use a single matrix file and overlay the factored matrix data on the original matrix values, then you may need to rebuild part of the matrix when restarting. The matrix segment immediately after the highest factored equation printed by ***FMS*** may contain partially written records of factored data. This data must be restored to unfactored matrix data before the restart process is begun.

To restart a job during factoring, the following steps are required:

- Read the matrix file attributes you saved.
- Set the ***FMS*** Parameter IEXIST=1 to direct ***FMS*** to open existing files.
- Reopen the matrix file(s), specifying the saved file attribute lists.
- Set the ***FMS*** Parameter LOWASM to the first unfactored equation. Review the output printed by ***FMS*** during the first run to determine the highest equation factored. The highest unfactored equation is one more than this value.
- Branch in your application to the factoring routine which was interrupted.
- If you want to delete the ***FMS*** files at the end of the job, set the ***FMS*** Parameter IFKEEP=0.

Copyright © Multipath Corporation

D.4.5. FMS FILE PARAMETER IOLAST

Data Type

Integer

Default Value

0

Description

This parameter determines whether ***FMS*** writes a word at the end of a file when it is opened. The following options are available:

- `0, Do not write a word at the end of the file when it is opened.`
- `1, Write a word at the end of the file when it is opened.`

Writing a word at the end on open will physically reserve the space and guarantee that sufficient disk space is available for solving the problem. However a significant amount of time may be required, as much as writing the entire file, to perform this operation.

Copyright © Multipath Corporation

D.4.6. FMS FILE PARAMETER IOKIDS

Data Type

Integer

Default Value

0

Description

This parameter determines whether ***FMS*** permits read or write access to files by children threads. The following options are available:

- 0, Open files once for parent access.
- 1, Open files once for parent access and again for each child.

If you do not plan to read or write to files from children processes you created by calling FMSPAR, you should leave this parameter at the default value.

For some applications, you may want to populate the matrix by calling FMSPUT, FMSROW or FMSCOL from subroutines you are running in parallel from child threads you created by calls to FMSPAR. In order for ***FMS*** to create the file descriptors for concurrent access by children, you must specify the number of children that will have access when the file is opened. This parameter provides that function.

If you are performing raw I/O (IOTYPE=1) and want access to the raw devices from the children, you must specify this parameter in the license file. Raw devices are opened and initialized during the call to FMSINI.

The number of file descriptors may become large, especially if the file contains a large number of stripes and is accessed by many children. You should reset this value to the default value for all files which do not require access by children.

The valid range of values for IOKIDS is 0, for parent access only, to (MAXCPU - 1) for all children.

On some machines only a single file descriptor is required for access by both the parent and children. In these cases, ***FMS*** internally sets the value of IOKIDS to -1 to indicate that the file does not need to be opened also by the children. Any attempt by the application program to change IOKIDS is ignored. While it is not necessary to set IOKIDS on these machines, programming your application as described above will provide portability across all platforms.

Copyright © Multipath Corporation

D.4.7. FMS FILE PARAMETER NUMIOQ

Data Type

Integer

Default Value

1

Description

This parameter specifies the number of queues used by the ***FMS*** file striping system.

When FMS writes a record to a striped file, the record is first divided among the queues. Within each queue, the record is then divided among the stripes. The total number of disks running in parallel for each record transfer is the number of queues times the number of stripes.

FMS uses this two layer file striping system to accommodate machines which have distributed memory. By running separate queues in each memory region, data is automatically transferred to the region where it is required.

The maximum number of queues is 10.

This option may also be used on systems having centralized memory if the number of stripes exceeds 10.

Copyright © Multipath Corporation

D.4.8. FMS FILE PARAMETER NUMSTP

Data Type

Integer

Default Value

1

Description

This parameter specifies the number of stripes per queue used by the ***FMS*** file striping system.

When ***FMS*** writes a record to a striped file, the record is first divided among the queues. Within each queue, the record is then divided among the stripes. The total number of disks running in parallel for each record transfer is the number of queues times the number of stripes.

FMS uses this two layer file striping system to accommodate machines which have distributed memory. By running separate queues in each memory region, data is automatically transferred to the region where it is required.

The maximum number of stripes per queue is 10. If you want to further stripe the files, use multiple queues.

Copyright © Multipath Corporation

D.4.9. FMS FILE PARAMETER NUMVOL

Data Type

Integer

Default Value

0

Description

This parameter specifies the number of volumes used for each stripe in the ***FMS*** file striping system. Use this parameter to create a large disk out of two or more smaller disk partitions.

For large files the amount of storage required per stripe may exceed the size of the disk partition. This parameter provides for binding multiple disk partitions together.

On some systems there is a 2 Gbyte address limitation for individual disk files. This option may be used to exceed that limitation.

When NUMVOL is greater than 1, ***FMS*** writes data to the first volume until MBYTES of data have been transferred. ***FMS*** then switches to the second volume. This process continues until the entire file has been written.

Only symmetric configurations are supported. All disk volumes must have the same capacity.

Copyright © Multipath Corporation

D.4.10. FMS FILE PARAMETER MBYTES

Data Type

Integer

Default Value

0

Description

This parameter specifies the capacity of each volume of the ***FMS*** file striping system in megabytes. It is only used if you specify a value for NUMVOL greater than 1.

When NUMVOL is greater than 1, ***FMS*** writes data to the first volume until MBYTES of data have been transferred. ***FMS*** then switches to the second volume. This process continues until the entire file has been written.

Only symmetric configurations are supported. All disk volumes must have the same capacity.

Copyright © Multipath Corporation

D.4.11. FMS FILE PARAMETER IOTYPE

Data Type

Integer

Default Value

0

Description

This parameter determines how ***FMS*** performs I/O to the disks. The following options are available:

- 0, Perform I/O using the file system.
- 1, Perform raw I/O directly to the disks, which are mounted as raw devices. (UNIX system only).
- 2, Perform I/O directly to the disks without using the buffer cache.
- 3, Perform I/O using the file system. Flush writes to disk.

By default, file systems first write data to a buffer maintained by the operating system (buffer cache). At some point determined by the operating system, this data is then physically transferred to disk. You may force data to be physically transferred to disk during each write operation by setting IOTYPE=3. For applications with small files this can improve performance by eliminating small physical transfers to/from disk.
For large files which exceed the size of physical memory, moving the data first to a buffer cache and then to the disk makes no sense. Eventually the data must reside on disk. The double move results in memory contention between the operating system trying to maintain the buffer cache and ***FMS*** memory. Further ***FMS*** performs disk transfers in parallel with processing (asynchronous I/O). By having the data first accumulate in the buffer cache, and then transfer to disk in a large chunk as determined by the operating system, prevents the uniform continuous flow of data when controlled by ***FMS***. By setting IOTYPE=2, this double move is eliminated and the ***FMS*** asynchronous I/O works at maximum performance.

For small problems, where the data will fit in memory, specify the ***FMS*** Parameter INCORE=1. This will allow ***FMS*** to operate directly on the data in your application without any moves.

RAW I/O

FMS supports raw I/O to perform transfers to/from disks that totally bypasses the operating system. This can be useful on systems that do not support direct I/O.

To use the RAW I/O feature, you specify IOTYPE=1 in the license file. In addition, you must provide a file called FMSRAW that describes the raw devices. This file contains a record for each raw I/O device in the following format:

```
Column 1: Queue  number (0 to 9)
Column 2: Stripe number (0 to 9)
Column 3: Volume number (0 to 9)
Column 4: Space
Column 5 and up: Raw device name
```

The first record contains the mount point for the unstriped files. This is the information normally placed in the directory FMSDISK. The first four columns of this record should be blank.

Each of the following records specifies the mount point for one of the FMS stripes. The values of the ***FMS*** Parameters NUMIOQ, NUMSTP and NUMVOL determine the number of records required. This is the information normally placed in directories FMSDISKijk, where i is the queue number, j is the stripe number and k is the volume number.

FMS opens each of these files when FMSINI is called. When you call FMS routines to create files from your application, they are placed within these files.

You may also use the RAW I/O option to address files that are part of the UNIX file system. If you specify a valid path name instead of the raw device name in the FMSRAW file, FMS opens the file and treats it as a raw device. While this does not provide the potential performance benefits of true RAW devices, it may be used to store some of the file data as normal files and some as raw files.

RAW I/O Example

Suppose we have 3 disk drives (rrz3c, rrz4c and rrz5c) that we have initialized as raw devices. We would like to create a 3-way file stripe system to store the large ***FMS*** files on these devices (the large ***FMS*** files are vector files and off-diagonal matrix files). Normally these files would be placed in the directories FMSDISK000, FMSDISK010 and FMSDISK020. The small nonstriped files we want to place in a normal file called /tmp/MYDATA, which we have created with the command

```
touch /tmp/MYDATA
```

The file FMSRAW would contain the following:

```
    /tmp/MYDATA
000 rrz3c
010 rrz4c
020 rrz5c
```

where rrz is the character entry point to the disk partition as listed in the /dev directory.

Without RAW I/O, ***FMS*** would normally place the files in the directories FMSDISK, FMSDISK000, FMSDISK010 and FMSDISK020. With RAW I/O, the files are placed within the RAW files. In essence, each RAW file contains the same information that would normally be placed in a directory.

Determining the type of file systems on Linux

You can determine the file systems used for each mount point on Linux with the command

```
df -T
```

Copyright © Multipath Corporation

D.4.12. FMS FILE PARAMETER IOMASK

Data Type

Integer

Default Value

0

Description

This parameter binds the I/O thread for FMSDISK to a range of CPU cores. The following options are available:

> - `0, Thread for FMSDISK can run on any CPU core.`
> - `CPU1:CPU2, Thread for FMSDISK can only run on CPU cores CPU1 through CPU2. CPU cores are numbered starting at 1.`

This parameter binds the thread performing I/O to FMSDISK to run on a specified range of CPU cores. On most systems, the I/O is performed by the first processor. Restricting the I/O thread to run on those processing cores can reduce traffic. The form of this command is:

```
IOMASK=CPU1:CPU2
```

where CPU1:CPU2 is the range of CPU cores (starting at 1). For example to bind the FMSDISK thread to CPU cores 1 through 8 you would enter

```
IOMASK=1:8
```

This command should be placed in the license file between the FMSSET and RETURN lines.
This feature works best when hyperthreading is turned off (usually a BIOS setup option).

Copyright © Multipath Corporation

D.4.13. FMS FILE PARAMETER IOWARN

Data Type

Integer

Default Value

0

Description

This parameter provides additional options to the ***FMS*** file system. Most of these options may be used to debug your application when performing direct I/O.
The value of IWARN you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IOWARN.

- 0, No warning messages are printed. When performing direct I/O, ***FMS*** increases the transfer length if required.
- +1, Stop ***FMS*** if a condition is found that violates the requirements for direct I/O. You must also select one of the warning message tests listed below: (+4), (+8), (+16).
- +2, Do not increase the length of transfers, even if the requested length does not satisfy the conditions for direct I/O.
- +4, Print a warning message containing "DIO" if the transfer length requested does not satisfy the conditions for direct I/O. Note that by default, ***FMS*** automatically increases the transfer length if required. However this may have undesirable side effects if insufficient space was allocated on disk or in memory.
- +8, Print a warning message containing "DIO" if the starting disk address does not satisfy the conditions for direct I/O.
- +16, Print a warning message containing "DIO" if the starting memory address does not satisfy the conditions for direct I/O.
- +1024, Do not print properties of removable disks. This option is useful in Windows systems when the removable media is not present.

A typical use of this Parameter is as follows:

1. Add the following two lines to the license file:

```
IOTYPE=2
IOWARN=28
```

2. Run your application, directing the output to a file, myout.txt.

```
./myapp > myout.txt
```

3. Search for warning messages:

```
grep DIO myout.txt
```

Copyright © Multipath Corporation

D.4.14. FMS FILE PARAMETER IOSYNC

Data Type

Integer

Default Value

0

Description

This parameter determines whether ***FMS*** performs file transfers synchronously or asynchronously. The following options are available:

- `0, Perform file transfers asynchronously. This allows data transfers to occur in parallel with processing to reduce runtime.`
- `1, Perform file transfers synchronously. The CPU processes are suspended while file transfers occur.`

By default, ***FMS*** uses separate I/O processes running in parallel with the CPU processes to perform disk transfers. These I/O processes automatically read ahead so data is present in memory when required. After data is computed, it is turned over to the I/O processes for writing to disk so the CPU processes can continue with the solution.

This option should always be left at the default value unless you are conducting performance experiments.

Copyright © Multipath Corporation

D.4.15. FMS FILE PARAMETER LSECTR

Data Type

Integer

Default Value

Computed by ***FMS***.

Description

This parameter is set automatically when FMSINI is called. It specifies the minimum physical transfer length to the file system per stripe.

If you are using disk drives that are hardware striped, or are using the ***FMS*** direct I/O option with IOTYPE=2, this Parameter may need to be increased beyond its default value. When this occurs, ***FMS*** calculates and prints the correct value for LSECTR, which may then be set in the license file.

Copyright © Multipath Corporation

D.4.16. FMS FILE PARAMETER NOOPEN

Data Type

Integer

Default Value

0

Description

A nonzero value of this parameter skips the physical opening of ***FMS*** files. It is used to estimate the amount of disk space that will be required when files are actually opened by a later application. When using this option, you may perform a reduced initialization by calling FMSIN2, provided no matrix algebra or I/O is being performed. EXAMPLE_20 illustrates how this Parameter may be used to compute disk space requirements.

When a file is opened and this parameter is set, ***FMS*** skips the final open and assigns a file ID of 0 for the file. All other operations are performed the same as if the file were physically opened. When a matrix file is opened by calling RSDI, RNDI, CHDI, CSDI or CNDI, the returned Matrix File Attribute list is completed. When a vector file is opened by calling FMSOV, the returned Vector File Attribute list is completed. If you list the open ***FMS*** files by calling FMSCST(" SHOW", "FILES"), the files will be listed.

When you no longer need the opened files, you should call FMSCM to close matrix files or FMSCV to close vector files.

Several ***FMS*** Parameters influence how files are blocked and the amount of storage that will be required. Normally these parameters are set automatically if you are using the target machine to estimate disk space for an application on the same machine. However, if you want to estimate the disk space requirements for a different machine, you should set the following ***FMS*** Parameters values for the target machine on which the files will actually be opened. These Parameters may be set in the license file.

NAME	Function
	Hardware Parameters
MAXCPU	Number of processors to use
MAXMD	Amount of memory to use (8-byte words)
MDUSED	Amount of memory already used (8-byte words, read-only)

LPAGE	Page size (bytes)
ITPCPU	Processor type (1=scalar, 2=vector)
LENVRG	Vector register length (8-byte words)
	NUMA Parameters
NUMAFL	Master flag. Must be set to 1 for any of the following parameters to take effect.
NPNODE	Number of processors per node
NUMNOD	Number of nodes (read-only)
MAXLMD	Stride between nodes for placing memory (bytes)
	File Parameters
INCORE	File location (disk or memory)
NUMIOQ	Number of I/O queues
NUMSTP	Number of stripes per queue
LSECTR	Disk sector length (8-byte words)
	Matrix File Initialization Parameters
MFMAT	Matrix format (PROFILE, BLOCK, SLAB)
NEQBIO	I/O block size (BLOCK & SLAB format)
NEQBLK	Compute block size
NEQBLO	Minimum compute block size
NEQBHI	Maximum compute block size
NEQBV	Vector block size
	Matrix Factoring Parameters
IALGOR	Optional factoring algorithms

Copyright © Multipath Corporation

D.4.17. FMS FILE PARAMETER LBUFSI

Data Type

Integer

Default Value

8192 Integer words.

Description

This parameter determines the size of the buffer used for submatrix integer records.

When you call FMSOS to open a submatrix file, two buffers of size LBUFSI are created to store the submatrix integer data. As you call FMSWRT or FMSSWR to write the data to disk, it is first transferred to these buffers.

FMS delays opening a physical file until both buffers are full. That way if the two buffers are large enough to hold the entire file, no disk transfers are performed.

If the amount of data you write exceeds 2*LBUFSI, a physical file having a record length LBUFSI is opened. As you continue to write to the file, data is staged in one buffer while the other is written to disk asynchronously.

For today's large memory machines, it is usually possible to keep the submatrix files incore. You should try setting the INCORE option prior to calling FMSOS to create an incore submatrix file.

Copyright © Multipath Corporation

D.4.18. FMS FILE PARAMETER LBUFSR

Data Type

Integer

Default Value

16384 REAL*8 (8-byte) words

Description

This parameter determines the size of the buffer used for submatrix real records which hold the submatrix data **[Si]**.

When you call FMSOS to open a submatrix file, two buffers of size LBUFSR are created to store the submatrix real data. As you call FMSWRT or FMSSWR to write the data to disk, it is first transferred to these buffers.

FMS delays opening a physical file until both buffers are full. That way if the two buffers are large enough to hold the entire file, no disk transfers are performed.

If the amount of data you write exceeds 2*LBUFSR, a physical file having a record length LBUFSR is opened. As you continue to write to the file, data is staged in one buffer while the other is written to disk asynchronously.

For today's large memory machines, it is usually possible to keep the submatrix files incore. You should try setting the INCORE option prior to calling FMSOS to create an incore submatrix file.

Copyright © Multipath Corporation

D.4.19. FMS FILE PARAMETER LBUFSV

Data Type

Integer

Default Value

8192 REAL*8 (8-byte) words

Description

When you call FMSOS to open a submatrix file, two buffers of size LBUFSV are created to store the submatrix vector data. As you call FMSWRT or FMSSWR to write the data to disk, it is first transferred to these buffers.

FMS delays opening a physical file until both buffers are full. That way if the two buffers are large enough to hold the entire file, no disk transfers are performed.

If the amount of data you write exceeds 2*LBUFSV, a physical file having a record length LBUFSV is opened. As you continue to write to the file, data is staged in one buffer while the other is written to disk asynchronously.

For today's large memory machines, it is usually possible to keep the submatrix files incore. You should try setting the INCORE option prior to calling FMSOS to create an incore submatrix file.

Copyright © Multipath Corporation

D.4.20. FMS FILE PARAMETERS FREEDU, FREEDS

Data Type

REAL*8 (8-byte) words

Default Value

Read-only

Description

These Parameters return the amount of free space in Gbytes.

> **FREEDU** returns the free disk space on the unstriped file system, pointed to by the environment variable FMSDISK.
>
> **FREEDS** returns the free disk space on the striped file system, pointed to by the environment variables FMSDISKijk.

To obtain these values in your program

```
REAL*8 FREEDU, FREEDS
CALL FMSRGT('FREEDU', FREEDU)
CALL FMSRGT('FREEDS', FREEDS)
```

To display a report on the state of the current file system in your program

```
CALL FMSCST('SHOW','FMSDISK')
```

Copyright © Multipath Corporation

D.5. FMS MATRIX FILE PARAMETERS

Matrix File Parameters

Index	NAME	Function
D.5.1	MFMAT	Matrix format (PROFILE, BLOCK, SLAB)
D.5.2	LENSDF	I/O record size (PROFILE format)
D.5.3	NEQBIO	I/O block size (BLOCK & SLAB format)
D.5.4	NEQBLK	Compute block size
D.5.5	NEQBLO	Minimum compute block size
D.5.6	NEQBHI	Maximum compute block size
D.5.7	NEQBV	Vector block size
D.5.8	IJSTEP	Stride toward diagonal
D.5.9	INSIDE	Blocking pattern (PROFILE format)
D.5.10	MINDSK	Minimize disk usage

Copyright © Multipath Corporation

D.5.1. FMS MATRIX FILE PARAMETER MFMAT

Data Type

Integer

Default Value

1 for sparse matrices

2 for full matrices.

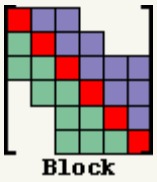
Description

This parameter controls how the matrix **[A]** is stored and processed. The following options are available:



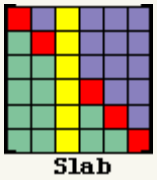
MFMAT=1: Profile Solver

Accounts for the sparsity of matrix **[A]** on an equation by equation basis. Typical applications include finite element and finite difference programs. This is the default for sparse matrices.



MFMAT=2: Block Solver

Divides the matrix **[A]** into square blocks, accounting for sparsity on a block by block basis. This solver uses industry standard BLAS3 kernels and provides excellent I/O performance on extremely large problems. Typical applications include boundary integral and moment method programs. This is the default for full matrices.



MFMAT=3: Slab Solver

Extends the functionality of the Block Solver by providing full column partial pivoting for full nonsymmetric matrices. You must explicitly set MFMAT to select this option.

FMS stores the matrix data on three separate files:

- File LUA(1) contains the terms in the lower triangle, **[AL]**
- File LUA(2) contains the matrix diagonals as a single vector, **{D}**
- File LUA(3) contains the terms in the upper triangle, **[AU]**

For symmetric matrices this file is not used.

For nonsymmetric problems ***FMS*** assumes that the sparsity of the matrix is symmetric but the values are not. This allows ***FMS*** to use a single set of pointers to address the matrix elements in the lower triangle **[AL]** and upper triangle **[AU]**. Data in the upper triangle is always stored in a transposed direction from data in the lower triangle. For example if the **FMS** parameter IJSTEP is one, the data in **[AL]** will be stored by rows and the data in **[AU]** will be stored by columns. Note that if a symmetric matrix was processed as nonsymmetric, the files LUA(1) and LUA(3) would be identical.

FMS uses a single vector **{LOWEQ}** to describe the sparsity of the matrix. This vector contains the lowest coupled equation for each equation. For terms in the lower triangle **[AL]** this is the first non-zero term of each row, proceeding from the left edge of the matrix toward the diagonal. For terms in the upper triangle **[AU]** this is the first non-zero term of each column, proceeding from the top of the matrix toward the diagonal. Because ***FMS*** uses symmetric storage, the same profile vector **{LOWEQ}** describes both the sparsity of **[AL]** and **[AU]**. Note that for a full matrix, the profile vector **{LOWEQ}** would contain all 1's.

Copyright © Multipath Corporation

D.5.2. FMS MATRIX FILE PARAMETER LENSDF

Data Type

Integer

Units

REAL*8 (8-byte) words

Default Value

0 = determined by ***FMS*** in subroutine RSDI, RNDI, CHDI, CSDI, CNDI
PROFILE matrixes only (MFMAT=1).

Description

This is the record size used for storing the off-diagonal matrix data in **[AL]** and **[AU]** on files LUA(1) and LUA(3) for Profile matrices.

When ***FMS*** selects this value, the following factors are considered:

- The shape of the matrix, as specified by the profile vector **{LOWEQ}**
- The amount of memory available in the ***FMS*** memory pool
- The compute block dimension NEQBLK
- The number of CPU processing cores MAXCPU
- The file location INCORE
- The minimum unit of address for the disk LSECTR
- The number of stripes per record, as specified by NUMIOQ and NUMSTP

If you change the default value of LENSDF, ***FMS*** may change the value you specify if it determines that your specified value would not allow processing to complete.

If you are creating permanent matrix files in different jobs that you later want to process together, you may need to set LENSDF so all files will have compatible data storage.

Copyright © Multipath Corporation

D.5.3. FMS MATRIX FILE PARAMETER NEQBIO

Data Type

Integer

Units

Equations

Default Value

0 = determined by ***FMS*** in subroutine RSDI, RNDI, CHDI, CSDI, CNDI

Block and Slab matrices only (MFMAT=2 or 3).

Description

This is the block size used for storing the off-diagonal matrix data in **[AL]** and **[AU]** on files LUA(1) and LUA(3) for Block and Slab matrices. The record size on files LUA(1) and LUS(3) in REAL*8 (8-byte) words will be approximately

```
(record size) = IDTYPE*NEQBIO*NEQBIO
```

where IDTYPE is 1 for REAL*8 data and 2 for COMPLEX*16 data.

When ***FMS*** selects this value, the following factors are considered:

- The amount of memory available in the ***FMS*** memory pool
- The compute block dimension NEQBLK
- The number of CPU processing cores MAXCPU
- The number of GPUs MAXGPU
- The GPU chunk size ICHUNK
- The file location INCORE
- Optional processing algorithms IAGOR
- The minimum unit of address for the disk LSECTR
- The number of stripes per record, as specified by NUMIOQ and NUMSTP

If you change the default value of NEQBIO, ***FMS*** may change the value you specify if it determines that your specified value would not allow processing to complete.

If you are creating permanent matrix files in different jobs that you later want to process together, you may need to set NEQBIO so all files will have compatible data storage.

Copyright © Multipath Corporation

D.5.4. FMS MATRIX FILE PARAMETER NEQBLK

Data Type

Integer

Units

Equations

Default Value

0 = determined by ***FMS*** in subroutine RSDI, RNDI, CHDI, CSDI, CNDI

Description

This is the compute block size used by ***FMS*** for matrix processing. At the lowest level, ***FMS*** implements highly optimized matrix kernel operations which are designed to achieve peak performance on each specific machine. The block size used by these operations is often dictated by the machine architecture (number of floating point registers, size of cache, etc.). These optimized kernels are therefore written for specific values of NEQBLK which maximize the use of these low level machine resources.

These matrix kernels are designed to be run on a single processing core or GPU. They become the atomic building blocks for all ***FMS*** operations. ***FMS*** allocates these kernel operations to the processing cores to obtain the parallel solution.

FMS may have more than one optimized kernel available. For example, it is typical to have a different kernel and value of NEQBLK for real and complex data. Typically kernels are designed to operate efficiently over a range of values. ***FMS*** uses the Parameters NEQBLO and NEQBHI to specify that range. Default values of these parameters are set for each machine architecture.

You may change the value of NEQBLK but be prepared for dramatic changes in performance. For example, setting NEQBLK = 1 will have the effect of performing all processing at the level 1 BLAS performance.

Copyright © Multipath Corporation

D.5.5. FMS MATRIX FILE PARAMETER NEQBLO

Data Type

Integer

Units

Equations

Default Value

Machine dependent

Description

This Parameter determines the minimum compute block dimension. It is used when ***FMS*** determines the compute block size NEQBLK. If you specify a value for NEQBLK, that value will be used and this parameter will be ignored.

Copyright © Multipath Corporation

D.5.6. FMS MATRIX FILE PARAMETER NEQBHI

Data Type

Integer

Units

Equations

Default Value

Machine dependent

Description

This Parameter determines the maximum compute block dimension. It is used when ***FMS*** determines the compute block size NEQBLK. If you specify a value for NEQBLK, that value will be used and this parameter will be ignored.

Copyright © Multipath Corporation

D.5.7. FMS MATRIX FILE PARAMETER NEQBV

Data Type

Integer

Units

Equations

Default Value

Machine dependent

Description

This Parameter determines the block size used for storing vectors on vector files when vector blocking is used.

Copyright © Multipath Corporation

D.5.8. FMS MATRIX FILE PARAMETER IJSTEP

Data Type

Integer

Units

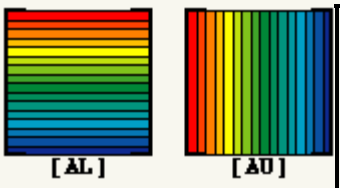
Equations

Default Value

0 = determined by ***FMS*** in subroutine RSDI, RNDI, CHDI, CSDI, CNDI.

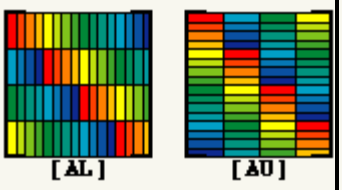
Description

This is the stride toward the diagonal that ***FMS*** uses for storing off-diagonal matrix data on files LUA(1) and LUA(3). The following options are available:



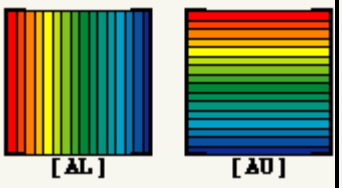
IJSTEP=1

Data in the lower triangle **[AL]** is stored by rows. Data in the upper triangle **[AU]** is stored by columns.



IJSTEP=NEQBLK

Data in the lower triangle **[AL]** is first stored in columns NEQBLK equation tall. These columns are then stored in a direction proceeding toward the diagonal. Data in the upper triangle **[AU]** is first stored in rows NEQBLK equations wide. These rows are then stored in a direction proceeding toward the diagonal.



IJSTEP=NEQBIO

Data in the lower triangle **[AL]** is stored in the blocks by columns (not transposed). Data in the upper triangle **[AU]** is stored in the blocks by rows (transposed). Applies to Block and Slab matrices only.

Note that in all cases, the storage in **[AU]** is the transpose of the storage in **[AL]**.

This option is provided in ***FMS*** to accommodate different machine architectures. In all cases, fetching data from

memory sequentially is desirable (incremental addressing). The following algorithms are matched to each of the data storage options provided by IJSTEP to produce incremental addressing.

IJSTEP=1

This data storage is naturally aligned for performing dot products

```
        DO J = 1,N
           DO I = 1,N
              S = 0
C             Loop across row I of [AL], down column J of [AU]:
              DO K = 1,N
                 S = S + AL(I,K)*AU(K,J)
              END DO
              C(I,J) = C(I,J) + S
           END DO
        END DO
```

On most machines dot products give good performance because there are only two memory load operations and no memory store in the inner-loop. However, the accumulation of data into the register S is not implemented on some machines with vector hardware.

IJSTEP=NEQBLK

This is a variation of the dot product algorithm designed for RISC processors. For example, suppose NEQBLK=2. Then 4 dot products would be computed in an interleaved fashion by:

```
        DO J = 1,N,NEQBLK
           DO I = 1,N,NEQBLK
              S11= 0
              S21= 0
              S12= 0
              S22= 0
              DO K = 1,N
C                Fetch next NEQBLK terms from [AL] and [AU]:
                 S11 = S11 + AL(I  ,K)*AU(K,J  )
                 S21 = S21 + AL(I+1,K)*AU(K,J  )
                 S12 = S12 + AL(I  ,K)*AU(K,J+1)
                 S22 = S22 + AL(I+1,K)*AU(K,J+1)
              END DO
              C(I  ,J  ) = C(I  ,J  ) + S11
              C(I+1,J  ) = C(I+1,J  ) + S21
              C(I  ,J+1) = C(I  ,J+1) + S12
```

```
               C(I+1,J+1) = C(I+1,J+1) + S22
            END DO
         END DO
```

Note that all data is fetched from **[AL]** and **[AU]** incrementally. In addition, the inner-loop has only 4 memory loads for 4 multiply and 4 add operations. This uses only half the memory bandwidth of the dot product. Increasing NEQBLK further reduces memory requirements. This algorithm is preferred for RISC processors. NEQBLK is picked as large as possible to use all the floating point registers for accumulations.

IJSTEP=NEQBIO

This data is naturally aligned for performing outer products

```
         DO K = 1,N
            DO J = 1,N
C           Loop across row J of [AU]:
             S = AU(K,J)
C              Loop down column I of [C] and [AL]:
               DO I = 1,N
                  C(I,J) = C(I,J) + AL(I,K)*S
               END DO
            END DO
         END DO
```

This algorithm avoids the accumulation of the dot product and is optimal for some vector machines.
If you attempt to set IJSTEP to a value that is not permitted, ***FMS*** will correct it to the closest reasonable value.

The default value of IJSTEP is designed to work in conjunction with the optimized matrix kernels specified with NEQBLK. Changing the value of IJSTEP may significantly effect performance.

NOTE: If you are performing substructuring, values of IJSTEP=NEQBLK are not permitted. In some cases, it may be necessary to override the default values.

Copyright © Multipath Corporation

D.5.9. FMS MATRIX FILE PARAMETER INSIDE

Data Type

Integer

Default Value

0 = determined by ***FMS*** in subroutine RSDI, RNDI, CHDI, CSDI, CNDI.

Profile matrices only.

Description

This parameter determines how the optimized matrix kernels, which are NEQBLK equations square, are fit within the matrix profile. The following options are available:

- -1, Outside Blocking.
  The profile is expanded so the compute blocks exactly fit. This results in all processing being performed with the optimized kernels. However, extra 0's around the profile must be stored and processed.
- 1, Inside Blocking.
  The blocks are fit within the matrix profile, starting at the diagonal and moving outward. The terms adjacent to the profile which do not fit within the compute blocks are performed with normal code.

Which value of INSIDE provide the best performance for your problem depends on the shape of your profile and the size of the compute blocks that have been optimized for your machine. In order to help you make this decision, ***FMS*** includes the option IPRI=1024, which analyzes your profile for both options.

Copyright © Multipath Corporation

D.5.10. FMS MATRIX FILE PARAMETER MINDSK

Data Type

Integer

Default Value

0

Description

This parameter changes the block selection criteria to minimuze the amount of disk space required. The following options are available:

- `0, The matrix blocking is selected for maximum performance.`
- `1, The matrix blocking is selected to minimize disk usage.`

Copyright © Multipath Corporation

D.6. FMS MATRIX ASSEMBLY PARAMETERS

Matrix Assembly Parameters

Index	NAME	Function
D.6.1	LOWASM	First changed equation for nonlinear reassembly
D.6.2	MDATAU	Call your subroutine to define or modify matrix data
D.6.3	LUOK	Make a nonsymmetric matrix symmetric

Copyright © Multipath Corporation

D.6.1. FMS MATRIX ASSEMBLY PARAMETER LOWASM

Data Type

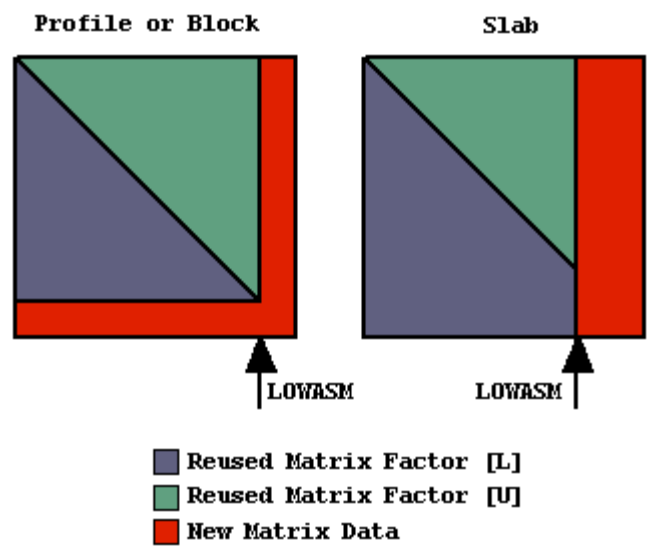
Integer

Default Value

1

Description

LOWASM specifies the first equation in the matrix which has a term that changed. By properly locating the equations containing nonlinearities at the bottom of the matrix, you may save a significant amount of computation.



FMS will round the value of LOWASM down to start on a segment boundary. This prevents records from containing a mix of factored and unfactored data.

FMS begins assembling or factoring the new matrix at the segment containing LOWASM. For Profile or Block format matrices, this is the row number in the lower triangle **[AL]** and column number in the upper triangle **[AU]**. For Slab format matrices, this is the column number in **[AL]** and **[AU]**. As the new matrix factors **[L]** and **[U]** are computed, they are overwritten on the files containing the previous factors.

One benefit of this feature is that you do not need to know the restart point in advance when the matrix **[A]** is

factored the first time. You simply determine LOWASM as you are processing the solution **{X}** and ***FMS*** automatically skips the appropriate calculations on the next solution.

For example, in a structural finite element analysis, this feature would be used as follows:

1. Initialize your value of LOWASM to 1 greater than the number of equations.
2. Loop over the elements. Use the displacements **{X}** from the previous solution to compute new values of strains and stresses.
3. If nonlinear effects are observed due to large deformations or material nonlinearities, set LOWASM=MIN0(LOWASM,ILOW), where ILOW is the lowest equation in the element.
4. `CALL FMSIST('LOWASM',LOWASM)`

   to pass your value of LOWASM to ***FMS***.

This parameter may also be used to recover a factoring process that terminates prematurely. During the initial factoring, set the ***FMS*** Parameter IREST to write the restart data. On the second (recovery) run, set this ***FMS*** parameter LOWASM to the highest factored segment (or slab) printed by ***FMS*** during the first run.

Copyright © Multipath Corporation

D.6.2. FMS MATRIX ASSEMBLY PARAMETER MDATAU

Data Type

Integer

Default Value

0

Description

This parameter directs ***FMS*** to call subroutines you provide to define or modify the matrix coefficients. The following options are available:

- 0, Do not call the user-supplied subroutines.
- 1, Call the user-supplied subroutines while assembling the matrix during RSDA, RNDA, CHDA, CSDA, CNDA or RSDAF, RNDAF, CHDAF, CSDAF, CNDAF.
- 2, Call the user-supplied subroutines before factoring the matrix in RSDF, RNDF, CHDF, CSDF, CNDF or RSDAF, RNDAF, CHDAF, CSDAF, CNDAF.

The subroutines you need to provide depend on your matrix format, data type and matrix symmetry according to the following table:

Matrix Format	Data Type	Matrix Symmetry	Subroutines Required
PROFLIE or BLOCK	Real	Symmetric	RSUBLK
PROFLIE or BLOCK	Real	Nonsymmetric	RSUBLK, RNUBLK
PROFLIE or BLOCK	Complex	Hermitian	CHUBLK
PROFLIE or BLOCK	Complex	Symmetric	CSUBLK
PROFLIE or BLOCK	Complex	Nonsymmetric	CSUBLK, CNUBLK
SLAB	Real	Nonsymmetric	RNUSLB

SLAB	Complex	Nonsymmetric	CNUSLB

Copyright © Multipath Corporation

D.6.3. FMS MATRIX ASSEMBLY PARAMETER LUOK

Data Type

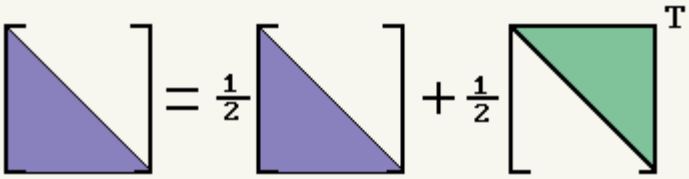
Integer

Default Value

0

Description

This parameter directs ***FMS*** to make a symmetric matrix from a nonsymmetric matrix by averaging the off-diagonal terms. The following options are available:



- `0, Do not average the matrix data.`
- `1, Form a symmetric matrix by averaging` **`[AL]`** `and` **`[AU]`**`.`

Because symmetric matrices may be factored twice as fast as nonsymmetric matrices, forcing a matrix to be symmetric can save a considerable amount of processing time. In some applications it is known from the physics of the problem that the matrix should be symmetric. However, due to numerical modeling, the matrix coefficients appear slightly nonsymmetric. Using this option for this application can save both computer time, storage and improve accuracy.

A second application for this feature occurs when matrix data is generated. In some cases generating half of the matrix coefficients as required for a symmetric matrix will naturally populate some terms in the upper triangle and some terms in the lower triangle. Transposing the upper triangle terms to force all data into the lower triangle may be awkward. This feature allows you to provide both lower and upper triangles during matrix assembly and then compact the data into symmetric form after the matrix coefficients have been generated. Note that ***FMS*** automatically initializes all matrix coefficients to zero so only the nonzero values need to be specified.

To use this feature you initially open the matrix as nonsymmetric by calling subroutine RNDI or CNDI. The matrix is

then populated as a nonsymmetric matrix. You then call the symmetric factoring subroutine RSDAF or CSDAF to average the matrix and factor it as symmetric.

Copyright © Multipath Corporation

D.7. FMS MATRIX FACTORING PARAMETERS

Matrix Factoriong Parameters

Index	NAME	Function
D.7.1	MPOSDF	Positive definite flag
D.7.2	MZERO	Zero pivot flag
D.7.3	NBITSZ	Number of lost bits on diagonal to call zero
D.7.4	NBITSP	Number of lost bits on diagonal to print a warning
D.7.5	RPIVOT	Value to set zero pivot reciprocal
D.7.6	PIVMIN	Absolute value of minimum pivot
D.7.7	NPIVOT	Number of equations pivoted (SLAB format) read-only
D.7.8	NUMSCG	Number of diagonal sign changes during factoring read-only
D.7.9	NEQSUB	First equation of substructure
D.7.10	IALGOR	Optional factoring algorithms
D.7.11	MINDIM	Minimum dimension to terminate Strassen's algorithm
D.7.12	SAVOPS	Number of floating point operations saved read-only
D.7.13	CONINV	Inverse condition number read-only
D.7.14	ATERMS	Number of terms in matrix read-only
D.7.15	AZEROS	Number of zero's in matrix read-only
D.7.16	FZEROS	Number of zero's in matrix factor read-only

D.7.17	IPOLLD	CPU synchronization on diagonal elements

Copyright © Multipath Corporation

D.7.1. FMS MATRIX FACTORING PARAMETER MPOSDF

Data Type

Integer

Default Value

0

Description

This parameter directs ***FMS*** to stop processing if it encounters a negative pivot during the factoring process. The following options are available:

- `0, Continue if a negative pivot is found.`
- `1, Halt if a negative pivot is found.`

For many problems it is known that the matrix should be positive definite. This condition implies that there should be no negative pivots during the matrix factoring process. If a negative pivot is found for a positive definite matrix, it is an error condition. This parameter provides the option of halting the processing when this error condition occurs.

If this flag is set and this error condition occurs, it is probably due to incorrect matrix data above the equation where the error occurred.

Copyright © Multipath Corporation

D.7.2. FMS MATRIX FACTORING PARAMETER MZERO

Data Type

Integer

Default Value

0

Description

This parameter directs ***FMS*** to perform special actions if a zero pivot is encountered during the factoring process. The following options are available:

- 0, Halt with an error condition.
- 1, Set the pivot reciprocal value to the ***FMS*** parameter RPIVOT, print a warning message and continue.
- 2, Call subroutine RSUPIV or CSUPIV which you provide to compute the pivot reciprocal value.

When factoring a matrix, the diagonal elements (pivots) must be inverted. This calculation fails if the pivot value is zero. This option provides you with alternatives to bypass this error, which may be appropriate for some applications.

One option is to set the value of the pivot reciprocal to zero and proceed. This has the effect of replacing the original diagonal in the matrix with a huge number. In effect, this eliminates that equation from the system by constraining its solution value to zero.

An example of where this is useful is in structural analysis. If the model was improperly constrained, rigid body motion (translation and rotation) will be possible. At some point in the analysis this will result in a zero pivot for each of the unconstrained rigid body motions (up to 3 translations and 3 rotations). By using the option MZERO=1, the model automatically becomes constrained as required. The equations whose diagonals are modified by this option become the reference points for the solution value.

A similar condition occurs in fluid analysis where there may be a large spread in the magnitude of the matrix values (ill-conditioned matrix). By replacing the pivots which are close to zero with a known value RPIVOT, the solution

can proceed.

For special applications you may want to handle this error condition yourself. By setting MZERO to 2, ***FMS*** passes the appropriate information to your subroutine RSUPIV or CSUPIV. If you want to continue processing, you pass the pivot reciprocal back to ***FMS***.

Copyright © Multipath Corporation

D.7.3. FMS MATRIX FACTORING PARAMETER NBITSZ

Data Type

Integer

Default Value

39

Description

This parameter controls the tolerance on zero when computing diagonal pivots during the factoring process.

During matrix factoring the diagonal terms (pivots) must be inverted. If their value is zero, their inverse cannot be computed. Because machines have finite precision, numbers "close" to zero should be treated as zero for this condition. This parameter defines what ***FMS*** uses for "close".

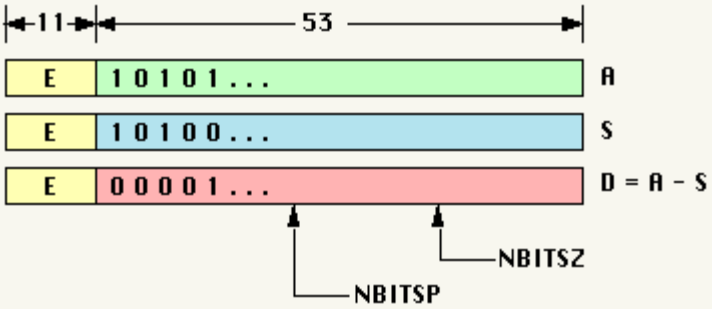
In order to establish a meaningful definition of "close to zero", it is necessary to examine how the diagonal pivots are computed. The actual calculation performed by ***FMS*** to factor the diagonals is given by

```
D = A - S
P = 1/D
```

where

- A is the original matrix diagonal,
- S is the dot product of the ith row of **[AL]** and the ith column of **[AU]** and
- P is the pivot reciprocal for the diagonal that is being computed.

If the value of S is close to A, the calculation of P will be in error. To see how this error occurs it is useful to examine this calculation from a machine's point of view. The values of D and S are stored as floating point numbers. For most machines today, that means 64 bits in IEEE format. Within these 64 bits, 11 are used to represent the exponent and 53 for the mantissa. In general, this provides about 15 decimal digits of accuracy.



When two numbers are "close" in value, their exponents and leading mantissa bits will be identical. If these numbers are subtracted, the result will have the same exponent but the leading mantissa bits which were identical will be set to zero. The machine then normalize the result by shifting the remaining nonzero mantissa bits to the left, inserting zero's in their place and appropriately adjusting the exponent. As you can expect, the number of zero's inserted into the mantissa during this process provides a measure of how close the numbers are in value and how much precision is lost in computing their difference.

FMS monitors this process and provides a way to treat the result of the subtraction as zero if a large number of zero's were inserted. The default value of NBITSZ=39 will treat the difference of two numbers as zero if their exponent and leading 39 bits of mantissa are identical. If you want to loosen this tolerance and increase the range of numbers that will be treated as zero, set the value of NBITSZ to a smaller number. If you want to tighten the tolerance, increase the value of NBITSZ.

NOTE: If the absolute value of (A - S) is less than the ***FMS*** Parameter PIVMIN, a zero pivot condition also results. (For complex data the value of (A - S) squared is compared to PIVMIN).

Copyright © Multipath Corporation

D.7.4. FMS MATRIX FACTORING PARAMETER NBITSP

Data Type

Integer

Default Value

26

Description

This parameter determines when ***FMS*** prints a warning message about pivots close to zero during the factoring process.

During matrix factoring the diagonal terms (pivots) must be inverted. If their value is zero or close to zero, their inverse may be an unreasonable large number (or infinity). This parameter controls when ***FMS*** warns about this condition with a printed warning message.

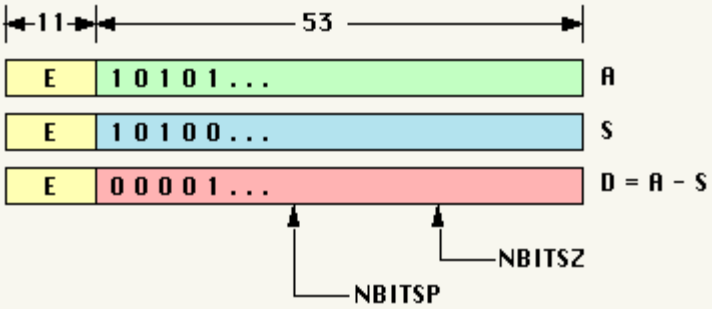
The actual calculation performed by ***FMS*** to factor the diagonals is given by

```
D = A - S
P = 1/D
```

where

- A is the original matrix diagonal,
- S is the dot product of the ith row of **[AL]** and the ith column of **[AU]** and
- P is the pivot reciprocal for the diagonal that is being computed.

If the value of S is close to A, the calculation of P will be in error. To see how this error occurs it is useful to examine this calculation from a machine's point of view. The values of D and S are stored as floating point numbers. For most machines today, that means 64 bits in IEEE format. Within these 64 bits, 11 are used to represent the exponent and 53 for the mantissa. In general, this provides about 15 decimal digits of accuracy.



When two numbers are close in value, their exponents and leading mantissa bits will be identical. If these numbers are subtracted, the result will have the same exponent but the leading mantissa bits which were identical will be set to zero. The machine then normalize the result by shifting the remaining nonzero mantissa bits to the left, inserting zero's in their place and appropriately adjusting the exponent. As you can expect, the number of zero's inserted into the mantissa during this process provides a measure of how much precision is lost.

FMS monitors this process and provides a way for you to be notified if the lost precision exceeds an unacceptable level. The default value of NBITSP=26 will provide a warning for each diagonal pivot that has lost 26 bits of precision (had 26 zero's inserted during the subtraction operation). If you get too many warning messages you could try setting the value of NBITSP to a larger number. If you want to monitor the process more closely, decrease the value of NBITSP.

CAUTION: If you decrease NBITSP to a value close to zero, you may get a warning message for each diagonal.

Copyright © Multipath Corporation

D.7.5. FMS MATRIX FACTORING PARAMETER RPIVOT

Data Type

REAL*8 (8-byte) words

Default Value

0

Description

When you specify the ***FMS*** parameter MZERO=1 and ***FMS*** detects a zero pivot, the pivot reciprocal is set to RPIVOT. This parameter specifies that value.

Copyright © Multipath Corporation

D.7.6. FMS MATRIX FACTORING PARAMETER PIVMIN

Data Type

REAL*8 (8-byte) words

Default Value

0

Description

When performing matrix factoring, if ***FMS*** detects a pivot value less than or equal to PIVMIN, a zero pivot condition results. For complex data, the square of the pivot is compared to PIVMIN.

The action taken when a zero pivot condition occurs is controlled by the ***FMS*** parameter MZERO.

A zero pivot condition can also result if ***FMS*** detects that a significant number of bits were lost in computing the pivot, as specified by NBITSZ.

Copyright © Multipath Corporation

D.7.7. FMS MATRIX FACTORING PARAMETER NPIVOT

Data Type

Integer

Default Value

This is a read-only parameter.
SLAB matrixes only.

Description

This parameter returns the number of equations which were pivoted (processed out of the natural order) during the factoring process. This value is also printed if the factoring print code IPRF contains the 2 option.

Copyright © Multipath Corporation

D.7.8. FMS MATRIX FACTORING PARAMETER NUMSCG

Data Type

Integer

Default Value

This is a read-only parameter.

Description

This parameter counts the number of times the sign of the diagonal changes during the factoring process. It may be used as part of a Sturm sequence to establish the number of Eigenvalues below the current shift point.

Copyright © Multipath Corporation

D.7.9. FMS MATRIX FACTORING PARAMETER NEQSUB

Data Type

Integer

Default Value

2147483647 which is (2^{31} - 1), a big number.

Description

This parameter modifies the factoring and solving processes to produce a reduced system of equations (substructuring). The new smaller system may be extracted, modified and solved.

The parameter NEQSUB partitions the original system of equations

[A]{X} = {B}

as follows:

```
+-          -+ +-  -+   +-  -+
| A11 | A12 | | X1 |   | B1 |
+-----+-----+ +----+ = +----+<--NEQSUB
| A21 | A22 | | X2 |   | B2 |
+--   |     -+ +-  -+   +-  -+
```

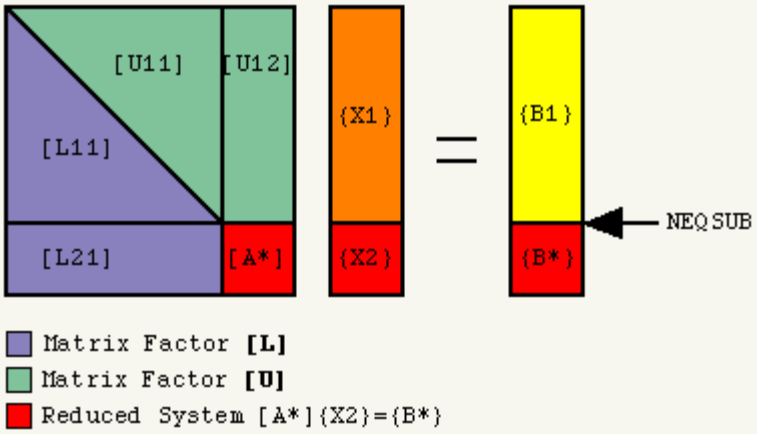
For nonsymmetric problems, the matrix terms **[A11]**, **[A21]** and **[A12]** are factored normally into **[L11]**, **[U11]**, **[L21]** and **[U12]**. However the matrix **[A22]** is only partially factored to produce the modified matrix **[A*]**,

[A*] = [A22] - [L21][U12] .

Similarly, during forward reduction the vector **{B1}** is reduced normally, but the vector **{B2}** is only partially reduced by

{B*} = {B2} - [L21]{B1} .

The result is a reduced system of equations



which may be solved for the solution vector **{X2}**. Once the solution vector **{X2}** is obtained, the values may be placed in the original vector **{X}** and ***FMS*** will complete the solution process for **{X1}**.

One important property of the reduced matrix **[A*]** and vector **{B*}** is worth noting. They are computed from the original matrix **[A22]** and vector **{B2}** by subtracting terms only. No multiplications on **[A*]** and **{B*}** have been performed. Therefore if you want to produce changes to the original terms in **[A22]** or **{B2}**, and those changes are in the form of additions or subtractions, they may be applied directly to the reduced system **[A*]** and **{B*}**.

The above strategy has some important applications which can save a considerable amount of computer time.

Design Optimization and Nonlinear Analysis

One application is in design optimization or nonlinear analysis where the system must be solved several times. Between each solution, some of the matrix coefficients in the lower right corner of the matrix are changed. Instead of completely resolving the entire system at each step, NEQSUB is first used to produce a reduced system containing the part of the matrix that changes. This smaller system is then used to find the optimal design or perform the nonlinear analysis. When the final solution is obtained, the values of **{X2}** are substituted back into the original system to compute **{X1}**. Note that the benefits of doing this depend on locating the changing matrix values in the bottom of the matrix.

Superelements

A second application is in building models of objects that have repeated sections. For example, suppose you want to analyze a ship that is constructed by repeating the same design down its length. You could first build a model of the generic section, numbering the equations so that the unknowns on the boundary were at the bottom of the matrix. The parameter NEQSUB could then be used to compute a reduced matrix **[A*]** which exactly represents the generic section in terms of the boundary unknowns only. The element assembly feature of ***FMS*** could then be used to assemble these generic sections into an overall model.

FMS contains some tools that can simplify this process. Subroutine RSDEX, RNDEX, CHDEX, CSDEX, CNDEX will extract the matrix **[A*]** from the original matrix file after factoring. Subroutine RSDANN, RNDANN, CHDANN, CSDANN, CNDANN will create a ***FMS*** matrix file from the extracted matrix **[A*]**. The regular ***FMS*** factoring and solving routines can be used to solve the reduced system.

Example

For further illustration on using NEQSUB to perform substructuring, refer to Example 7.

Copyright © Multipath Corporation

D.7.10. FMS MATRIX FACTORING PARAMETER IALGOR

Data Type

Integer

Default Value

Machine dependent

Description

This parameter directs ***FMS*** to use alternative algorithms when performing matrix multiplies during factoring, solving and matrix-vector multiply. These algorithms are designed to reduce the number of floating point operations performed, thereby providing a faster solution. The options may be selected independently. To determine the value for IALGOR, add the values of the options selected. (Note that the option values are a power of 2 and have the effect of setting bits in IALGOR).

- 0, Perform matrix multiplies using traditional techniques.
- +1, Use Strassen's algorithm for performing matrix multiplies. This algorithm modifies the traditional matrix multiply

  [C] = [A][B]

  by first bisecting each of the matrices **[A]** and **[B]** into 4 equal parts. By appropriately adding, subtracting and multiplying the resulting quadrants of **[A]** and **[B]**, it is possible to compute the matrix **[C]** with 7/8 of the multiplies required by the traditional technique. Because this algorithm is recursive, the matrices may be bisected several times, with each bisection reducing the remaining work by another factor of 7/8.

  Ultimately, however, the matrices become too small and the overhead of adding and subtracting the quadrants exceeds the 7/8 benefit. The ***FMS*** parameter MINDIM is used to terminate this process by specifying the minimum dimension of a matrix quadrant at or below which traditional techniques are used.

  There are, however, some downsides to this algorithm. First, because additions and subtractions are performed ahead of multiplications,

some precision will be lost. The amount is problem-dependent. You should always check your answer with the traditional technique, IALGOR=0.

Second, the adding and subtracting of the quadrants results in a more random memory access pattern than the traditional technique. For this reason, some of the benefit may be lost in cache thrashing.

Third, current floating point processors fuse the multiplier and adder so the output of the multiplier flows directly into the adder. This hardware development occurred after these software algorithms were originally developed. When additions are performed, multiply cycles go unused.

- +2, 3M Complex multiply (Complex data type only).

Perform complex matrix multiply using 3 instead of the traditional 4 multiply and add operations. In the traditional case, when two complex numbers A and B are multiplied and added to C, 4 multiplies and 4 adds are required as follows:

```
(Re_C + Im_C i) = (Re_A + Im_A i) * (Re_B + Im_B i)

P1 = Re_A*Re_B
P2 = Re_A*Im_B
P3 = Im_A*Re_B
P4 = Im_A*Im_B

Re_C = Re_C + (P1 - P4)
Im_C = Im_C + (P2 + P3)
```

In this reduced operation case, the same calculation is performed with 3 multiplies as follows:

```
P1 = Re_A*(Re_B - Im_B)
P2 = (Re_A + Im_A)*Re_B
P3 = (Re_A - Im_A)*Im_B

Re_C = Re_C + (P1 + P3)
Im_C = Im_C + (P2 - P1)
```

At first it would appear that we have not gained anything because, while we have reduced the number of multiplies from 4 to 3, we have increased the number of adds from 4 to 7. Since most computers perform multiplies and adds at the same speed (and many concurrently), we have actually increased the number of cycles to perform the operation. However when A, B and C become matrices **[A]**, **[B]** and **[C]**, the balance shifts. For matrices of size N by N, N^2 operations are required for addition while $2N^3$ operations are required for multiplication. For sufficiently large N, the extra additions become insignificant.

Again there is a down side to this algorithm. The additions and

```
subtractions performed prior to multiplies will reduce accuracy. The
amount is problem-dependent. Multiply cycles will be wasted on
floating point units with a fused multiply-adder.
```

- `+4, Complex data type only.`
  `This is the same as option 2, except the 3 real matrices are formed first, before the data is divided among the processors or Strassen's algorithm is applied. Since they are larger, the work performed for additions is reduced.`

For complex matrices both options may be selected with IALGOR=3 or 5.

In addition to observing run time, you may obtain the number of floating point operations saved from the parameter SAVOPS.

Copyright © Multipath Corporation

D.7.11. FMS MATRIX FACTORING PARAMETER MINDIM

Data Type

Integer

Default Value

Machine dependent

Description

This parameter specifies when Strassen's algorithm is terminated and traditional matrix multiplies are performed. When Strassen's algorithm is selected by specifying IALGOR=1, the matrices are recursively bisected to reduce the number of operations. At some point, the work performed bisecting the matrices exceeds the benefit. This parameter specifies that point.

If all dimensions of the matrices are greater than MINDIM, another bisection is performed. If any dimension is less than or equal to MINDIM, Strassen's algorithm is terminated and traditional matrix multiplies are performed.

FMS has set the default value of MINDIM based on tests performed on a machine of your type. However, you may want to try some experiments with other values.

Note that ***FMS*** makes every attempt to perform low level matrix operations with optimized compute blocks, whose size is NEQBLK. Therefore you will probably obtain the best results by picking values of MINDIM that are an integer multiple of the compute block dimension, NEQBLK.

Copyright © Multipath Corporation

D.7.12. FMS MATRIX FACTORING PARAMETER SAVOPS

Data Type

REAL*8 (8-byte) words

Default Value

Read only

Description

This parameter returns the number of floating point operations saved by the optional algorithms specified with the IALGOR parameter. This is an approximate number only.

For Strassen, it is based on 1/8 the number of multiply and add operations being saved at each level of the recursion.

For complex matrix multiply, it is based on 1/4 the number of multiply and add operations being saved.

This is a gross savings value. The amount of additional work required by these algorithms for adding and subtracting is not subtracted from SAVOPS.

In order to accumulate this data, you must specify a value of the LOGTIM parameter greater than or equal to 3.

Copyright © Multipath Corporation

D.7.13. FMS MATRIX FACTORING PARAMETER CONINV

Data Type

REAL*8 (8-byte) words

Default Value

Read-only

Description

After factoring, ***FMS*** computes the inverse of the condition number from the ratio of the minimum diagonal pivot to the maximum diagonal pivot. This value is available as parameter CONINV. This value is also printed if the factoring print code IPRF contains the 2 option.

In general, the condition number measures the spread of the Eigenvalues of a matrix. Matrices with condition numbers close to unity are well-conditioned. Matrices with large condition numbers are ill-conditioned.

Most computers today use a 64-bit IEEE floating point number, which provides about 15 decimal digits of precision. If the Eigenvalues of the matrix differ by more than that amount, the solution will fail. As they approach that limit, precision is lost.

If your solution fails due to an ill-conditioned matrix, you may need to change how you are modeling the problem. Often this can be as simple as remeshing to avoid regions that differ significantly in size. In other cases, the values picked as unknowns may need to be changed.

One simple example is the cantilever beam. If you were to model this structure using solid elements whose equations represent displacements, you would have an ill-conditioned matrix. There would be a low Eigenvalue corresponding to the diving board behavior of the beam and a very high Eigenvalue corresponding to the speed of sound through the beam. By changing the model to beam elements, which use rotations as unknowns, the high Eigenvalues are eliminated.

NOTE: Because the condition number could potentially approach infinity, ***FMS*** computes its inverse (which could approach 0).

Copyright © Multipath Corporation

D.7.14. FMS MATRIX FACTORING PARAMETER ATERMS

Data Type

REAL*8 (8-byte) words

Default Value

Read-only

Description

This parameter returns the number of terms (zero and nonzero) within the matrix profile. It includes the lower triangle, diagonal and upper triangle for nonsymmetric matrices. For complex data each complex number is counted as one term.

In order to accumulate this information, you must include the 2048 option in the IPRF print code parameter.

Copyright © Multipath Corporation

D.7.15. FMS MATRIX FACTORING PARAMETER AZEROS

Data Type

REAL*8 (8-byte) words

Default Value

Read only

Description

This parameter returns the number of zero terms within the matrix profile prior to factoring. It includes the lower triangle, diagonal and upper triangle for nonsymmetric matrices. For complex data, each complex number is counted as one term.

In order to accumulate this information, you must include the 2048 option in the IPRF print code parameter.

Copyright © Multipath Corporation

D.7.16. FMS MATRIX FACTORING PARAMETER FZEROS

Data Type

REAL*8 (8-byte) words

Default Value

Read-only

Description

This parameter returns the number of zero terms within the matrix profile after factoring. It includes the lower triangle, diagonal and upper triangle for nonsymmetric matrices. For complex data, each complex number is counted as one term.

In order to accumulate this information, you must include the 2048 option in the IPRF print code parameter.

Copyright © Multipath Corporation

D.7.17. FMS MATRIX FACTORING PARAMETER IPOLLD

Data Type

Integer

Default Value

0

Description

This parameter controls how ***FMS*** compute tasks are synchronized around the diagonal of the matrix during factoring. The following options are available:

- -1, SLAB matrices only.
  Compute the terms in the immediate vicinity of the diagonal using a single process.
- 0, Use event flags.
  This option provides for tasks to "sleep" if necessary.
- 1, Use spin locks.
  If a task needs to wait for data, it goes into a polling loop. Because the operating system cannot distinguish between polling and useful work, the task continues to tie up the processor. On some systems this can produce a deadlock condition if the number of tasks (***FMS*** or other jobs) exceeds the number of processors. For this reason, this option is not implemented on all systems.

When a matrix is factored, there is a lot of latitude in selecting the order in which the calculations are performed. Naturally the starting diagonal A(1,1) must be the first term computed and the final diagonal A(N,N) the last term computed. However, between these points, many options are available. As it turns out, the further you are away from the diagonal, the more choices you have. For example, the bottom left term A(N,1) can be computed right after A(1,1) or just before A(N,N).

FMS exploits this property by partitioning the matrix into off-diagonal areas which may be divided among the processors as independent tasks and diagonal areas where the tasks are coupled. This provides the best load balancing between the processors and minimizes the amount of synchronization which must be performed.

When computing close to the diagonal, the tasks are restricted to computing the diagonal terms in sequence. This requires them to communicate (signal) and perhaps wait for a brief period for a slower task to get caught up.

This parameter provides options on how this synchronization and communication are implemented. In general, the default value should be used.

Copyright © Multipath Corporation

D.8. FMS MATRIX SOLVING PARAMETERS

Matrix Solving Parameters

Index	NAME	Function
D.8.1	IPBACK	Use dot products during backsubstitution (PROFILE format)
D.8.2	LUTRAN	Solve **[A]T{X}={B}** instead of **[A]{X}={B}**

Copyright © Multipath Corporation

D.8.1. FMS MATRIX SOLVING PARAMETER IPBACK

Data Type

Integer

Default Value

0

Description

This parameter is used to modify how ***FMS*** performs back substitution for profile matrices. The following options are available:

- 0, Perform back substitution using outer products.
This is almost always the best choice if there is only one or a small number of RHS vectors.
- 1, Perform back subatitution using inner products.
Use memory to hold as many RHS vectors as possible and process the matrix segments 1 at a time. This achieves maximum reuse of the row of **[U]** that has been loaded into the temporary vector. However, the number of terms in the temporary vector is the number of equations in the matrix segment. For large problems in a small memory machine, this may lead to only a few terms being processed per execution of the inner-loop.
- N, Use memory to hold N RHS vectors.
By processing the matrix segments in groups, the temporary vector is longer. However **[U]** may need to be read multiple times per solution if all the RHS vectors do not fit in memory.

There are two options for the inner-loop when performing back substitution:

1. Outer products:

```
DO J = N,1,-1
S = X(J)
   DO I = 1,(J-1)
      X(I) = X(I) - S*U(I,J) <---Addresses [U] by columns
```

```
      END DO                          Loads X(I), U(I,J) and stores X(I)
   END DO
```

2. Inner products:

```
   DO I = N,1,-1
      S = 0
      DO J = (I+1),N
         S = S + U(I,J)*X(J)     <---Addresses [U] by rows
      END DO                          Loads X(J), U(I,J) and no stores.
     X(I) = X(I) - S
   END DO
```

The outer product has an advantage because it addresses the matrix **[U]** by columns, which is how the data is stored. However it requires 3 memory references, including a store, per cycle of the inner loop.
The inner product has an advantage because it requires only 2 memory references and no store per cycle of the inner loop. However the matrix **[U]** is addressed by columns, which is across the direction of storage.

To overcome the addressing difficulties of the dot product, it is possible to build a temporary vector containing the ith row of **[U]**. Then the algorithm proceeds with the advantage of the dot product and incremental addressing. This strategy only works if there are several vectors being processed (multiple RHS's) to amortize the cost of loading the temporary vector with the row of **[U]**.

This is one of those fine tuning parameters that is problem and machine dependent. It is recommended that you use the default value unless you are processing a large number of solution vectors on a small memory machine.

Copyright © Multipath Corporation

D.8.2. FMS MATRIX SOLVING PARAMETER LUTRAN

Data Type

Integer

Default Value

0

Description

This parameter is used to modify the solution process by using **[A]T** instead of **[A]**. The following options are available:

- 0, Solve **[A]{X} = {B}**
- 1, Solve **[A]T {X} = {B}**

This Parameter applies only to nonsymmetric matrices (NOTE: For symmetric matrices **[A]T = [A]**).

Copyright © Multipath Corporation

D.9. FMS MATRIX MULTIPLY PARAMETERS

Matrix Multiply Parameters

Index	NAME	Function
D.9.1	MMROW	Shifts the product of **[A][B]** by MMROW rows in **[C]**.
D.9.2	MMCOL	Shifts the product of **[A][B]** by MMCOL columns in **[C]**.
D.9.3	MMKA	Adds MMKA columns of 0's to the beginning of matrix **[A]**.
D.9.4	IACCUM	Accumulate results of product in output matrix.

Copyright © Multipath Corporation

D.9.1. FMS MATRIX MULTIPLY PARAMETER MMROW

Data Type

Integer

Default Value

0

Description

When using subroutine FMSMM to compute the matrix product

```
[C] = [D] + [A][B]
```

it is sometimes necessary to shift the product **[A][B]** before storing into **[C]**. This parameter allows the product of **[A][B]** to be shifted by MMROW rows.

Only the part of the product **[A][B]** that falls within the matrix **[C]** is computed and stored.

Copyright © Multipath Corporation

D.9.2. FMS MATRIX MULTIPLY PARAMETER MMCOL

Data Type

Integer

Default Value

0

Description

When using subroutine FMSMM to compute the matrix product

```
[C] = [D] + [A][B]
```

it is sometimes necessary to shift the product **[A][B]** before storing into **[C]**. This parameter allows the product of **[A][B]** to be shifted by MMCOL columns.

Only the part of the product **[A][B]** that falls within the matrix **[C]** is computed and stored.

Copyright © Multipath Corporation

D.9.3. FMS MATRIX MULTIPLY PARAMETER MMKA

Data Type

Integer

Default Value

0

Description

When using subroutine FMSMM to compute the matrix product

```
[C] = [D] + [A][B]
```

it is sometimes necessary to shift the matrix **[A]** before computing the product **[A][B]**. This parameter allows **[A]** to be shifted by MMKA columns before multiplying with **[B]**. The effect is as if MMKA columns of 0's were added to the beginning of **[A]**, to form **[0|A]**.

Copyright © Multipath Corporation

D.9.4. FMS MATRIX MULTIPLY PARAMETER IACCUM

Data Type

Integer

Default Value

0

Description

This parameter is used to modify how ***FMS*** stores the results of some multiply operations as follows:

```
IACCUM = -1, [C] = [C] - [A][B],
IACCUM =  0, [C] = [A][B],
IACCUM = +1, [C] = [C] + [A][B]   .
```

By default (IACCUM=0), the contents of the product **[A][B]** are overwritten on the output matrix **[C]**. If IACCUM does not equal 0, the product is added or subtracted from **[C]**, according to the sign of IACCUM.

Currently this Parameter is used by the Vectors-Vectors multiply routines RSDVVM, RNDVVM, CHDVVM, CSDVVM, CNDVVM.

Copyright © Multipath Corporation

D.10. FMS PRINTING PARAMETERS

Printing Parameters

Index	NAME	Function
D.10.1	PCODES	Print Code Summary
D.10.2	PRINT	Set all print codes
D.10.3	IPRI	Initialization print code
D.10.4	IPRA	Assembly print code
D.10.5	IPRF	Factoring print code
D.10.6	IPRS	Solution print code
D.10.7	IPRMV	Matrix-vectors multiply print code
D.10.8	IPRVV	Vectors-vectors multiply print code
D.10.9	IPRDV	Vectors-diagonal-vectors multiply print code
D.10.10	IPRDBG	FMS internal debug print flags
D.10.11	LOGTIM	Accumulate statistics on ***FMS*** subroutine usage
D.10.12	NEQPR1	Lower limit on equation to print
D.10.13	NEQPR2	Upper limit on equation to print
D.10.14	NROWPG	Number of rows per page
D.10.15	NCOLPG	Number of columns per page
D.10.16	IFMAT	Format for printing integers
D.10.17	RFMAT	Format for printing real numbers
D.10.18	CFMAT	Format for printing complex numbers
D.10.19	SHOW	Display reports
D.10.20	RESET	Reset ***FMS*** parameters to default values
D.10.21	LUINPT	Input unit number

D.10.22	LUPR	Output unit number

Copyright © Multipath Corporation

D.10.1. FMS PRINT CODE SUMMARY

Data Type

Integer

Default Value

2

Description

To determine the value of a print code, select the options from the table below. Then add the corresponding print code values for the options selected. Because these values are a power of 2, they have the effect of setting the appropriate bits in the print code parameter. For example, setting IPRS to 34 will print out the default headings (IPRS=2) and the solution vector (IPRS=64). You may set these individual print code to any value or set them all to the same value with the PRINT parameter.

CODE	IPRI	IPRA	IPRF	IPRS	IPRMV	IPRVV	IPRDV
0	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing	Nothing
1	Start & Stop Message	Start & Stop Message	Start & Stop Message	Start & Stop Message	Start & Stop Message	Start & Stop Message	Start & Stop Message
2	Matrix Blocking Information	Segment Group Timing	Segment Group Timing	Vector Group Timing	Vector Group Timing	Vector Group Timing	Vector Group Timing
4	- - -	{RHS}	{RHS}	{RHS}	{X}	{X},{Y}	{X},{Y}
8	- - -	[Si],{Vi}	Reduced	Reduced	[F]	[F]	[F]
16	Profile from FMSPF	- - -	Scaled	Scaled	[D]{X}	- - -	{X}T[D]
32	- - -	- - -	Pivoted Equations	{SOL}	{Y}	- - -	- - -
64	Profile you specified	[D]	[D]	- - -	[D]	- - -	[D]
128	Profile after	- - -	[1/D]	[1/D]	- - -	- - -	- - -

	blocking						
256	Block Selection	**[AL], [AU]**	**[AL], [AU]**	- - -	**[AL], [AU]**	- - -	- - -
512	- - -	- - -	**[L], [U]**	**[L], [U]**	- - -	- - -	- - -
1024	Analysis of Profile	- - -	Actual Mflops	Actual Mflops	Actual Mflops	Actual Mflops	Actual Mflops
2048	- - -	- - -	Parallel Scaling	- - -	- - -	- - -	- - -
4096	- - -	User & System Times	User & System Times	User & System Times	User & System Times	User & System Times	User & System Times
8192	Write Profile and quit	ATERMS	Density	- - -	- - -	- - -	- - -
16384	- - -	I/O Hist.	I/O Hist.	I/O Hist.	I/O Hist.	- - -	- - -

Copyright © Multipath Corporation

D.10.2. FMS PRINTING PARAMETER PRINT

Data Type

Integer

Default Value

2

Description

Sets all the print codes (IPRI, IPRA, IPRF, IPRS, IPRMV, IPRVV and IPRDV) to the specified value.

Copyright © Multipath Corporation

D.10.3. FMS PRINTING PARAMETER IPRI

Data Type

Integer

Default Value

2

Description

This parameter controls the amount of printed output produced by calls to the matrix initialization routines RSDI, RNDI, CHDI, CSDI, CNDI and RSDANN, RNDANN, CHDANN, CSDANN, CNDANN. The print code summary table lists the values of IPRI and the output produced.

The following is a detailed description of each option. The value of IPRI you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRI. The following options are available:

- 0, No output from matrix initialization is printed.
- +1, Start and stop timing is printed.

The following is an example:

```
 START OF FMS MATRIX INITIALIZATION
 END   OF FMS MATRIX INITIALIZATION.
 TIME(SEC): CPU   =       0.055,        WALL=       0.330
```

- +2, Information about the matrix blocking is printed.

The following is an example:

```
     M A T R I X     I N I T I A L I Z A T I O N

 NEW FILE BEING OPENED.
 +---------------------+---------------------+---------------------
+
 |        MATRIX       |       BLOCKING      |         FILE
|
 +---------------------+---------------------+---------------------
+
 | Data Type =     Real | Format    =  Profile | Name      =      LUA
|
 | Symmetric =      Yes | Segments  =       22 | Location  =     Disk
|
```

```
 | Equations =     1000 | Pattern   =   Inside | Size (Mb) =    0.844
|
 | Avg. Band =       96 | Cache Blk.=        4 | Rec.(R*8) =     4608
|
 | Max. Band =      100 | Stride    =        1 | # Rec.[L] =       22
|
 +----------------------+----------------------+----------------------
+

 <---SEGMENTS--->     <------EQUATIONS----->
 NUMBER   COUPLED     FIRST    LAST   LENGTH     DATA       TOTAL
      1         1         1      92       92        4271     4608
      2         1        93     136       44        4341     4608
      3         1       137     180       44        4367     4608
      4         1       181     224       44        4367     4608
      5         2       225     268       44        4367     4608
      6         3       269     312       44        4367     4608
      7         4       313     356       44        4367     4608
      8         5       357     400       44        4367     4608
      9         6       401     444       44        4367     4608
     10         7       445     488       44        4367     4608
     11         8       489     532       44        4367     4608
     12         9       533     576       44        4367     4608
     13        10       577     620       44        4367     4608
     14        11       621     664       44        4367     4608
     15        12       665     708       44        4367     4608
     16        13       709     752       44        4367     4608
     17        14       753     796       44        4367     4608
     18        15       797     840       44        4367     4608
     19        16       841     884       44        4367     4608
     20        17       885     928       44        4367     4608
     21        18       929     972       44        4367     4608
     22        19       973    1000       28        2779     4608
```

- +16, The matrix profile renumbering is printed from subroutine FMSPF.
- +64, The profile you specified will be printed.

CAUTION: For large problems this could be a lot of output.

- +128, The profile is printed after it has been expanded for outside blocking.

This option is only useful for profile matrices (MFMAT=1) with outside blocking (INSIDE=-1).

- +256, The criteria for selecting block dimensions is printed.

This criteria depends on the optimized kernel size NEQBLK, the matrix format MFMAT, the amount of memory MAXMD, the number of processors MAXCPU and the file striping configuration.

- +1024, A report is printed that evaluates your profile for various

compute block sizes NEQBLK and blocking patterns INSIDE. This applies to profile format matrices MFMAT=1 only. It may be useful for fine tuning a specific problem to a specific platform.

- +2048, The matrix profile is written to file FMSLEQ and the program halts.
This profile may later be read into program DEMO with the SPARSITY=PROFILE option. Program DEMO is useful for interactively changing ***FMS*** parameters for fine tuning.

Copyright © Multipath Corporation

D.10.4. FMS PRINTING PARAMETER IPRA

Data Type

Integer

Default Value

2

Description

This parameter controls the amount of printed output produced by calls to the matrix and vector assembly routines RSDA, RNDA, CHDA, CSDA, CNDA, RSDAF, RNDAF, CHDAF, CSDAF, CNDAF, RSDANN, RNDANN, CHDANN, CSDANN, CNDANN and RSDAV, CSDAV The print code summary table lists the values of IPRA and the output produced.

The following is a detailed description of each option. The value of IPRA you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRA.

- 0, No output from matrix assembly is printed.
- +1, Start and stop timing is printed.

The following is an example:

```
 START OF FMS MATRIX ASSEMBLY
 END   OF FMS MATRIX ASSEMBLY.
 TIME(SEC): CPU   =       0.314,        WALL=       0.340
```

- +2, Timing information is printed for the assembly of each matrix segment.

The following is an example:

```
 F M S     M A T R I X     A S S E M B L Y
 <----FUNCTION--->   <-EQUATIONS->     <-CPU  TIME(SEC)->     <-WALL
TIME(SEC)->
                     FIRST    LAST     DELTA        TOTAL     DELTA
TOTAL
 MATRIX ASSEMBLY         1     224     0.057        0.057     0.096
0.096
 MATRIX ASSEMBLY       225     400     0.054        0.110     0.069
0.165
 MATRIX ASSEMBLY       401     576     0.052        0.162     0.070
0.235
```

```
 MATRIX ASSEMBLY       577     752     0.048        0.210     0.062
0.298
 MATRIX ASSEMBLY       753     928     0.062        0.272     0.075
0.373
 MATRIX ASSEMBLY       929    1000     0.026        0.299     0.044
0.417
```

- +4, Right-hand side vectors **{RHS}** are printed after assembly.
- +8, The element matrices **[Si]** and element vectors **{Vi}** are printed before assembly.
- +64, The matrix diagonals **[D]** are printed after assembly.
- +256, The lower triangle matrix **[AL]** and upper triangle matrix **[AU]** (nonsymmetric) are printed after assembly.
- +2048, The number of terms within the matrix profile are counted. This value becomes available as the ***FMS*** parameter ATERMS.

Copyright © Multipath Corporation

D.10.5. FMS PRINTING PARAMETER IPRF

Data Type

Integer

Default Value

2

Description

This parameter controls the amount of printed output produced by calls to the matrix factoring routines RSDAF, RNDAF, CHDAF, CSDAF, CNDAF, RSDF, RNDF, CHDF, CSDF, CNDF and RSDEX, RNDEX, CHDEX, CSDEX, CNDEX. The print code summary table lists the values of IPRF and the output produced.

The following is a detailed description of each option. The value of IPRF you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRF.

- 0, No output from matrix factoring is printed.
- +1, Start and stop timing is printed.

The following is an example:

```
 START OF FMS MATRIX FACTORING
 END   OF FMS MATRIX FACTORING.
 TIME(SEC): CPU   =       0.740,        WALL=       0.930
```

- +2, Timing information for each matrix segment is printed.

The output also includes the inverse condition number (also available as ***FMS*** Parameter CONINV). The following is an example:

```
 F M S     M A T R I X     F A C T O R I N G
 <----FUNCTION--->   <-EQUATIONS->     <-CPU  TIME(SEC)->     <-WALL
TIME(SEC)->
                     FIRST    LAST     DELTA        TOTAL     DELTA
TOTAL
 MATRIX FACTORING        1     224     0.144        0.144     0.325
0.325
 MATRIX FACTORING      225     400     0.134        0.278     0.152
0.477
 MATRIX FACTORING      401     576     0.125        0.403     0.138
0.615
 MATRIX FACTORING      577     752     0.129        0.532     0.142
```

```
0.757
 MATRIX FACTORING      753     928     0.142        0.673     0.155
0.912
 MATRIX FACTORING      929    1000     0.071        0.745     0.087
0.999
 Inverse condition number...........= 0.0008333333
```

- +4, Right-hand side vectors **{RHS}** are printed before forward reduction.
- +8, Right-hand side vectors are printed after they have been reduced by the lower triangle factor, **[L]-1{RHS}**.
- +16, The reduced vectors are printed after diagonal scaling, **[1/D][L]-1{RHS}** (symmetric matrices only).
- +32, Information is printed about each equation that is pivoted. This applies to slab matrices (MFMAT=3) only.
  CAUTION: This can produce a line of output for each equation.
- +64, The matrix diagonals **[D]** are printed before factoring.
- +128, The diagonal factor reciprocals **[1/D]** are printed after factoring.
- +256, The lower triangle matrix **[AL]** and upper triangle matrix **[AU]** (nonsymmetric) are printed before factoring.
- +512, The lower triangle factor **[L]** and upper triangle factor **[U]** (nonsymmetric) are printed after factoring.
- +1024, The actual performance achieved in Megaflops is printed. This computation is based on an actual count of the number of floating point operations (adds and multiplies) required for this matrix and vector(s) and the actual time required.
  NOTE: The operation count may take some time. Therefore this option is not recommended as a default for production applications. It is provided for application tuning.
- 2048, The number of matrix terms and the number of zeros before and after factoring are counted and printed. A report of the matrix density in the following form is produced:

```
Number of matrix terms.............=         95050.
Number of zeros before factoring...=         93051., ( 97) percent
Number of zeros after  factoring...=             0., (  0) percent
```

CAUTION: For large problems this parameter can produce a lot of output. You can use the ***FMS*** Parameters NEQPR1 and NEQPR2 to create a window on the matrix and vector terms printed.

Copyright © Multipath Corporation

D.10.6. FMS PRINTING PARAMETER IPRS

Data Type

Integer

Default Value

2

Description

This parameter controls the amount of printed output produced by calls to the vector solution routine RSDS, RNDS, CHDS, CSDS, CNDS. The print code summary table lists the values of IPRS and the output produced.

The following is a detailed description of each option. The value of IPRS you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRS.

- 0, No output from vector solution is printed.
- +1, Start and stop timing is printed.

The following is an example:

```
 START OF FMS VECTOR SOLUTION
 END   OF FMS VECTOR SOLUTION.
 TIME(SEC): CPU   =       4.301,        WALL=       4.651
```

- +2, Timing information for each vector group is printed.

The following is an example:

```
 F M S     V E C T O R     S O L U T I O N
 <----FUNCTION--->   <--VECTORS-->     <-CPU  TIME(SEC)->     <-WALL
TIME(SEC)->
                     FIRST    LAST     DELTA        TOTAL     DELTA
TOTAL
 FORWARD REDUCTION       1      51     0.845        0.845     0.974
0.974
 DIAGONAL  SCALING       1      51     0.017        0.862     0.030
1.004
 BACK-SUBSTITUTION       1      51     0.803        1.665     0.820
1.824
 FORWARD REDUCTION      52     100     1.370        3.035     3.578
5.403
 DIAGONAL  SCALING      52     100     0.016        3.051     0.029
```

```
5.432
 BACK-SUBSTITUTION      52     100     0.792        3.842     0.807
6.239
```

- +4, Right-hand side vectors **{RHS}** are printed before forward reduction.
- +8, Right-hand side vectors are printed after they have been reduced by the lower triangle factor **[L]-1{RHS}**.
- +16, The reduced vectors are printed after diagonal scaling **[1/D][L]-1{RHS}** (symmetric matrices only).
- +32, The solution vectors **{SOL}** are printed after back-substitution.
- +128, The diagonal factor reciprocals **[1/D]** are printed.
- +512, The lower triangle factor **[L]** and upper triangle factor **[U]** (nonsymmetric) are printed.
- +1024, The actual performance achieved in Megaflops is printed. This computation is based on an actual count of the number of floating point operations (adds and multiplies) required and the actual time spent.
  NOTE: The operation count may take some time. Therefore this option is not recommended as a default for production applications. It is provided for application tuning.

CAUTION: For large problems this parameter can produce a lot of output. You can use the ***FMS*** Parameters NEQPR1 and NEQPR2 to create a window on the matrix and vector terms printed.

Copyright © Multipath Corporation

D.10.7. FMS PRINTING PARAMETER IPRMV

Data Type

Integer

Default Value

2

Description

This parameter controls the amount of printed output produced by calls to the matrix-vector multiply routines RSDMVM, RNDMVM, CHDMVM, CSDMVM, CNDMVM, RSDSVM, CHDSVM, CSDSVM, and RNDVMM, CNDVMM. The print code summary table lists the values of IPRMV and the output produced.

The following is a detailed description of each option. The value of IPRMV you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRMV.t. Because these numeric values are a power of 2, they have the effect of setting bits in IPRS.

- 0, No output from matrix-vector multiply is printed.
- +1, Start and stop timing is printed.

The following is an example:

```
 START OF FMS MATRIX-VECTORS MULTIPLY
 END   OF FMS MATRIX-VECTORS MULTIPLY.
 TIME(SEC): CPU   =       4.941,        WALL=       4.985
```

- +2, Timing information for each vector group is printed.

The following is an example:

```
 F M S     M A T R I X   -   V E C T O R S     M U L T I P L Y
 <----FUNCTION--->   <--VECTORS-->     <-CPU  TIME(SEC)->     <-WALL
TIME(SEC)->
                     FIRST    LAST     DELTA        TOTAL     DELTA
TOTAL
 M*V   MULTIPLY          1      24     1.159        1.159     1.204
1.204
 M*V   MULTIPLY         25      48     1.143        2.301     1.160
2.364
 M*V   MULTIPLY         49      72     1.166        3.468     1.184
3.548
 M*V   MULTIPLY         73      96     1.177        4.645     1.199
```

```
4.747
 M*V   MULTIPLY         97     100     0.288        4.933     0.303
5.050
```

- +4, The **{X}** vectors are printed before matrix multiplication.
- +8, The **[F]** matrix is printed before matrix multiplication. (RNDVMM, CNDVMM only).
- +16, The **{X}** vectors are printed after diagonal multiplication, **[D]{X}**.
- +32, The output **{Y} = [A]{X}** vectors are printed after matrix multiplication.
- +64, The diagonals **[D]** are printed.
- +256, The lower triangle matrix **[AL]** and upper triangle matrix **[AU]** (nonsymmetric) are printed.
- +1024, The actual performance achieved in Megaflops is printed. This computation is based on an actual count of the number of floating point operations (adds and multiplies) required and the actual time spent.
  NOTE: The operation count may take some time. Therefore this option is not recommended as a default for production applications. It is provided for application tuning.

CAUTION: For large problems this parameter can produce a lot of output. You can use the ***FMS*** Parameters NEQPR1 and NEQPR2 to create a window on the matrix and vector terms printed.

Copyright © Multipath Corporation

D.10.8. FMS PRINTING PARAMETER IPRVV

Data Type

Integer

Default Value

2

Description

This parameter controls the amount of printed output produced by calls to the vector-vector multiply routine RSDVVM, RNDVVM, CHDVVM, CSDVVM, CNDVVM. The print code summary table lists the values of IPRVV and the output produced.

The following is a detailed description of each option. The value of IPRVV you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRVV.

- 0, No output from vector-vector multiply is printed.
- +1, Start and stop timing is printed.

The following is an example:

```
 START OF FMS VECTORS-VECTORS MULTIPLY
 END   OF FMS VECTORS-VECTORS MULTIPLY.
 TIME(SEC): CPU   =       2.435,        WALL=       2.638
```

- +2, Timing information for each vector group is printed.

The following is an example:

```
 F M S     V E C T O R S   -   V E C T O R S     M U L T I P L Y
 <----FUNCTION--->   <--VECTORS-->     <-CPU  TIME(SEC)->     <-WALL
TIME(SEC)->
                     FIRST    LAST     DELTA        TOTAL     DELTA
TOTAL
 V*V   MULTIPLY          1      55     1.670        1.670     1.722
1.722
 V*V   MULTIPLY         56     100     0.740        2.410     0.756
2.478
```

- +4, The **{X}** and **{Y}** vectors are printed before multiplication.
- +8, The initialization matrix **{D}** is printed before multiplication. The output matrix **{F}** is printed after multiplication.

- +1024, The actual performance achieved in Megaflops is printed. This computation is based on an actual count of the number of floating point operations (adds and multiplies) required and the actual time spent.

CAUTION: For large problems this parameter can produce a lot of output. You can use the ***FMS*** Parameters NEQPR1 and NEQPR2 to create a window on the matrix and vector terms printed.

Copyright © Multipath Corporation

D.10.9. FMS PRINTING PARAMETER IPRDV

Data Type

Integer

Default Value

2

Description

This parameter controls the amount of printed output produced by calls to the vector-diagonal-vector multiply routine RSDDVM, RNDDVM, CHDDVM, CSDDVM, CNDDVM. The print code summary table lists the values of IPRDV and the output produced.

The following is a detailed description of each option. The value of IPRDV you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRDV.

- 0, No output from vector-diagonal-vector multiply is printed.
- +1, Start and stop timing is printed. The following is an example:

```
 START OF VECTORS-DIAGONAL-VECTORS MULTIPLY
 END   OF VECTORS-DIAGONAL-VECTORS MULTIPLY.
 TIME(SEC): CPU   =       2.484,        WALL=       2.516
```

- +2, Timing information for each vector group is printed. The following is an example:

```
 F M S     V E C T O R S   -   D I A G.   -   V E C T O R S     M U L
T I P L Y
 <----FUNCTION--->   <--VECTORS-->     <-CPU  TIME(SEC)->     <-WALL
TIME(SEC)->
                     FIRST    LAST     DELTA        TOTAL     DELTA
TOTAL
 V*D*V MULTIPLY          1      54     1.668        1.668     1.749
1.749
 V*D*V MULTIPLY         55     100     0.795        2.463     0.849
2.598
```

- +4, The **{X}** and **{Y}** vectors are printed before multiplication.
- +8, The output matrix **[F] = {X}T[D]{Y}** is printed.
- +16, The **{X}** vectors are printed after diagonal multiplication,

{X}T[D].

- +64, The diagonal matrix **[D]** is printed.
- +1024, The actual performance achieved in Megaflops is printed. This computation is based on an actual count of the number of floating point operations (adds and multiplies) required and the actual time spent.

CAUTION: For large problems this parameter can produce a lot of output. You can use the ***FMS*** Parameters NEQPR1 and NEQPR2 to create a window on the matrix and vector terms printed.

Copyright © Multipath Corporation

D.10.10. FMS PRINTING PARAMETER IPRDBG

Data Type

Integer

Default Value

0

Description

This parameter provides internal details on ***FMS*** operations. It can produce a lot of output so the problem size should be made as small as possible.

The following is a detailed description of each option. The value of IPRDBG you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in IPRDBG.

- 0, No debug output is printed.
- +1, Memory initialization and release.
- +2, Thread creation and termination.
- +4, File open and close.
- +8, Read operations.
- +16, Write operations.
- +32, Semaphore creation and deletion.
- +64, Semaphore operations.

Note: This option must be selected to accumulate history, if you want to use SHOW=HISTORY.

- +128, Polling.
- +256, Semaphore history.
- +512, Memory allocation and deallocation.
- +1024, BLAS Library version.
- +2048, Output HTML for benchmark report.
- +4096, Calls to FMSIGT, FMSRGT, FMSCGT, FMSIST, FMSRST and FMSCST
- +8192, Calls to malloc and free

Copyright © Multipath Corporation

D.10.11. FMS PRINTING PARAMETER LOGTIM

Data Type

Integer

Default Value

11

Description

This parameter is used to accumulate timing statistics on individual ***FMS*** and user subroutines.

The following is a detailed description of each option. The value of LOGTIM you specify is obtained by adding together the numeric value of each option you select. Because these numeric values are a power of 2, they have the effect of setting bits in LOGTIM.

- 0, Do not accumulate times.
In order to completely turn off timing, you must also specify all the print codes to 0 by setting the ***FMS*** Parameter PRINT = 0.
- +1, Accumulate times for high level ***FMS*** routines called by your application.
- +2, Accumulate times for low level ***FMS*** subroutines.
These include:
  - Matrix triple products RSDWTP and RNDWTP
  - Matrix multiply
  - Triangle solve
  - Diagonal block factoring
  - Counting the number of operations saved by reduced operation algorithms. These algorithms are selected by the ***FMS*** Parameter IALGOR. The number of saved operations is available in the SAVOPS Parameter.
- +4, Print a flowchart as each enabled subroutine is called. This flowchart is suspended during parallel execution.
- +8, Accumulate times for your subroutines which contain calls to FMSPSH and FMSPOP.

Copyright © Multipath Corporation

>

D.10.12. FMS PRINTING PARAMETER NEQPR1

Data Type

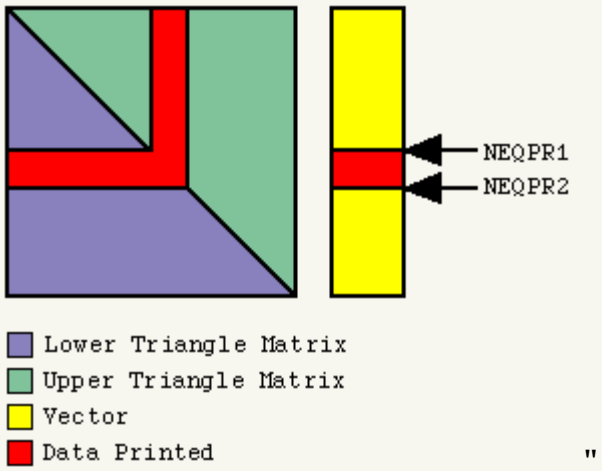
Integer

Default Value

1

Description

This parameter is used to limit the amount of information displayed by the print codes.



"

For vector data, it sets the first element of the vector that prints. For lower triangle matrix data, it sets the first row that prints. For upper triangle matrix data, it sets the first column that prints.

Copyright © Multipath Corporation

D.10.13. FMS PRINTING PARAMETER NEQPR2

Data Type

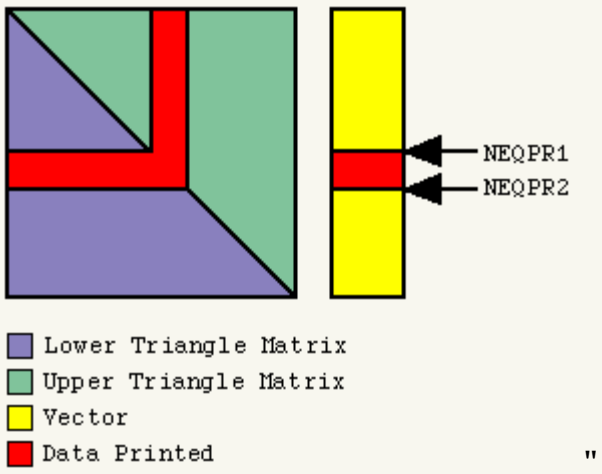
Integer

Default Value

2147483647 which is (2**31 - 1), a big number.

Description

This parameter is used to limit the amount of information displayed by the print codes.



"

For vector data, it sets the last element of the vector that prints. For lower triangle matrix data, it sets the last row that prints. For upper triangle matrix data, it sets the last column that prints.

Copyright © Multipath Corporation

D.10.14. FMS PRINTING PARAMETER NROWPG

Data Type

Integer

Default Value

50

Description

This parameter controls the number of rows per page of printed output. You may set it to a large number to turn off page breaks in the output.

Copyright © Multipath Corporation

D.10.15. FMS PRINTING PARAMETER NCOLPG

Data Type

Integer

Default Value

80

Description

This parameter controls the number of columns per page of printed output. If you are printing vector and matrix data to a printer with a wide (landscape) format, you could set NCOLPG to 132.

Copyright © Multipath Corporation

D.10.16. FMS PRINTING PARAMETER IFMAT

Data Type

Character string

Default Value

(I7)

Description

This parameter controls the format used to print integer data. You may change it to any valid FORTRAN format descriptor used to print integers.

NOTE: Be sure and include the parenthesis and character string quotations in the format descriptor when calling subroutine FMSCST. When setting IFMAT in the license file or with FMSSET, the string quotations are not required.

EXAMPLES:

To change IFMAT with a subroutine call,

```
       CALL FMSCST ('IFMAT','(I9)')
```

To change IFMAT using FMSSET,

```
FMSSET>IFMAT=(I9)
```

Copyright © Multipath Corporation

D.10.17. FMS PRINTING PARAMETER RFMAT

Data Type

Character string

Default Value

(E12.4)

Description

This parameter controls the format used to print real data. You may change it to any valid FORTRAN format descriptor used to print real data. For example, if you are printing matrix data and want to find out if the terms are zero or not, you might change it to (E9.1). On the other hand if you are printing data and are concerned about the least significant digits, you might select (E23.15). Naturally the more information you print out about each term, the more space (screen or paper) is used.

NOTE: Be sure and include the parenthesis and character string quotations in the format descriptor when calling subroutine FMSCST. When setting RFMAT in the license file or with FMSSET, the string quotations are not required.

EXAMPLES:

To change RFMAT with a subroutine call,

```
      CALL FMSCST ('RFMAT','(E9.1)')
```

To change RFMAT using FMSSET,

```
FMSSET>RFMAT=(E9.1)
```

Copyright © Multipath Corporation

D.10.18. FMS PRINTING PARAMETER CFMAT

Data Type

Character string

Default Value

(E12.4,1H,,E11.4)

Description

This parameter controls the format used to print complex data. You may change it to any valid FORTRAN format descriptor used to print complex data. For example, if you are printing matrix data and want to find out if the terms are zero or not, you might change it to (E9.1,1H,,E8.1). On the other hand if you are printing data and are concerned about the least significant digits, you might select (E23.15,1H,,E22.15). Naturally the more information you print out about each term, the more space (screen or paper) is used.

NOTE: Be sure and include the parenthesis and character string quotations in the format descriptor when calling subroutine FMSCST. When setting CFMAT in the license file or with FMSSET, the string quotations are not required. Also be sure and use a format descriptor that accounts for the real and imaginary parts of the complex number, as well as something to separate them.

EXAMPLES:

To change CFMAT with a subroutine call,

```
      CALL FMSCST ('CFMAT','(E9.1,1H,,E8.1)')
```

To change CFMAT using FMSSET,

```
FMSSET>CFMAT=(E9.1,1H,,E8.1)
```

Copyright © Multipath Corporation

D.10.19. FMS PRINTING PARAMETER SHOW

Data Type

Character string

Default Value

None. This is an input parameter only.

Description

This directive may be used to obtain various reports on the current status of ***FMS***. You may obtain a list of the available reports by entering SHOW at the FMSSET> prompt as follows:

```
FMSSET>show
SHOW What?
Enter one of the following:
SHOW=PARAMETERS for FMS parameter values
SHOW=MODIFIED   for modified FMS parameters only
SHOW=MEMORY     for status of memory allocation
SHOW=MALLOC     for status of malloced memory
SHOW=FILES      for status of opened files
SHOW=FMSDISK    for file systems used
SHOW=CALLS      for subroutines called and I/O report
SHOW=PROBLEM    for maximum problem size parameters
SHOW=SEMAPHORES for status of semaphores
SHOW=STACK      for subroutine call stack
SHOW=SEMHISTORY for history of semaphores
SHOW=IOQUEUE    for status of I/O queue
SHOW=USAGE      for usage statistics
SHOW=PLACEMENT  for detail NUMA memory placement
SHOW=PLACEMENTSUMMARY for NUMA placement summary
SHOW=LIBRARIES  for shared object libraries
SHOW=GPU
FMSSET>
```

At the FMSSET> prompt you can obtain any of the reports by entering

`SHOW=`*`value`*

where *value* is the name of one of the reports.

You can also print any of the reports from within your application by entering the line

```
CALL FMSCST('SHOW', 'value')
```

For example you can show the current memory allocation with

```
CALL FMSCST('SHOW', 'MEMORY')
```

or the current open files with

```
CALL FMSCST('SHOW', 'FILES')
```

You can also generate WEB pages that contain this (and other) information concurrently with your run by including IWATCH in your License File.

Copyright © Multipath Corporation

D.10.20. FMS PRINTING PARAMETER RESET

Data Type

Character string

Default Value

None. This is an input parameter only.

Description

This parameter resets the ***FMS*** Parameters to their default values. The following options are available:

- `0, Reset all parameters.`
- `1, Reset all parameters except the unit numbers LUINPT, LUPR and LUERR.`

Copyright © Multipath Corporation

D.10.21. FMS PRINTING PARAMETER LUINPT

Data Type

Integer

Default Value

5

Description

FORTRAN unit number from which input to ***FMS*** is read. This value can only be changed to an open file.

Copyright © Multipath Corporation

D.10.22. FMS PRINTING PARAMETER LUPR

Data Type

Integer

Default Value

6

Description

FORTRAN unit number to which print output is written. This value can only be changed to an open file.

Copyright © Multipath Corporation

D.11. DASHBOARD PERFORMANCE REPORT PARAMETERS

Dashboard Performance Report Parameters

Index	NAME	Function
D.11.1	MINDB	Minimum display width (pixels)
D.11.2	MAXDB	Maximum display width (pixels)
D.11.3	MTSIZE	Maximum matrix table size (pixels)
D.11.4	IWATCH	Master option selector - bit masks
D.11.5	NSUPD	Frequency of regenerating pages.
D.11.6	NSREF	Frequency for browser to reload pages.
D.11.7	NSMOVE	Time between movie frames.
D.11.8	MAXMOV	Maximum number of movie frames.

Copyright © Multipath Corporation

D.11.1. DASHBOARD PARAMETER MINDB

Data Type

Integer

Units

Pixels

Default Value

500

Description

This Parameter sets the minimum width of the display area. If the browser shrinks the display area below this value, the display is clipped.

Copyright © Multipath Corporation

D.11.2. DASHBOARD PARAMETER MAXDB

Data Type

Integer

Units

Pixels

Default Value

1000

Description

This Parameter sets the maximum width of the display area. If the browser expands the display width beyond this value, the additional area is filled with the background color.

Copyright © Multipath Corporation

D.11.3. DASHBOARD PARAMETER MTSIZE

Data Type

Integer

Units

Pixels

Default Value

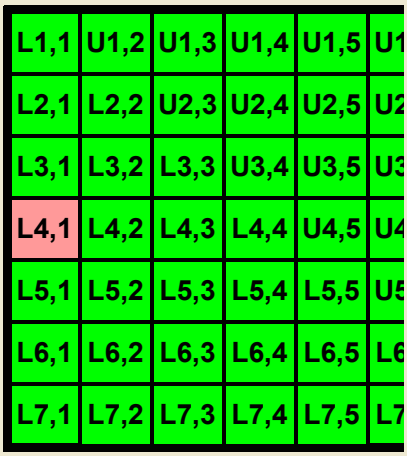
500

Description

This Parameter sets the maximum width of the matrix table. If the table is larger than this area, a horizontal scroll bar is added.

Legend

Undefined	Assembled	Factored	Reduced
Solved	Multiplied	Accessing	Computing



A matrix table is used to graphically show the data that is currently being computed in the context of the problem being solved. For matrix factoring, this is a picture of the matrix as divided into the blocks used for processing. For other computations, including vector solution and matrix multiply, a picture of the matrix and vectors is displayed.

As the calculation proceeds, the status (color, label) of the blocks is changed.

You can use the scroll bar to examine the part of the display not currently in view. This example is performing forward reduction. The factored block $L_{4,1}$ is being multiplied with the reduced vector block $R_{1,1}$ and subtracted from the vector block $M_{4,1}$.

Copyright © Multipath Corporation

D.11.4. DASHBOARD PARAMETER IWATCH

Data Type

Integer

Default Value

0 for ***FMS***lib,

1+2+32+64=99 for **MatrixWarrior**

Description

This is the master flag for activating the **Dashboard** performance reports, which generates WEB HTML pages showing the details of your machine and application as it runs. The options set bit flags. Select the value of IWATCH by adding the individual options you want. The following options are available:

- 0
  Do not generate a performance report.
- 1
  Generate and update performance report pages for the current running job.
  New pages overwrite old pages. Where appropriate, pages are set to automatically refresh in your browser based on the amount of time since the last update. Automatic refresh is turned off for all pages when the job terminates.
- 2
  Include subprograms in the performance chart on the Performance page. These include matrix multiply, triangle solve, diagonal block factoring, etc. By default only the total performance is listed.
- 4
  Archive the last performance page at the end of each subroutine. A unique file name is generated that includes the date, time, subroutine being monitored and the HTML page name. This option can be useful if several ***FMS*** subroutines are called and you want to examine the performance at the completion of the job. Archived pages have file names in the form
  YYYYMMDDHHMM*Page*.html
- 8
  Archive all pages at the end of each subroutine.

> This is similar to option 4 except all pages are archived.
> - 16
> Generate a movie of the performance pages.
> - 32
> Generate a picture of the matrices being processed.
> - 64
> Include a table showing the time spent in the subprograms.
> - 128
> Include the most recently called function at the top of the table.
> - 256
> List the pages as they are updated in the console window.
> - 512
> Skip updating the Disks page. Gathering information from the operating system on disk usage can take time. If intermediate renderings of the Disks page are not required, setting this flag will skip updates. The Disks page is always displayed at the start and end of the job.
> - 1024
> Skip updating the GPU-Dyn page. Gathering information from nvidia-smi on GPU usage can take time. If intermediate renderings of the GPU-Dyn page are not required, setting this flag will skip updates. The GPU-Dyn page is always displayed at the start and end of the job.
> - 2048
> Execute commands at startup by calling popen to obtain system information. This results in a fork of your process, which may fail if your application has allocated a significant amount of memory. These calls occur before ***FMS*** allocated it's memory.
> - 4096
> Include the diagonal blocks in the Component Performance table.

There is a two step process for generating and displaying the performance report pages.

1. WEB page generation.
The instrumented machine generates WEB pages in HTML format to a directory you specify with the FMSHTML environment variable. If FMSHTML is not specified, the WEB pages are created in your current working directory. The content of these pages is controlled by this IWATCH Parameter. Separate pages are available which describe the machine, CPUs, GPUs, Memory, Disks, Files, Software, Call history, ***FMS*** Parameters, Performance, Usage (Linux), Links and Help. As your application runs, new data becomes available and the pages are updated. By default new pages overwrite old pages. In order to make this process more efficient, pages are updated at an interval, rather than continuously, as data changes. This interval is specified by the ***FMS*** Parameter NSUPD, which specifies the update interval in seconds.
2. WEB page display.

Using a standard WEB browser you display the WEB pages which are stored in the FMSHTML directory. This may be the same as the instrumented machine or any capable WEB device with access to the FMSHTML directory. This browser machine does not communicate back to the instrumented machine. It simply views what is written to the files. As the files are updated, the browser needs to reload the updated pages. This occurs automatically by the browser polling the files at predetermined intervals. The polling interval is set within the page when it is generated, based on the ***FMS*** Parameter NSREF and how frequently the page is being updated. You may also force a reload of the page by entering Control-r.

For a general discussion on the IWATCH performance report, see Chapter 7 of the ***FMS*** manual.

Copyright © Multipath Corporation

D.11.5. DASHBOARD PARAMETER NSUPD

Data Type

Integer

Units

Seconds

Default Value

5

Description

This Parameter determines how often pages in the IWATCH performance report get updated.

Initally all pages are generated when ***FMS*** starts. A page becomes a candidate for updating whenever one of the following occurs:

- Its contents change
- A subroutine is called or returned;
- Memory is allocated or released;
- A file is opened, closed, read or written;
- A **FMS** Parameter is changed;
- A subtask (matrix multiply, etc.) is completed.

In some cases the page is updated immediately. For less critical pages, the updates are accumulated and performed after NSUPD seconds.

The ***FMS*** Parameter IWATCH must include the 1 option for this Parameter to have any effect.

Copyright © Multipath Corporation

D.11.6. DASHBOARD PARAMETER NSREF

Data Type

Integer

Units

Seconds

Default Value

3

Description

This Parameter determines how often your browser automatically refreshes pages.

The IWATCH performance report updates the contents of the pages according to the Parameter NSUPD. However your browser will continue to display the old contents until it is *refreshed*. The WEB page contains the referesh interval, based on how often the contents change and the value of this NSREF ***FMS*** Parameter. You can force a refresh by entering Control-r.

A value of 0 results in no automatic refreshing of the pages.

At the end of the job, automatic refresh is turned off for all pages.

Copyright © Multipath Corporation

D.11.4. FMS PERFORMANCE REPORT PARAMETER NSWAIT

Data Type

Integer

Units

Tenths of a second

Default Value

0

Description

This Parameter causes a pause in execution when the Performance page is updated. It allows closer study of intermediate displays which would otherwise be overwritten during fast processing.

NOTE: The units are tenths of a second; a value of 10 causes a pause for 1 second. When this Parameter is set, the amount of time reported for HTML will be significantly larger.

Copyright © Multipath Corporation

D.11.7. DASHBOARD PARAMETER NSMOVE

Data Type

Integer

Units

Seconds

Default Value

2

Description

This Parameter sets the pause time between movie frames. When the ***FMS*** Parameter IWATCH contains the value of 16, Dashboard generates a movie of the Performance pages. This Parameter sets the pause time between frames when the movie is replayed.

Copyright © Multipath Corporation

D.11.8. DASHBOARD PARAMETER MAXMOV

Data Type

Integer

Units

Number of movie frames.

Default Value

1000

Description

This Parameter controls the maximum number of movie frames **Dashboard** will generate. When a large problem is run on a machine with a small amount of memory, the problem will be divided into a large number of blocks. Each step in the computation of a block will generate a movie frame. Under these conditions, it is possible to generate an excessive number of movie frame files. This parameter places an upper limit on the number of frames (files) generated.

Copyright © Multipath Corporation

D.12. FMS ERROR HANDLING PARAMETERS

Error Parameters

Index	NAME	Function
12.1	NERROR	***FMS*** error number. Read-only
12.2	IOSTAT	I/O status returned from last system call. Read-only
12.3	LUERR	Unit number for writing error messages
12.4	IPRERR	Information to print if an error occurs
12.5	IERROR	Action to take if a fatal error is detected

Copyright © Multipath Corporation

D.12.1. FMS ERROR PARAMETER NERROR

Data Type

Integer

Default Value

Read-only

Description

This parameter returns the status of ***FMS*** operations. It is not automatically reset after each call. Following is a list of possible values for NERROR and the error they indicate.

- 0

  No error detected.
- 1, FMS$ERR_LICENSE

  Software license violation.
- 2, FMS$ERR_PROFILE

  Error on profile vector.
- 3, FMS$ERR_SYSTEM

  Error calling system routine.
- 4, FMS$ERR_FILENAME

  Illegal file name.
- 5, FMS$ERR_CREATE

  Error creating file.
- 6, FMS$ERR_OPEN

  Error opening file.
- 7, FMS$ERR_WRITE

  Error writing to file.
- 8, FMS$ERR_READ

  Error reading from file.
- 9, FMS$ERR_WAIT

  Error waiting for I/O to complete.
- 10, FMS$ERR_EXIST

  Existing file has wrong attributes.

- 11, FMS$ERR_ALIGN

Data does not start on an appropriate address.

- 12, FMS$ERR_ARRAY

Array dimension error.

- 13, FMS$ERR_CLOSE

Error closing file.

- 14, FMS$ERR_DELETE

Error deleting file.

- 15, FMS$ERR_MEMROY

Insufficient ***FMS*** memory for requested problem.

- 16, FMS$ERR_TMRAM

Insufficient Table Memory RAM for requested problem.

- 17, FMS$ERR_MVDIF

Specified matrix and vector files are inconsistent.

- 18, FMS$ERR_MMDIF

Input and output matrix files are inconsistent.

- 19, FMS$ERR_VVDIF

Input and output vector files are inconsistent.

- 20, FMS$ERR_POSDEF

Negative pivot during factoring.

- 21, FMS$ERR_NOWORK

No work specified.

- 22, FMS$ERR_UNFACT

Attempt to solve with an unfactored matrix.

- 23, FMS$ERR_UNRED

Attempt to skip forward reduction with unreduced vectors.

- 24, FMS$ERR_SYMDIF

Inconsistent symmetry, matrix file and subroutine.

- 25, FMS$ERR_MMDIF

Inconsistent data type, matrix file and subroutine.

- 26, FMS$ERR_VVDIF

Inconsistent data type, vector file and subroutine.

- 27, FMS$ERR_REDUCED

Attempt to reduce already reduced vectors.

- 28, FMS$ERR_LUPR

Error writing to print output file LUPR

- 29, FMS$ERR_OUTSIDE

Submatrix equation outside profile.

- 30, FMS$ERR_SUBTYPE

Submatrix type inappropriate for problem.

- 31, FMS$ERR_ORDER

Submatrix equations not in required order.

- 32, FMS$ERR_LOWASM

Starting equation for matrix assembly out of range.

- 33, FMS$ERR_PIVOT

Small or zero pivot during factoring.

- 34, FMS$ERR_VALUE

Actual parameter exceeds maximum allowed.

- 35, FMS$ERR_GETMEM

Error allocating memory.

- 36, FMS$ERR_RELMEM

Error releasing memory.

- 37, FMS$ERR_XXUSUB

Test user-supplied subroutine called.

- 38, FMS$ERR_SUBEQN

Submatrix equation exceeds maximum specified.

- 39, FMS$ERR_SUBZERO

Zero submatrix record length specified.

- 40, FMS$ERR_MAXVEC

More vectors specified than file contains.

- 41, FMS$ERR_FMSROW

FMSROW and FMSCOL both called.

- 42, FMS$ERR_NOTFULL

Attempt to perform a full matrix operation on a sparse matrix.

- 43, FMS$ERR_FMSINI

FMSINI not called as first ***FMS*** subroutine.

- 44, FMS$ERR_CPUOVER

CPU task command buffer overflow.

- 45, FMS$ERR_NOASYNC

Attempt to do asynchronous I/O to a file opened for synchronous I/O.

- 46, FMS$ERR_NOSHR

Argument not in shared memory.

- 47, FMS$ERR_FILLIMIT

FMS stripe file limit exceeded.

- 48, FMS$ERR_LICVAL

Attempt to set ***FMS*** parameter beyond license maximum.

- 49, FMS$ERR_BUFFULL

All buffers full.

- 50, FMS$ERR_SEM

Error in semaphore operation.

- 51, FMS$ERR_EXTEND

Error extending file.

- 52, FMS$ERR_DEADLOCK

Deadlock - All tasks waiting on events.

- 53, FMS$ERR_FMSPUT

Bad parameter on call.

- 54, FMS$ERR_FMSGET

Bad parameter on call.

- 55, FMS$ERR_FORMAT

Matrix file in wrong format.

- 56, FMS$ERR_NOCODE

Option not implemented.

- 57, FMS$ERR_IODIRECT

Alignment or length error.

- 58, FMS$ERR_FMSSET

Bad parameter on call.

- 59, FMS$ERR_PARENT

Not called from parent.

- 60, FMS$ERR_NOTHREAD

No thread found.

- 61, FMS$ERR_RUNSYNC

Inconsistent calls to FMSRUN and FMSYNC.

- 62, FMS$ERR_GPU

Error in GPU processing.

- 63, FMS$ERR_BLAS

Error in BLAS prameter.

Copyright © Multipath Corporation

D.12.2. FMS ERROR PARAMETER IOSTAT

Data Type

Integer

Default Value

Read-only

Description

This parameter returns the status of the last system call. The values are generally machine dependent and interpreted by the ***FMS*** error exit utilities.

Copyright © Multipath Corporation

D.12.3. FMS ERROR PARAMETER LUERR

Data Type

Integer

Default Value

System dependent (usually 6 or 0)

Description

FORTRAN unit number to which error output is written. This value can only be changed to an open file.

Copyright © Multipath Corporation

D.12.4. FMS MATRIX ERROR PARAMETER IPRERR

Data Type

Integer

Default Value

3

Description

This parameter controls the amount of printed output produced when an ***FMS*** error occurs. The value specified for IPRERR is obtained by adding together the numeric value for each option selected. (Note: Since the numeric values are powers of 2, this has the effect of setting bits in IPRERR). The following options are available:

- 0, Nothing.
- +1, Error message describing the error.
List of ***FMS*** subroutines called in the order called.
Status of all ***FMS*** files and file transfers.
FMS parameters changed from default values.
- +2, Traceback showing the last ***FMS*** subroutine you called and the values of the actual parameters passed.
- +4, All ***FMS*** parameters.
- +8, Include the entries in system log file for the range of time that this job ran in the ***FMS*** output.
(Not available on all systems).
- +16, Make an entry in the system log file when this job starts, and when it completes.
(Not available on all systems.)
- +128, Factored matrix diagonals, **{1/D}**
- +512, Matrix factors **[L]** and **[U]** (nonsymmetric).

Copyright © Multipath Corporation

D.12.5. FMS MATRIX ERROR PARAMETER IERROR

Data Type

Integer

Default Value

0

Description

This parameter determines if ***FMS*** stops or continues when a fatal error occurs. The following options are available:

- 0, Call the ***FMS*** error exit routine.
Print the information requested by IPRERR, and stop.
- 1, Set the ***FMS*** error code parameter NERROR and return to your application.
If you select this option, it is your responsibility to check the value of NERROR after each call to a ***FMS*** subroutine as follows:

```
CALL FMSIGT ('NERROR', NERROR)
IF(NERROR .NE. 0) STOP
```

If NERROR is not 0 you may take whatever corrective action is necessary. However the last ***FMS*** subroutine called would not have completed its operation.

In general, it is recommended that you use the default value and let ***FMS*** handle the error.

Copyright © Multipath Corporation

D.13. FMS APPLICATION PROGRAM PARAMETERS

Application Program Parameters

Index	NAME	Function
D.13.1	MAINC	Programming Language
D.13.2	ANAME	Application Name

Copyright © Multipath Corporation

D.13.1. FMS LANGUAGE PARAMETER MAINC

Data Type

Integer

Default Value

Read-only

Description

This parameter returns the language of the calling program. The following options are used:

- 0, FORTRAN
- 1, C

Copyright © Multipath Corporation

D.13.2. APPLICATION PROGRAM NAME

Data Type

Character*32

Default Value

Blank

Description

This parameter stores the name of your application. You set it as follows:

```
CALL FMSCST(ANAME, 'your application name')
```

Dashboard displays this name on the Main and Calls pages.

Copyright © Multipath Corporation

E. FMS ERROR HANDLING

FMS includes special features for handling fatal error conditions. These errors result from illegal requests for system services, illegal parameters specified on subroutine calls, and data errors.

System service errors usually result from requests for hardware that is unavailable. Examples include attempting to open a file that exceeds available disk space, transferring data to an illegal address, or requesting more memory than is currently available. The IOSTAT parameter, which is printed as part of the error message, contains the returned status for system errors.

FMS checks the actual parameters specified on subroutine calls for possible error conditions. Examples include inconsistent matrix symmetry or data type, files that contain matrices or vectors that are not at the appropriate stage of the solution process for the requested function, or array dimension errors. ***FMS*** prints the values of the actual parameters, including the file attribute lists for specified files. The addresses of the actual parameters are also printed, which can be examined to determine if input and output files are the same.

Data errors include bad pivot values during the factoring process and attempts to store matrix data outside the specified profile. ***FMS*** prints the appropriate equation numbers and data values when these errors occur.

The ***FMS*** Error Parameters control the action taken by ***FMS*** when fatal errors occur. If IERROR=0 (default), ***FMS*** prints the error message and stops execution. This is the recommended procedure for handling ***FMS*** fatal errors. All error messages are written to unit LUERR. The amount of printed output is controlled by IPRERR. The default value includes the error message and traceback, which is sufficient for production applications.

If the IERROR parameter is set to 1, ***FMS*** returns to your program when the error is detected. In general, the calculation requested will be incomplete. If you elect to handle ***FMS*** errors by specifying IERROR=1, the NERROR parameter must be examined after each call to determine if a fatal error occurred.

The following are the error condition codes that are stored in NERROR:

FMS Error Messages

NERROR	Description
0	**FMS$ERR_: NO ERROR** No error occurred.
1	**FMS$ERR_LICENSE: SOFTWARE LICENSE VIOLATION** The copy of ***FMS*** used in the program is not currently licensed to operate on this machine.
	FMS$ERR_PROFILE: ERROR ON PROFILE VECTOR

2	The profile vector terms, LOWEQ(I), must be in the range 0<LOWEQ(I)<=I. The error message includes the equation number and the profile value LOWEQ(I). For further debugging, the entire profile vector can be printed during initialization by including 64 in the print code IPRI.
3	**FMS$ERR_SYSTEM: ERROR IN SYSTEM ROUTINE** ***FMS*** has generated an error when requesting a system service. These errors usually result from file or memory management. The error message is processor dependent.
4	**FMS$ERR_FILENAME: ILLEGAL FILE NAME** The specified file name has an illegal number of characters. The error message includes the file name and its length. Valid file names can contain from 1 through 119 characters.
5	**FMS$ERR_CREATE: ERROR CREATING FILE** The requested disk file cannot be created. The error message includes the file name, record length, and number of records. If IOSTAT=0, ***FMS*** has determined that the file does not fit on the available disk space. If IOSTAT is not 0, refer to the appropriate operating system manual for your processor to determine the cause of failure. If the message "MAXIMUM NUMBER OF FMS UNITS EXCEEDED" appears, you have exceeded the number of ***FMS*** files which may be concurrently opened. You should check the subroutine call report to determine if a file is being opened in a loop without being closed. This error typically occurs on UNIX systems if there are old files left in the directories. You should clean out all directories pointed to by the links to FMSDISKijk before the run (unless permanent file storage is used). This error can also occur if the directories requested do not exist. This is especially true if the ***FMS*** file striping system is being used.
6	**FMS$ERR_OPEN: ERROR OPENING FILE** The requested disk file cannot be opened. The error message includes the file name, record length, and number of records. Examine the value of IOSTAT, and refer to the appropriate operating system manual for your processor to determine the cause of failure.
	FMS$ERR_WRITE: ERROR WRITING TO FILE The requested data cannot be transferred to the disk file. The error message includes the ***FMS*** file number, the starting disk address in words, the starting memory address in words, and the number of words to transfer. The disk address must be one or greater. The number of words to transfer must be one or greater. For some files, the disk address must start on a record boundary. The requested transfer must not exceed the allocated disk space and must be within the range of allocated memory. This error can be caused by using the incorrect disk address increment. You should verify that the disk addresses are being incremented according to the following table:

7

FILE TYPE	FILE NUMBER	RECORD LENGTH
vector file	LUX(1)	LUX(4)
submatrix matrix file	LUS(1)	LUS(3)
submatrix integer file	LUS(2)	LUS(4)
submatrix vector file	LUS(8)	LUS(25)

NOTE: Fixed length submatrix files only, LUS(7) = 0.
If the information in the error message does not indicate a problem, examine IOSTAT and refer to the appropriate operating system manual. Possible system errors include the device becoming unavailable (taken off-line) during the run.

8

FMS$ERR_READ: ERROR READING FROM FILE
The requested data cannot be transferred from the disk file. The error message and recommended action is the same as error 7, writing to file.

9

FMS$ERR_WAIT: ERROR WAITING FOR I/O TO COMPLETE
Examine the IOSTAT parameter and refer to the appropriate operating system manual for your processor to determine the cause of failure. If the problem cannot be resolved, synchronous I/O can be forced by setting the ***FMS*** Parameter IOSYNC=1.

10

FMS$ERR_EXIST: EXISTING FILE HAS WRONG ATTRIBUTES
You have requested ***FMS*** to use an existing file by setting the ***FMS*** parameter IEXIST to 1. An existing file having the same name, record length and size could not be found. This problem can be corrected by deleting the old file, selecting a different name for the new file, or changing the problem size, so the new file matches the existing file.

11

FMS$ERR_ALIGN: DATA DOES NOT START ON AN APPROPRIATE ADDRESS
FMS requires memory resident arrays, passed as actual parameters on subroutine calls, to start on appropriate address boundaries. The error message printed by ***FMS*** includes the appropriate action to be taken for the particular machine on which the error was generated. This error typically occurs when integer variables are specified for real or complex arrays or the number of actual parameters is less than the number required by the subroutine being called. If you specify dummy arguments for options you are not using, the dummy arguments must be of the appropriate data type and be properly aligned.

12

FMS$ERR_ARRAY: ARRAY DIMENSION ERROR
FMS checks the dimensions of arrays passed as actual parameters. The error message includes the starting array address, number of dimensions, and the values of each dimension. Refer to the

	individual subroutine descriptions for limitations on array dimensions.
13	**FMS$ERR_CLOSE: ERROR CLOSING FILE** Examine the IOSTAT Parameter and refer to the appropriate operating system manual to determine the cause of failure.
14	**FMS$ERR_DELETE: ERROR DELETING FILE** Examine the IOSTAT Parameter and refer to the appropriate operating system manual for your processor to determine the cause of failure. The ***FMS*** error message lists the file name and the ***FMS*** file number.
15	**FMS$ERR_MEMORY: INSUFFICIENT MAIN MEMORY FOR REQUESTED PROBLEM** ***FMS*** uses the largest contiguous memory region available. This error occurs when the available memory region is insufficient for minimum ***FMS*** requirements. The error message includes the first available memory location, the highest memory location that could be allocated, the length of memory available, and the amount of memory required. If incore files were selected, try setting INCORE=0 (default) to use disk files. If the matrix segment size LENSDF or block size NEQBIO was specified, try letting ***FMS*** determine them.
16	**NOT USED**
17	**FMS$ERR_MVDIF: SPECIFIED MATRIX AND VECTOR FILES ARE INCONSISTENT** The data type or number of equations specified for the matrix and vector files are different. Check the matrix and vector file attribute lists printed by ***FMS*** to determine the inconsistency.
18	**FMS$ERR_MMDIF: INPUT AND OUTPUT MATRIX FILES ARE INCONSISTENT** The record length, number of records, number of equations, diagonal length, matrix data length, data type, symmetry, or file location are different for the input and output matrix during factoring. Check the matrix file attribute lists printed by ***FMS*** to determine the inconsistency. This error is only generated during factoring when the matrix factors are not overlaid on the original matrix. Note that ***FMS*** requires the file location for both matrix files to be the same, incore, or on disk.
19	**FMS$ERR_VVDIF: INPUT AND OUTPUT VECTOR FILES ARE INCONSISTENT** The number of equations, vector length, or data type are different for the two specified vector files. Check the vector file attribute lists printed by ***FMS*** to determine the inconsistency.
20	**FMS$ERR_POSDEF: NEGATIVE PIVOT DURING FACTORING FOR EQUATION number** ***FMS*** has computed a negative pivot during factoring and the ***FMS*** positive definite flag, MPOSDF, was set to halt on negative pivots. This error message includes the equation number, the original matrix diagonal, the inner product sum, and the resulting pivot. This message usually

	indicates an error in the model, which results in bad matrix data being generated.
21	**FMS$ERR_NOWORK: NO WORK SPECIFIED** The actual parameters passed to ***FMS*** indicate that no work was requested. This can result from specifying all zero values for MDATAU, NUMAI, NUMSF and calling the assemble or assemble-factor subroutine. This message also results if you attempt to perform a vector operation and specify zero vectors.
22	**FMS$ERR_UNFAC: ATTEMPT TO SOLVE WITH AN UNFACTORED MATRIX** The matrix file attribute, NEWSEG, is less than or equal to the number of matrix segments, NUMSEG. This condition indicates that the matrix was not completely factored. Review the output from the factoring subroutine to determine if factoring was complete. Check the specified matrix file attribute for possible coding errors.
23	**FMS$ERR_UNRED: ATTEMPT TO SKIP REDUCTION WITH UNREDUCED VECTORS** Subroutine RSDS, RNDS, CHDS, CSDS, or CNDS was called with the formal parameter ISKIP=1. The vector file status on the specified input vector file, IVSTAT, indicates that the vectors have not been reduced. Check to determine if the appropriate value of ISKIP was specified on the solution subroutine and that the appropriate vector files were specified. If forward reduction is to be skipped during solution, check the call to the factoring subroutine to determine if it was performed.
24	**FMS$ERR_SYMDIF: INCONSISTENT SYMMETRY, MATRIX FILE AND SUBROUTINE** The matrix file symmetry, ISTYPE, is inconsistent with the ***FMS*** subroutine called. Check the file attribute list and the subroutine traceback printed by ***FMS***.
25	**FMS$ERR_MDDIF: INCONSISTENT DATA TYPE, MATRIX FILE AND SUBROUTINE** The matrix file data type, IDTYPE, is inconsistent with the ***FMS*** subroutine called. Check the file attribute list and the subroutine traceback printed by ***FMS***.
26	**FMS$ERR_VDDIF: INCONSISTENT DATA TYPE, VECTOR FILE AND SUBROUTINE** The vector file data type, IDTYPE, is inconsistent with the ***FMS*** subroutine called. Check the file attribute list and the subroutine traceback printed by ***FMS***.
27	**FMS$ERR_REDUCED: ATTEMPT TO REDUCE ALREADY REDUCED VECTORS** Subroutine RSDS, RNDS, CHDS, CSDS, or CNDS was called with the formal parameter ISKIP=0. The vector file status on the specified input vector file, IVSTAT, indicates that the vectors have already been reduced. Check the specified vector file and the value of ISKIP to determine the inconsistency.
	FMS$ERR_LUPR: ERROR WRITING TO PRINT OUTPUT FILE LUPR An I/O error occurred while writing to the ***FMS*** output file, LUPR. Check the value of IOSTAT and

28	refer to the appropriate operating system manual for your processor to determine the cause of failure.
29	**FMS$ERR_OUTSIDE: SUBMATRIX EQUATION OUTSIDE MATRIX PROFILE** A submatrix integer file contains an equation number that falls outside the matrix profile. The error message includes the submatrix file number, record number, format type, equation number, and coupling equation number. This error usually indicates an error in computing the matrix profile.
30	**FMS$ERR_SUBTYPE: SUBMATRIX TYPE INAPPROPRIATE FOR PROBLEM** The submatrix type for symmetric problems must be in the range one through five and for nonsymmetric problems one to two. The error message includes the submatrix file number, record number, and format type. This error occurs if symmetric matrix types 3, 4, or 5 are specified for nonsymmetric problems. Submatrix types that are out of range (zero or greater than five) usually indicate that the submatrix integer record was not written properly.
31	**FMS$ERR_ORDER: SUBMATRIX EQUATIONS NOT IN REQUIRED ORDER** Submatrix type 3 was specified but the equations were not in numerical order. The message includes the submatrix file number, record number, format type, equation number, and coupling equation number. Check the submatrix integer file for possible errors. If the equations are in a random order, change the submatrix type to 4.
32	**FMS$ERR_LOWASM: STARTING EQUATION FOR MATRIX ASSEMBLY OUT OF RANGE** The ***FMS*** parameter, LOWASM, is greater than the number of equations. This condition indicates an error in specifying LOWASM or an attempt to reassemble and factor a matrix that has not changed.
33	**FMS$ERR_PIVOT: SMALL OR ZERO PIVOT DURING FACTORING FOR EQUATION number** This condition occurs when a pivot is close to zero, and the ***FMS*** Parameter MZERO=0 (default) is set to halt on zero pivots. The error message includes the equation number, the original matrix diagonal D, the inner product sum S, and the resulting pivot P. This message indicates possible linear dependent equations, due to bad matrix data. Refer to the discussion on pivot precision monitoring for more details.
34	**FMS$ERR_VALUE: ACTUAL PARAMETER EXCEEDS MAXIMUM ALLOWED** The value of an actual parameter exceeded the allowed value. The message includes the specified value and the allowed maximum. This error may occur if you specify more vectors for processing than exist on the input or output files.
	FMS$ERR_GETMEM: ERROR ALLOCATING MEMORY

35	The dimensioned number of memory regions was exceeded. The message includes the current memory region that was assigned and the maximum allowed. This error indicates that an excessive number of incore files are opened concurrently.
36	**FMS$ERR_RELMEM: ERROR RELEASING MEMORY** The requested memory region cannot be returned. The message includes the starting and ending memory addresses, the region number, and the number of regions. This error should not occur. It indicates that the memory pointers were destroyed, probably due to exceeding array bounds.
37	**FMS$ERR_USRSUB: TEST USER SUPPLIED SUBROUTINE CALLED** The ***FMS*** object library contains subroutines RSUBLK, RNUBLK, CHUBLK, CSUBLK, CNUBLK, RNUSLB, CNUSLB, RSUPIV, RNUPIV, which are used to generate test data for the DEMO program. You have directed ***FMS*** to call one of these routines by specifying MDATAU = 1 or 2, or MZERO = 2. You must write your own version of these subroutines. When linking, you must specify the files containing your version, before the ***FMS*** library version.
38	**FMS$ERR_SUBEQN: SUBMATRIX EQUATION EXCEEDS MAXIMUM SPECIFIED.** An equation number found on a submatrix integer record exceeded the maximum number of specified equations MAXEQ on a call to FMSPF or FMSWF.
39	**FMS$ERR_SUBZERO: ZERO SUBMATRIX RECORD LENGTH SPECIFIED.** A submatrix file having fixed-length records (LUS(7)=0) has a zero record length specified (LUS(3), LUS(4) or LUS(10)). This error indicates that the contents of your submatrix file attribute list have been modified since the call to FMSOS.
40	**FMS$ERR_MAXVEC: MORE VECTORS SPECIFIED THAN FILE CONTAINS** The number of vectors specified for processing on the subroutine call exceeds the number of vectors on the vector file(s).
41	**FMS$ERR_FMSRONOTFULLW: ERROR CALLING SUBROUTINE FMSROW OR FMSCOL** This error is caused by one of the following conditions: 1. FMSROW and FMSCOL both called, 2. Attempt to write an equation twice, 3. Too many calls, 4. Not all equations written. You use FMSROW to define the matrix by rows, FMSCOL to define the matrix by columns. You cannot mix calls to FMSROW and FMSCOL for the same matrix. You must define each equation once. FMSROW and FMSCOL expect to be called N+2 times, where N is the number of equations. The extra 2 calls are for setup (IEQN=0) and termination

	(IEQN=N+1).
42	**FMS$ERR_: ATTEMPT TO PERFORM A FULL MATRIX FUNCTION ON PROFILE MATRIX** You have directed ***FMS*** to perform pivoting by specifying a SLAB matrix with MFMAT=3. However, your matrix is not full, as required.
43	**FMS$ERR_FMSINICPUOVER: FMSINI NOT CALLED AS FIRST SUBROUTINE** You must call FMSINI before calling any other ***FMS*** subroutine. In most cases, ***FMS*** has done this for you.
44	**FMS$ERR_: CPU TASK BUFFER OVERFLOW** Too many tasks were dispatched to FMSPAR. A maximum of (MAXCPU-1) children tasks can be started in parallel before FMSYNC must be called.
45	**FMS$ERR_NOASYNC: ATTEMPT TO DO ASYNC I/O TO FILE OPENED FOR SYNC I/O** Asynchronous I/O was attempted on a file opened for synchronous I/O. The IOSYNC Parameter was set for synchronous when the file was opened, but was changed for asynchronous operation after the file was opened.
46	**FMS$ERR_NOSHR: ARGUMENT NOT IN SHARED MEMORY** An argument to the parallel dispatch routine FMSPAR was not in shared memory. You should allocate data needed by subroutines called in parallel from the ***FMS*** memory pool using the memory management system.
47	**FMS$ERR_FILLIMIT: FMS STRIPE FILE LIMIT EXCEEDED** The maximum number of open stripe files per queue is too large. Try closing some files if they are not used.
48	**FMS$ERR_LICVAL: ATTEMPT TO SET FMS PARAMETER BEYOND LICENSE MAXIMUM** An attempt was made to set an ***FMS*** parameter to a value which would exceed the maximum for that parameter as authorized in the ***FMS*** license file.
49	**FMS$ERR_BUFFULL: ALL BUFFERS FULL** The buffer pointers ***FMS*** uses for performing asynchronous I/O have become corrupted. Check the bounds of your arrays.
50	**FMS$ERR_SEM: ERROR PROCESSING SEMAPHORE** An error has occurred acquiring or releasing a semaphore.

51	**FMS$ERR_EXTEND: ERROR EXTENDING FILE** An error has occurred extending a file.
52	**FMS$ERR_DEADLOCK: DEADLOCK - ALL TASKS WAITING ON EVENTS** The parent and children are all waiting on each other. If you have changed the ***FMS*** Parameter IPOLLD, reset it to the default value. If all else fails, try setting MAXCPU to 1.
53	**FMS$ERR_FMSPUT: BAD PARAMETER ON CALL** Subroutine FMSPUT was called with a bad file number in the file attribute list LUFMS, or the data window specified by **A(IROW1:IROW1+NROWS-1,JCOL1:JCOL1+NCOLS-1)** is outside the data being stored on the file.
54	**FMS$ERR_FMSGET: BAD PARAMETER ON CALL** Subroutine FMSGET was called with a bad file number in the Vector File Attribute List, or the data window specified by **A(IROW1:IROW1+NROWS-1,JCOL1:JCOL1+NCOLS-1)** is outside the data being stored on the file.
55	**FMS$ERR_FORMAT: MATRIX FILE DATA IN WRONG FORMAT** Subroutine RSDEX, RNDEX, CHDEX, CSDEX, CNDEX was called but the matrix file is not in standard ***FMS*** format. This can occur if the matrix data was written by FMSROW, FMSCOL or FMSPUT and subroutine RSDAF, RNDAF, CHDAF, CSDAF, CNDAF has not been called to reformat the matrix data.
56	**FMS$ERR_NOCODE: FEATURE NOT IMPLEMENTED** The call requested functionality that is not currently implemented.
57	**FMS$ERR_IODIRECT: ALIGNMENT OR LENGTH ERROR** The ***FMS*** Parameter IOTYPE specified that direct I/O is to be performed but the conditions of the requested transfer do not meet the requirements. This can occur if the transfer length is not an integer multiple of the transfer unit size (usually a page), the memory address does not fall on a required boundary (usually a page) or the disk address does not fall on a required boundary (usually a page). Check the printed output for the transfer to determine which value is in error.
58	**FMS$ERR_FMSSET: BAD PARAMETER ON CALL** Subroutine FMSIST, FMSRST, FMSCST or FMSIGT, FMSRGT, FMSCGT was called with a bad Parameter name or value.
59	**FMS$ERR_PARENT: NOT CALLED FROM PARENT** A subroutine which must be called by the parent was called by a child thread. This usually occurs with calls to subroutines FMSPAR, FMSILG, FMSRLG, FMSCLG or FMSILR, FMSRLR, FMSCLR.

60	**FMS$ERR_NOTHREAD: THREAD CANNOT BE FOUND** A call to FMSIGT(' MYNODE', MYNODE) was made to obtain the thread number MYNODE of the calling thread but the thread could not be found.
61	**FMS$ERR_RUNSYNC: INCONSISTENT CALLS TO FMSRUN AND FMSYNC** This error occurs when the calls to FMSPAR, FMSRUN and FMSYNC are inconsistent. The order of the calls must be as follows: 1. FMSPAR is called to queue work for each child thread, 2. FMSRUN is called once to start the children running, 3. The parent performs some work (usually similar to the children tasks), 4. FMSYNC is called once to wait for all children to complete.
62	**FMS$ERR_GPU: ERROR IN GPU PROCESSING** This message occurs if a call to a CUDA RUNTIME LIbRARY or CUBLAS routine returns an error code. See printout for the name of the routine and additional information.
63	**FMS$ERR_BLAS: ERROR ON BLAS PARAMETER** This error occurs if one of the arguments to a BLAS routine is in error.

Copyright © Multipath Corporation

F. FMS PROGRAM DEMO

FMS includes program DEMO, which may be used for the following functions:

- Installation Verification

  After installing ***FMS*** you can run DEMO to verify that all software has been properly loaded.

- Performance Studies.

  Program DEMO can be used to construct test problems of any size and study their performance on various hardware configurations.

- Training.

  Program DEMO is an excellent tool to learn about ***FMS***. Designed to be run interactively from a terminal, DEMO can easily modify matrix and hardware parameters to determine their effect on output and performance.

Building Program DEMO

The distribution of ***FMS*** includes both FORTRAN and C versions of program DEMO, which are listed below:

```
        PROGRAM DEMO
        CALL FMSSUB
        STOP
        END
```

```
#include <stdio.h>
#include <stdlib.h>
extern void fmssub ();
main ()
{
   fmssub ();
}
```

This program is compiled and linked to the ***FMS*** library, which includes FMSSUB. For details on compiler options and linking procedures, consult the WEB page for downloading ***FMS*** at fmslib.com.

Running the Test Program

Program DEMO is designed to run interactively from a terminal. When you start DEMO, you will see a prompt soliciting input in the form

FMS>

The input you enter is in the form

FMS>`name=value`

where

`name`=the name of a test parameter,

`value`=the value assigned to the parameter.

The equal sign functions as a delimiter, separating the name parameter from the value parameter. The name parameter starts at the beginning of the input data line after the FMS> prompt.

You may set up loops to run several tests together by specifying up to five values as follows:

FMS>`name=value1,value2,value3,value4,value5`

For example, to test all modules you would specify

FMS>`MODULE=RS,RN,CH,CS,CN`

DEMO will automatically create nested loops when you specify multiple values for more than one parameter. You may also specify a random number for a test parameter as follows:

FMS>`name=low:high`

where `low` and `high` specify the range of random numbers to be selected.

You may examine and set any ***FMS*** Parameter using FMSSET. To call FMSSET, enter FMSSET on the command line for name:

FMS>`FMSSET`

To return to DEMO input, type the characters RETURN:

FMSSET>`RETURN`

When you no longer want to run demo, enter

FMS>`QUIT`

The following table contains a list of the test parameter names, their data types, and a description of the function performed. Only the first four characters of the name parameter must be specified.

DEMO Test Parameters

Name	Data Type	Description
STATUS	N/A	No data required. Displays a list of the test parameter names and the current values assigned. This is a useful command to list all these options.
MODIFY	N/A	Display test parameters which have been modified from default values.

FMSSET	N/A	Call subroutine FMSSET to examine or modify FMS parameters. While in FMSSET, you can list all the FMS Parameters with the `FMSSET>SHOW=PARAMETERS` command. When you want to return to the DEMO input prompt, enter the characters `RETURN` `FMSSET>RETURN`
MODULE	CHARACTER	FMS module =RS, Real symmetric =RN, Real nonsymmetric =CH, Complex Hermitian =CS, Complex symmetric =CN, Complex nonsymmetric. Default value = RS
EQUATIONS	INTEGER	Number of equations. Default value = 1000.
VECTORS	INTEGER	Number of vectors. Default value = 1.
SPARSITY	CHARACTER	Type of matrix sparsity. =FULL, Full matrix. =BANDED, Banded matrix. =RANDOM, Random profile. =PROFILE, Profile vector read from unit 2. =DIAGONAL, Random profile with ten percent diagonal-only equations. Default value = BANDED
BANDWIDTH	INTEGER	Bandwidth to be used with SPARSITY=BANDED or RANDOM. For banded matrices, this is the constant half bandwidth. For random profile matrices, the bandwidth ranges from 1 to 2*BAND. Default value = 100.
SUBMATRIX	INTEGER	Submatrix test code obtained by adding together the options listed below. Each submatrix option generates 2x2 submatrices that, when added together, produce the same global matrix as MTEST=1. =0, None. =1, Type 1 (Full stored by columns). =2, Type 2 (Full sorted by rows). =4, Type 3 (Lower triangle, numerical order, symmetric).

		=8, Type 4 (Lower triangle, random order, symmetric). =16, Type 5 (Row matrix, symmetric). =64, Use variable-length record format. Several submatrices may be tested together. For example, SUBM=3 builds the global matrix using submatrices of types 1 and 2. Default value = 0.
MTEST	INTEGER	Test data generation option. =0, No test data generated. =1, Integer test data generated. =2, Random number test data generated. Default value = 1.
ISEED	INTEGER	Seed used to generate random numbers. Default value = 0.
NUMAI	INTEGER	Number of initialization matrices. Default value = 0.
REDUCE	CHARACTER	Reduce during factoring option. =NO, Reduce during vector solution. =YES, Reduce during factoring. Default value = NO.
IOA	INTEGER	Location of matrices =0, Use INCORE to determine location. =1, Memory. =2, Memory if it fits, disk otherwise. Default value = 0
IOX	INTEGER	Location of solution and {X} vectors. =0, Use INCORE to determine location. =1, Memory. =2, Memory if it fits, disk otherwise. Default value = 0.
IOB	INTEGER	Location of right-hand side and {Y} vectors. =0, Use INCORE to determine location. =1, Memory. =2, Memory if it fits, disk otherwise. Default value = 0.
		Use Profile or Wavefront numbering

IPROF	INTEGER	=0, use existing profile =1, compute profile using subroutine FMSWF =2, compute profile using subroutine FMSPF without renumbering =3, compute profile using subroutine FMSPF with renumbering Default value = 0.
FMSROW	INTEGER	Populate matrix using FMSROW, FMSCOL or FMSPUT. = -2, use FMSCOL called with decreasing column numbers. = -1, use FMSROW called with decreasing row numbers. = 1, use FMSROW called with increasing row numbers. = 2, use FMSCOL called with increasing column numbers. = 3, use FMSPUT called with rows of data. CAUTION: This can result in writing 1 word at a time to the disk. = 4, use FMSPUT called with columns of data. Default value = 0.
FMSIST	INTEGER	Call subroutine FMSIST. The forms of this command are `FMSIST/name=value` `FMSIST/name=value1:value2` `FMSIST/name=value1,value2,value3...` This is a alternative to entering `FMSIST` `name=value` `RETURN` It allows loops to be created over ***FMS*** Parameters by specifying a range to randomly generate values (value1:value2) or a list of values (value1,value2,...). Default value = FMSIST not called.
PRINT	INTEGER	Print code obtained by adding together the options listed below: =0, Nothing. =1, Maximum error. =2, Error norms. =4, Right-hand side test vectors {B}. =16, Files. =32, Solution vectors {X}. =256, Test matrix [AL], [D], [AU]. =512, Matrix factors [L], [I/D], [U]. In addition, the test program prints the output requested by the FMS parameters; IPRI, IPRA, IPRF, IPRS, IPRMV, IPRVV.

		Default value = 1
FILE	CHARACTER	A one to 32-character name that specifies the input file for reading commands. When FMS encounters this command, input is switched from the terminal to the specified file. When FMS encounters an end-of-file, input is switched back to the terminal. This command is useful for running standard test programs. If the print parameter, PRINT, described above is specified greater than 1, data read from the file is echoed at the terminal.
IERTOL	INTEGER	Tolerance on solution error to print an error message. Value is expressed as 10^{value}. Default value = -8
LOOP	INTEGER	Number of times to repeat test. =0, infinite looping. Default = 1.
RANDOM	INTEGER	Number of times to repeat random test. When a range of test or ***FMS*** Parameters are specified as value1:value2, random values within this range are generated. This Parameter determines the number of times the tests are repeated, with new values being generated for each test. This feature is used to generate random test data for testing ***FMS*** prior to release. Default = 0
RESET	N/A	Reset test parameters to default values.
WRITE	N/A	Causes FMS to write a file that contains all the current settings of any modified DEMO or FMS Parameter. This file can later be input with the FILE= command to repeat the test. FMS displays the name of the file it created.
COMMENT	N/A	Causes the test program to ignore the material that follows. Can be used to document test data files.
GO	N/A	Start the test program.
GOSTAT	N/A	Display modified test and FMS parameters and start the test program.
QUIT	N/A	End the test program.

Copyright © Multipath Corporation

G. FMS ENVIRONMENT VARIABLES

FMS uses environment variables for customizing the software to your environment.

Setting Environment Variables

Setting environment variables depends on your operating system. In the discussion below, we will set the environment variable **`envar`** to **`value`**.

Linux

```
export envar=value (ksh)
or
setenv envar value (csh)
```

Windows

1. Open the control panel (right-click on the Windows icon in the lower left corner)
2. Select "System and Security"
3. Select "System".
4. Select "Advanced system settings".
5. In the window that pops up, select "Environment Variables"
6. In the window that pops up, select "New..." under "System variables".
7. In the "Variable name" box, enter the name of the environment variable, **`envar`**
8. In the "Variable value" box, enter the value of the variable, **`value`**
9. Click OK
10. Close the other two windows by clicking "OK".

Environment Variables used by ***FMS***

FMS Environment Variables

Variable	Description
FMSDISK	Directory where ***FMS*** stores data on disk. If file striping is not used, all the data is stored in this directory. If file striping is used, "small" files are stored in this directory and large files are striped and stored in the stripe directories. Default value: None. Must be specified if disk storage is used.

FMSDISKijk	Directories where striped files are stored. The Environment Variable FMSDISKijk specifies the directory for queue i, stripe j and volume k. Default value: None. Must be specified when File Striping is used.
FMSPAGE	Directory where the page file will be created. When the ***FMS*** Parameter NOPAGE=0, a private page file is created in this directory. Default value: None. Must be specified when a private page file is used.
FMSLIC	Directory containing the license file. ***FMS*** uses the following license files: • fmslic.txt, for production applications; • mwlic.txt, for **MatrixWarrior**; • exlic.txt, for programs DEMO and the EXAMPLE programs. This specified the directory containing one (or more) of these files. Default value: The current working directory.
FMSMAN	Directory where the ***FMS*** manual is located. If you have installed the ***FMS*** manual on your local system, this should be set to the directory containing file FMS_Manual.html. Default value: Location of the ***FMS*** manual on the fmslib.com website.
FMSHTML	Directory where **Dashboard** writes the performance reports. When the ***FMS*** Parameter IWATCH is set, ***FMS*** generates a performance report in the FMSHTML directory. You may want to place this report on a cross-mounted disk, on a WEB server or other host with greater access than the machine currently used for running ***FMS***. Default value: The current working directory.
BROWSER	WEB Browser to display **Dashboard** reports. You may select which WEB Browser **Dashboard** uses to display the performance reports. The browser must be available on your system. Default value(Linux): "firefox" Default value(windows): The default value selected in your preferences.
FMSNVML	Directory where the shared object file libnvidia-ml.so (Linux) or nvml.dll (Windows) is located. This directory is usually where the program nvidia-smi is located. If specified, this directory is searched first. If the file is not found, the normal search paths including the current working directory are scanned. If the file is not found in any specified path, GPUs will not be used.
	Linux Only

FMSNUMA	Directory where the shared object library libNuma.so.1 is located. If specified, this directory is searched first. If the file is not found, the normal search paths including the current working directory are scanned. If the file is not found in any specified path, NUMA technology will not be used.
FMSCUBLAS	Windows Only Directory where the shared object library cublas64_xx.dll is located, where xx is the release number of the software. If specified, this directory is searched first. If the file is not found, the normal search paths including the current working directory are scanned. If the file is not found in any specified path, GPUs will not be used. This file is distributed with the library.
FMSGPUEND	Action to take if an error occurs initializing the GPUs. 0=continue without GPUs (default) 1=Exit. ***FMS*** is designed to run with and without GPUs. This GPU plug-and-play feature allows a single version of an application using ***FMS*** to run on all systems. At startup ***FMS*** searches for GPU hardware and software. If all requirements are met, GPUs are used. However if one of the complnents is missing, the application continues without using GPUs. If you know you have GPUs and want to exit if they will not be used, setting this environment variable will cause an exit. ***FMS*** prints detailed messages when GPU hardware and software components are missing providing the information required to correct the problem.
FMSTIMEOUT	Maximum time in wall seconds for job to run. 0=no time limit (default) When set, ***FMS*** starts a clock during initialization. If the specified time is exceeded, execution stops. This feature can be used to check the initialization phase before executing a longer run.

Copyright © Multipath Corporation

H. LIST OF FIGURES

Flowcharts

Flowchart Without FMS

Flowchart With FMS

Flowchart - Programming

Assembly Flowchart

Factor Flowchart

Assembly and Factor Flowchart

Subspace Iteration

Files

PROFILE Matrix Storage

BLOCK Matrix Format

SLAB Matrix Storage

FMS File Striping System

Submatrix Format 1

Submatrix Format 2

Submatrix Formats 3 and 4

Submatrix Format 5

FMS Parameters

NEQPR1 & NEQPR2

IJSTEP=1

IJSTEP=NEQBLK

IJSTEP=NEQBIO

LOWASM

LUOK

Defining a BLOCK of Data

Defining a SLAB of Data

Miscellaneous

MULTIPATH

Components of Computing

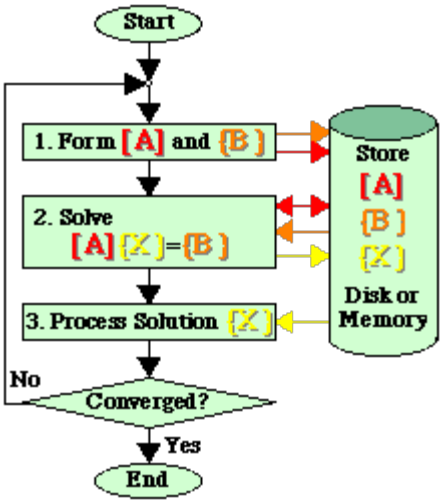
Data Alignment

CN Storage Example

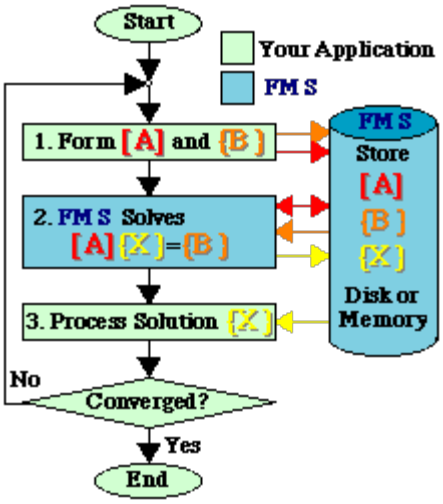
CN Time Example

Copyright © Multipath Corporation

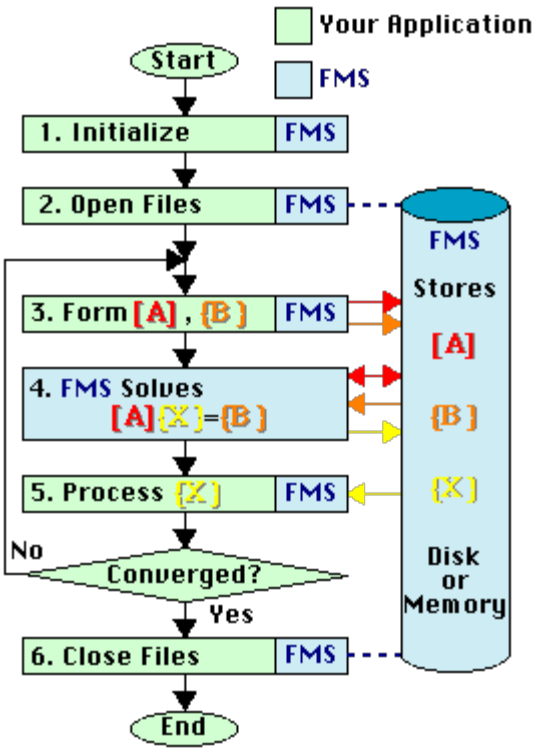
flow_nofms.gif %d×%d pixels



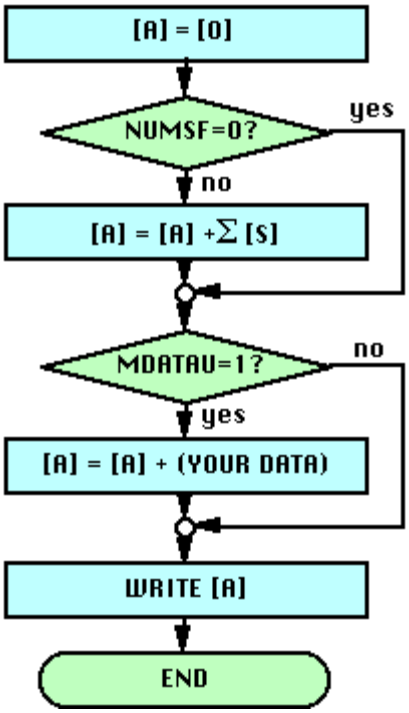
flow_fms.gif %d×%d pixels



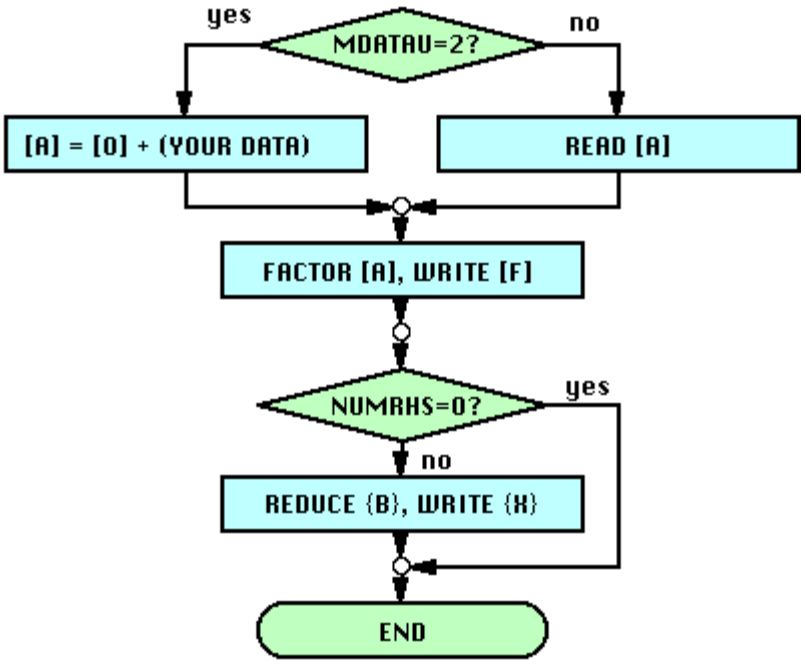
flow_6step.gif %d×%d pixels



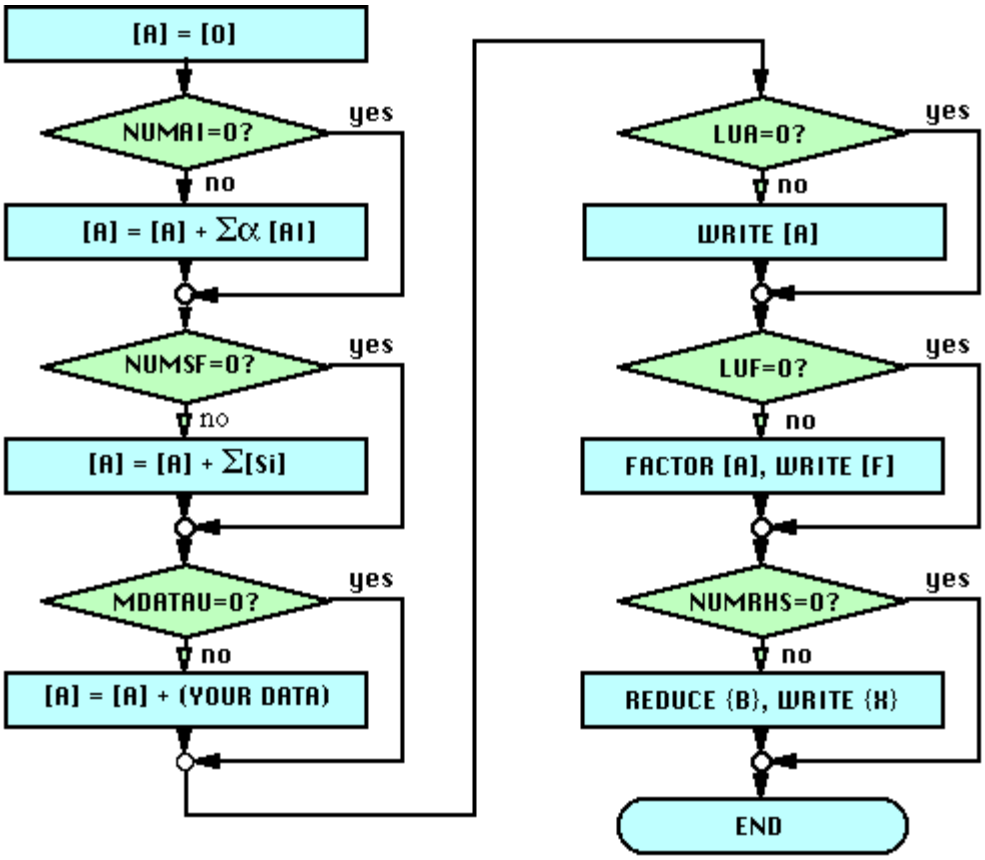
xxda.gif %d×%d pixels



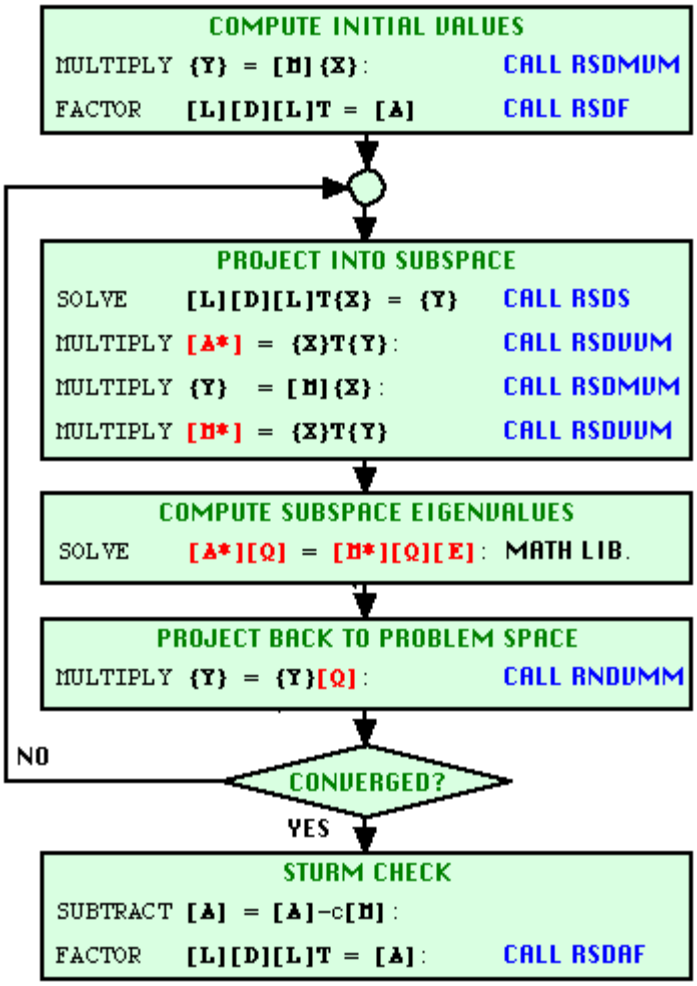
xxdf.gif %d×%d pixels



xxdaf.gif %d×%d pixels



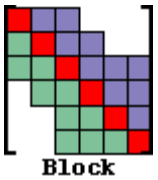
subspace.gif %d×%d pixels



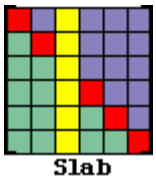
profile.gif %d×%d pixels



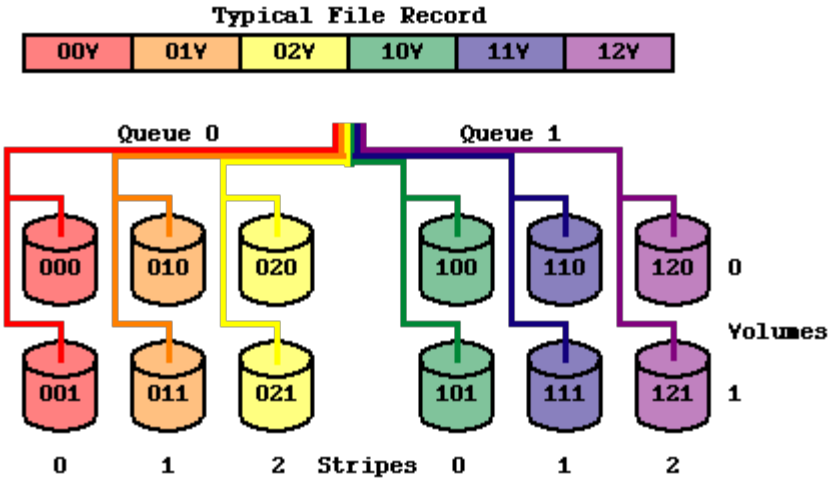
block.gif %d×%d pixels



slab.gif %d×%d pixels



stripe.gif %d×%d pixels



subfmt1.gif %d×%d pixels



subfmt2.gif %d×%d pixels



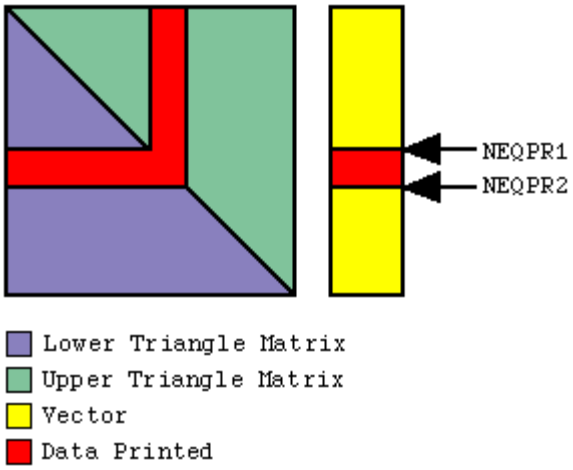
subfmt34.gif %d×%d pixels



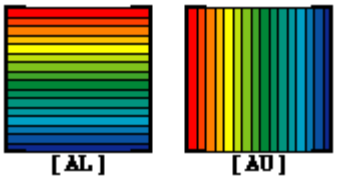
subfmt5.gif %d×%d pixels



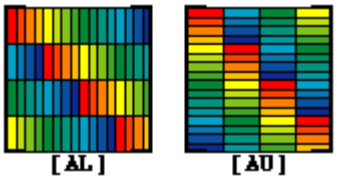
neqpr.gif %d×%d pixels



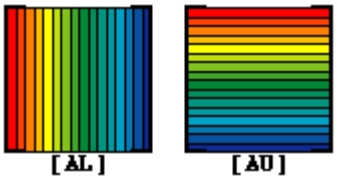
ijstep1.gif %d×%d pixels



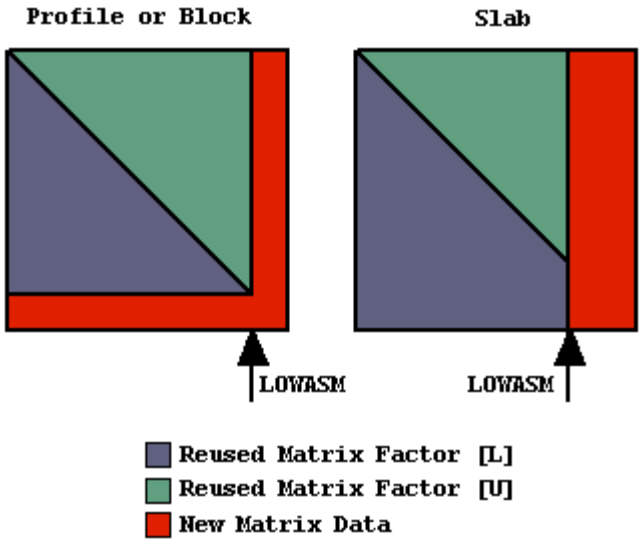
ijstep2.gif %d×%d pixels



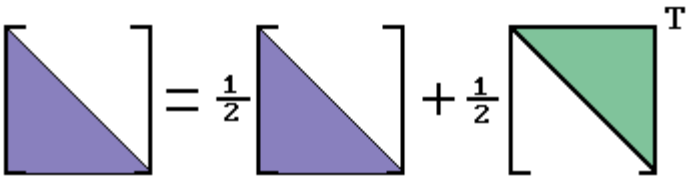
ijstep3.gif %d×%d pixels



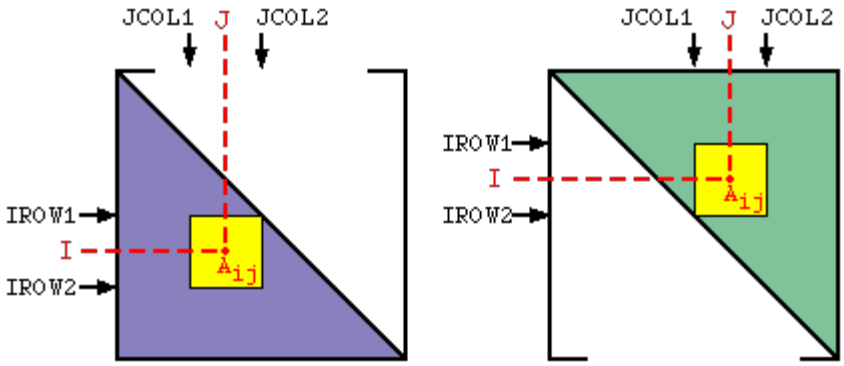
lowasm.gif %d×%d pixels



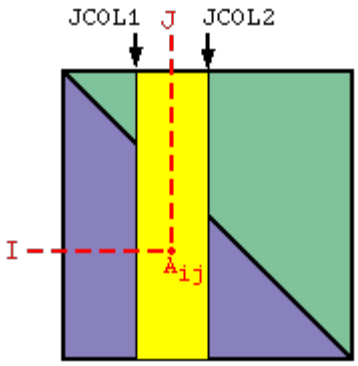
luok.gif %d×%d pixels



xxublk.gif %d×%d pixels



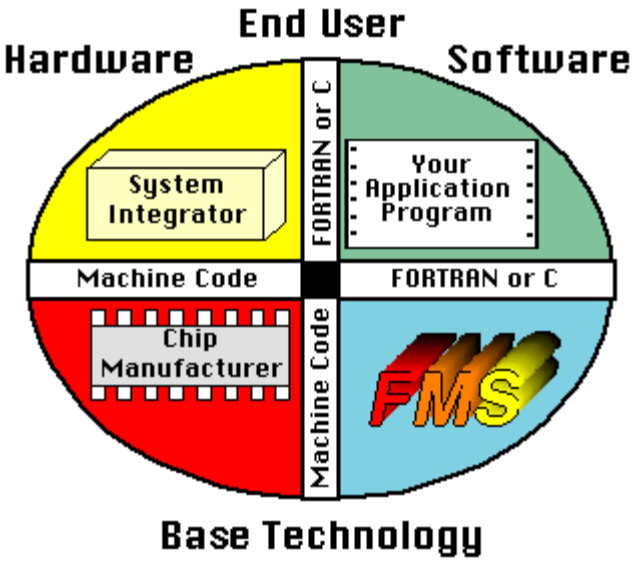
xxuslb.gif %d×%d pixels



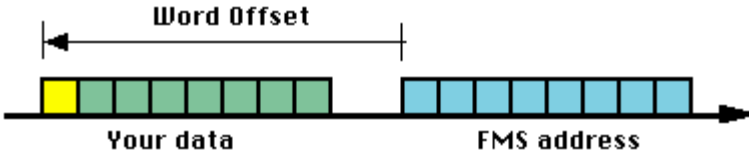
multipath.gif %d×%d pixels



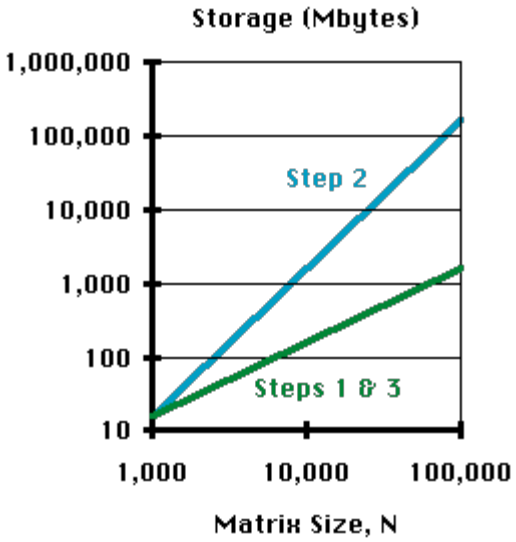
components.gif %d×%d pixels



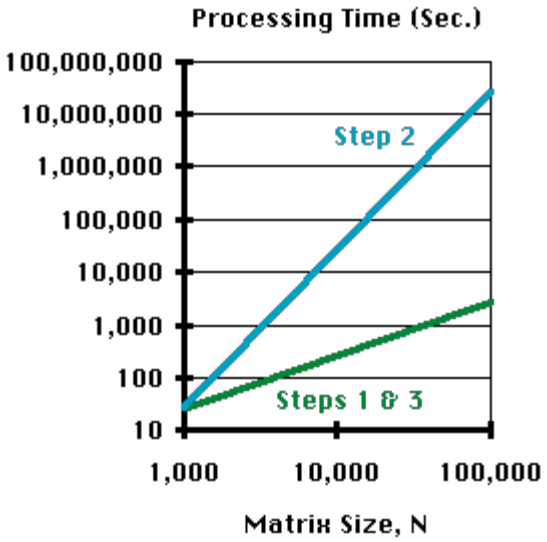
align.gif %d×%d pixels

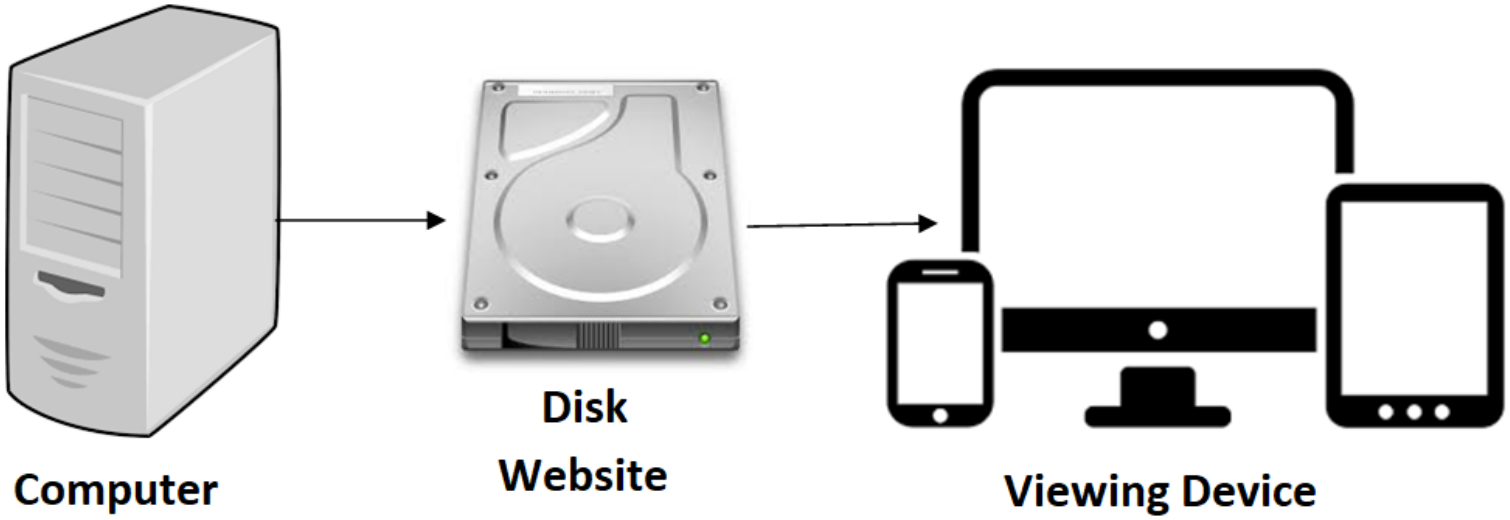


cn_storage.gif %d×%d pixels



cn_time.gif %d×%d pixels




Computer
Disk
Website
Viewing Device

Go Back

COPYRIGHT © 2025 MULTIPATH CORPORATION

Copyright

The contents of Multipath's Fast Matrix Solver (***FMS***) Manual are protected by the copyright laws of the United States. No portion may be reproduced in any form, or by any means, without the prior written consent of Multipath Corporation.
To obtain reproduction consent, email

sales@fmslib.com

Disclaimer

The information in this manual is provided "as is" with all faults and without warranty of any kind, expressed or implied, including those of merchantability and fitness for a particular purpose, or arising from a course of dealing, usage or trade practice. Multipath makes no warranties or representations regarding the accuracy or completeness of the information.
In no event shall Multipath be liable for any incidental or consequential damages, lost profits, or lost data, or any indirect damages even if Multipath has been informed of the possibility thereof.

Trademark

FMS is a registered trademark of Multipath Corporation.

Copyright © Multipath Corporation

MatrixWarrior Powered by FMSlib Copyright © Multipath Corporation

Fixed Properties for GPUs 0 to 3

GPU Number	RTL	0	1	2	3
	NVML	0	1	2	3
Product Name		Tesla V100-SXM2-16GB	Tesla V100-SXM2-16GB	Tesla V100-SXM2-16GB	Tesla V100-SXM2-16GB
Product Brand		Tesla	Tesla	Tesla	Tesla
Capability(major,minor)		(7.0)	(7.0)	(7.0)	(7.0)
Global Memory (bytes)		16936861696	16936861696	16936861696	16936861696
Multiprocessors		80	80	80	80
Cores per Multiprocessor		128	128	128	128
64-bit flops/SM/Clock		64	64	64	64
Warp size		32	32	32	32
Clock Rate (GHz)		1.53	1.53	1.53	1.53
CPUs Used		1 - 5	6 - 10	11 - 15	16 - 20
Accounting Buffer Size		1920	1920	1920	1920
Serial Number		0323617048032	0323317055385	0323417078460	0323417077918
Minor Number		/dev/nvidia0	/dev/nvidia1	/dev/nvidia2	/dev/nvidia3
VBIOS Version		88.00.18.00.01	88.00.18.00.01	88.00.18.00.01	88.00.18.00.01
Multi-GPU Board		No	No	No	No
Board Id		0x600	0x700	0xA00	0xB00
Board Part Number		692-2G503-0202-000	692-2G503-0202-000	692-2G503-0202-000	692-2G503-0202-000
Inforom Version	Image	G503.0202.00.01	G503.0202.00.01	G503.0202.00.01	G503.0202.00.01
	OEM	1.1	1.1	1.1	1.1
	ECC	5.0	5.0	5.0	5.0
	Power	N/A	N/A	N/A	N/A
Virtualization Mode		0	0	0	0
	Bus	0x06	0x07	0x0A	0x0B

PCI	Device	0x00	0x00	0x00	0x00
	Domain	0x00000000	0x00000000	0x00000000	0x00000000
	Device Id	0x1DB110DE	0x1DB110DE	0x1DB110DE	0x1DB110DE
	Bus Id	0000:06:00.0	0000:07:00.0	0000:0A:00.0	0000:0B:00.0
	Sub System Id	0x00000000	0x00000000	0x00000000	0x00000000
	Max Link Gen.	3	3	3	3
	Cur. Link Gen.	3	3	3	3
	Max Link Width	16x	16x	16x	16x
	Cur. Link Width	16x	16x	16x	16x
Temperature Threshold	Shutdown	90	90	90	90
	Slowdown	87	87	87	87
	Memory	85	85	85	85
	Gpu	83	83	83	83
Enforced Power Limit		300 Watts	300 Watts	300 Watts	300 Watts
Power Constraints	Min.	100 Watts	100 Watts	100 Watts	100 Watts
	Max.	300 Watts	300 Watts	300 Watts	300 Watts
Default App. Clocki (MHz)	Graphics	1312	1312	1312	1312
	Memory	877	877	877	877
Max. Clocks (MHz)	Graphics	1530	1530	1530	1530
	SM	1530	1530	1530	1530
	Memory	877	877	877	877
	Video	1372	1372	1372	1372
Supported Clock	Idle	Active	Active	Active	Active
	App. Setting	Active	Active	Active	Active
	Sw Power Cap	Active	Active	Active	Active
	Hw Slowdown	Active	Active	Active	Active

Throttle Reasons	**Hw Thermal**	Active	Active	Active	Active
	Hw Power	Active	Active	Active	Active
	Sync Boost	Active	Active	Active	Active
	Sw Thermal	Active	Active	Active	Active

Fixed Properties for GPUs 4 to 7

GPU Number	**RTL**	**4**	**5**	**6**	**7**
	NVML	**4**	**5**	**6**	**7**
Product Name		Tesla V100-SXM2-16GB	Tesla V100-SXM2-16GB	Tesla V100-SXM2-16GB	Tesla V100-SXM2-16GB
Product Brand		Tesla	Tesla	Tesla	Tesla
Capability(major,minor)		(7.0)	(7.0)	(7.0)	(7.0)
Global Memory (bytes)		16936861696	16936861696	16936861696	16936861696
Multiprocessors		80	80	80	80
Cores per Multiprocessor		128	128	128	128
64-bit flops/SM/Clock		64	64	64	64
Warp size		32	32	32	32
Clock Rate (GHz)		1.53	1.53	1.53	1.53
CPUs Used		21 - 25	26 - 30	31 - 35	36 - 40
Accounting Buffer Size		1920	1920	1920	1920
Serial Number		0323617048084	0323317055906	0323317055559	0323617048001
Minor Number		/dev/nvidia4	/dev/nvidia5	/dev/nvidia6	/dev/nvidia7
VBIOS Version		88.00.18.00.01	88.00.18.00.01	88.00.18.00.01	88.00.18.00.01
Multi-GPU Board		No	No	No	No
Board Id		0x8500	0x8600	0x8900	0x8A00
Board Part Number		692-2G503-0202-000	692-2G503-0202-000	692-2G503-0202-000	692-2G503-0202-000
Inforom Version	**Image**	G503.0202.00.01	G503.0202.00.01	G503.0202.00.01	G503.0202.00.01
	OEM	1.1	1.1	1.1	1.1
	ECC	5.0	5.0	5.0	5.0

	Power	N/A	N/A	N/A	N/A
Virtualization Mode		0	0	0	0
PCI	**Bus**	0x85	0x86	0x89	0x8A
	Device	0x00	0x00	0x00	0x00
	Domain	0x00000000	0x00000000	0x00000000	0x00000000
	Device Id	0x1DB110DE	0x1DB110DE	0x1DB110DE	0x1DB110DE
	Bus Id	0000:85:00.0	0000:86:00.0	0000:89:00.0	0000:8A:00.0
	Sub System Id	0x00000000	0x00000000	0x00000000	0x00000000
	Max Link Gen.	3	3	3	3
	Cur. Link Gen.	3	3	3	3
	Max Link Width	16x	16x	16x	16x
	Cur. Link Width	16x	16x	16x	16x
Temperature Threshold	**Shutdown**	90	90	90	90
	Slowdown	87	87	87	87
	Memory	85	85	85	85
	Gpu	83	83	83	83
Enforced Power Limit		300 Watts	300 Watts	300 Watts	300 Watts
Power Constraints	**Min.**	100 Watts	100 Watts	100 Watts	100 Watts
	Max.	300 Watts	300 Watts	300 Watts	300 Watts
Default App. Clocki (MHz)	**Graphics**	1312	1312	1312	1312
	Memory	877	877	877	877
Max. Clocks (MHz)	**Graphics**	1530	1530	1530	1530
	SM	1530	1530	1530	1530
	Memory	877	877	877	877
	Video	1372	1372	1372	1372
	Idle	Active	Active	Active	Active
	App. Setting	Active	Active	Active	Active
	Sw Power				

Supported Clock Throttle Reasons	**Cap**	Active	Active	Active	Active
	Hw Slowdown	Active	Active	Active	Active
	Hw Thermal	Active	Active	Active	Active
	Hw Power	Active	Active	Active	Active
	Sync Boost	Active	Active	Active	Active
	Sw Thermal	Active	Active	Active	Active

MatrixWarrior Powered by FMSlib Copyright © Multipath Corporation

Changeable Properties for GPUs 0 to 3

GPU Number	RTL	0	1	2	3
	NVML	0	1	2	3
Compute Mode		Default	Default	Default	Default
ECC Mode	Current	Enabled	Enabled	Enabled	Enabled
	Pending	Enabled	Enabled	Enabled	Enabled
Power Management Limit		300 Watts	300 Watts	300 Watts	300 Watts
Applications Clocks (MHz)	Graphics	1312	1312	1312	1312
	Memory	877	877	877	877
API Set Clock Restrictions	Application	any	any	any	any
	Auto Boosted	any	any	any	any

Changeable Properties for GPUs 4 to 7

GPU Number	RTL	4	5	6	7
	NVML	4	5	6	7
Compute Mode		Default	Default	Default	Default
ECC Mode	Current	Enabled	Enabled	Enabled	Enabled
	Pending	Enabled	Enabled	Enabled	Enabled
Power Management Limit		300 Watts	300 Watts	300 Watts	300 Watts
Applications Clocks (MHz)	Graphics	1312	1312	1312	1312
	Memory	877	877	877	877
API Set Clock Restrictions	Application	any	any	any	any
	Auto Boosted	any	any	any	any

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

Dynamic Properties for GPUs 0 to 3

GPU Number	**RTL**	**0**	**1**	**2**	**3**
	NVML	**0**	**1**	**2**	**3**
GPU Temperature		33 C	34 C	34 C	32 C
Utilization	**Encoder**	0 %	0 %	0 %	0 %
	Decoder	0 %	0 %	0 %	0 %
Clocks Current/Max (MHz)	**Graphics**	1312/1530	1312/1530	1312/1530	1312/1530
	SM	1312/1530	1312/1530	1312/1530	1312/1530
	Memory	877/877	877/877	877/877	877/877
Clocks(MHz) Current;Target/ Default;Max	**Graphics**	1312;1312	1312;1312	1312;1312	1312;1312
		1312;1530	1312;1530	1312;1530	1312;1530
	SM	1312;1312	1312;1312	1312;1312	1312;1312
		1312;1530	1312;1530	1312;1530	1312;1530
	Memory	877;877	877;877	877;877	877;877
		877;N/A	877;N/A	877;N/A	877;N/A
	Video	1177;N/A	1177;N/A	1185;N/A	1192;N/A
		N/A;N/A	N/A;N/A	N/A;N/A	N/A;N/A
Clocks Throttle Reasons		None	None	None	None
Performance State		P0	P0	P0	P0
Power State		P0	P0	P0	P0
Power Usage (Watts)		57	55	56	54
Power Usage (Joules)		14197193	13400142	13663643	13333534
Pcie Throughput	**RX (bytes)**	2000	0	0	0
	TX (bytes)	1000	22000	0	0
Pcie Replays		0	0	0	0
ECC Errors Volatile Corrected;	**Device**	0;0;0;0;	0;0;0;0;	0;0;0;0;	0;0;0;0;
	Register	0;0;0;0;	0;0;0;0;	0;0;0;0;	0;0;0;0;
	L1 Cache	0;0;0;0;	0;0;1;0;	0;0;0;0;	0;0;0;0;
	L2 Cache	0;0;0;0;	0;0;0;0;	0;0;0;0;	0;0;0;0;

Uncorrected; Aggregate Corrected; Uncorrected;	Texture	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;
	Texture Shr.	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;
	CBU	NA;0;NA;0;	NA;0;NA;0;	NA;0;NA;0;	NA;0;NA;0;
	Total	0;0;0;0	0;0;1;0	0;0;0;0	0;0;0;0

Dynamic Properties for GPUs 4 to 7

GPU Number	RTL	4	5	6	7
	NVML	4	5	6	7
GPU Temperature		31 C	33 C	35 C	31 C
Utilization	Encoder	0 %	0 %	0 %	0 %
	Decoder	0 %	0 %	0 %	0 %
Clocks Current/Max (MHz)	Graphics	1312/1530	1312/1530	1312/1530	1312/1530
	SM	1312/1530	1312/1530	1312/1530	1312/1530
	Memory	877/877	877/877	877/877	877/877
Clocks(MHz) Current;Target/ Default;Max	Graphics	1312;1312	1312;1312	1312;1312	1312;1312
		1312;1530	1312;1530	1312;1530	1312;1530
	SM	1312;1312	1312;1312	1312;1312	1312;1312
		1312;1530	1312;1530	1312;1530	1312;1530
	Memory	877;877	877;877	877;877	877;877
		877;N/A	877;N/A	877;N/A	877;N/A
	Video	1177;N/A	1177;N/A	1177;N/A	1185;N/A
		N/A;N/A	N/A;N/A	N/A;N/A	N/A;N/A
Clocks Throttle Reasons		None	None	None	None
Performance State		P0	P0	P0	P0
Power State		P0	P0	P0	P0
Power Usage (Watts)		55	56	56	54
Power Usage (Joules)		13535965	13621064	13637899	13250902
Pcie Throughput	RX (bytes)	0	0	0	0
	TX (bytes)	0	0	0	0
Pcie Replays		0	0	0	0
ECC Errors Volatile	Device	0;0;0;0;	0;0;0;0;	0;0;0;0;	0;0;0;0;
	Register	0;0;0;0;	0;0;0;0;	0;0;0;0;	0;0;0;0;
	L1 Cache	0;0;0;0;	0;0;0;0;	0;0;0;0;	0;0;0;0;

Corrected; Uncorrected; Aggregate Corrected; Uncorrected;	L2 Cache	0;0;0;0;	0;0;0;0;	0;0;0;0;	0;0;0;0;
	Texture	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;
	Texture Shr.	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;	NA;NA;NA;NA;
	CBU	NA;0;NA;0;	NA;0;NA;0;	NA;0;NA;0;	NA;0;NA;0;
	Total	0;0;0;0	0;0;0;0	0;0;0;0	0;0;0;0

MatrixWarrior Powered by FMSlib Copyright © Multipath Corporation

Run Time Library Properties for GPUs 0 to 3

RTL GPU Number		0	1	2	3
Product Name		Tesla V100-PCIE-16GB	Tesla V100-PCIE-16GB	Tesla V100-PCIE-16GB	Tesla V100-PCIE-16GB
Capability(major,minor)		(7.0)	(7.0)	(7.0)	(7.0)
Global Memory (bytes)		16936861696	16936861696	16936861696	16936861696
Multiprocessors		80	80	80	80
Cores per Multiprocessor		128	128	128	128
64-bit flops/SM/Clock		64	64	64	64
Warp size		32	32	32	32
Clock Rate (GHz)	SM	1.38	1.38	1.38	1.38
	Memory	0.88	0.88	0.88	0.88
CPUs Used		All	All	All	All
Compute Mode		Default	Default	Default	Default
TCC Driver		no	no	no	no
Memory Bus Width		4096	4096	4096	4096
Multi-GPU Board		no	no	no	no
PCI	Device ID	0x00	0x00	0x00	0x00
	Domain	0x00000000	0x00000000	0x00000000	0x00000000
	Bus Id	0x3B	0x5E	0x86	0xAF
Multi-GPU Board		Yes	Yes	Yes	Yes

MatrixWarrior Powered by ***FMS***lib Copyright © Multipath Corporation

A movie showing the history of the Performance page is available online at Multipath's WEB site. Click HERE for viewing.