GPU Acceleration of <u>BIG</u> Matrix Algebra

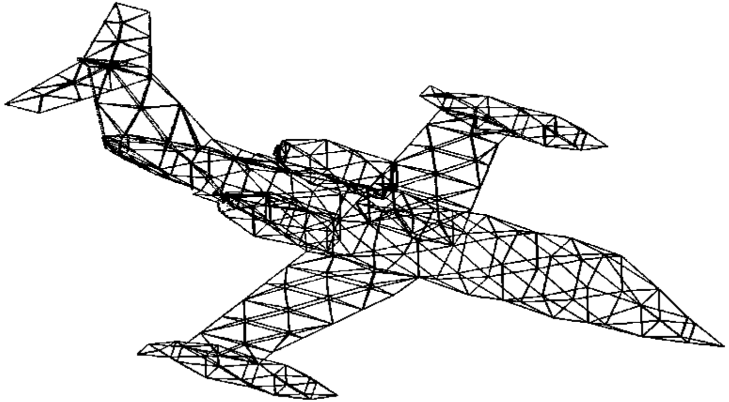
HP-CAST

Dr. Ronald Young
November 10, 2012

Why do you want to solve a BIG problem?

Scientific community
- Fluid Mechanics
- Structural Analysis
- Heat Transfer
- Electromagnetics
- Diffusion (reservoir simulation)
- Acoustics
- Circuit design
- Economic modeling

NVIDIA Tesla users



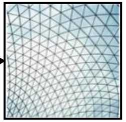
Discretized mesh model

A(N,N) is a model of the plane

$$[A]\{X\} = \{B\}$$

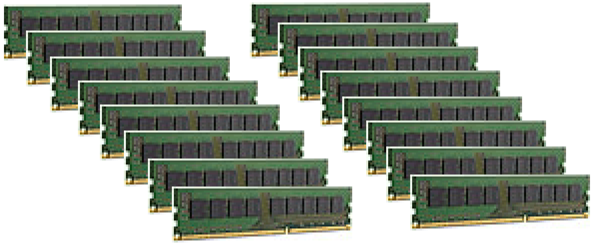
640x480

More is Better → N becomes Larger



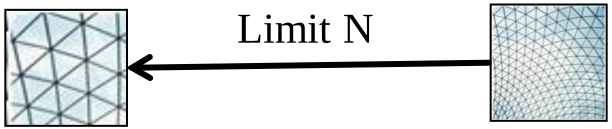
1920x1080

What limits the size of the problem (N)?



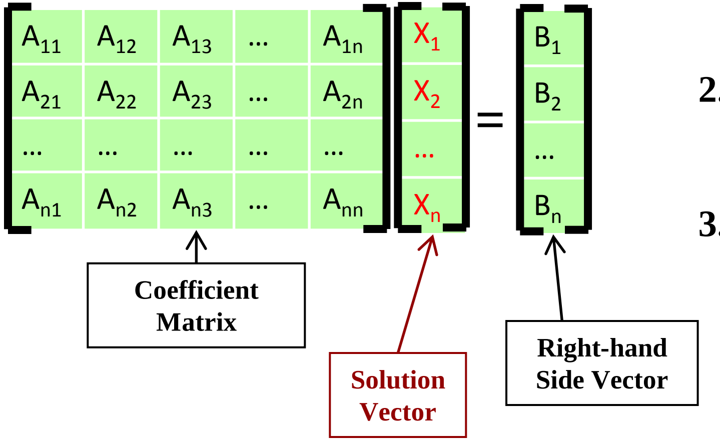
Memory

Slow Processors



Goal: Make N as big as possible <u>economically</u>

$$
\begin{aligned}
A_{11}X_1 + A_{12}X_2 + A_{13}X_3 + \ldots + A_{1n}X_n &= B_1 \\
A_{21}X_1 + A_{22}X_2 + A_{23}X_3 + \ldots + A_{2n}X_n &= B_2 \\
&\ldots \\
A_{n1}X_1 + A_{n2}X_2 + A_{n3}X_3 + \ldots + A_{nn}X_n &= B_n
\end{aligned}
$$

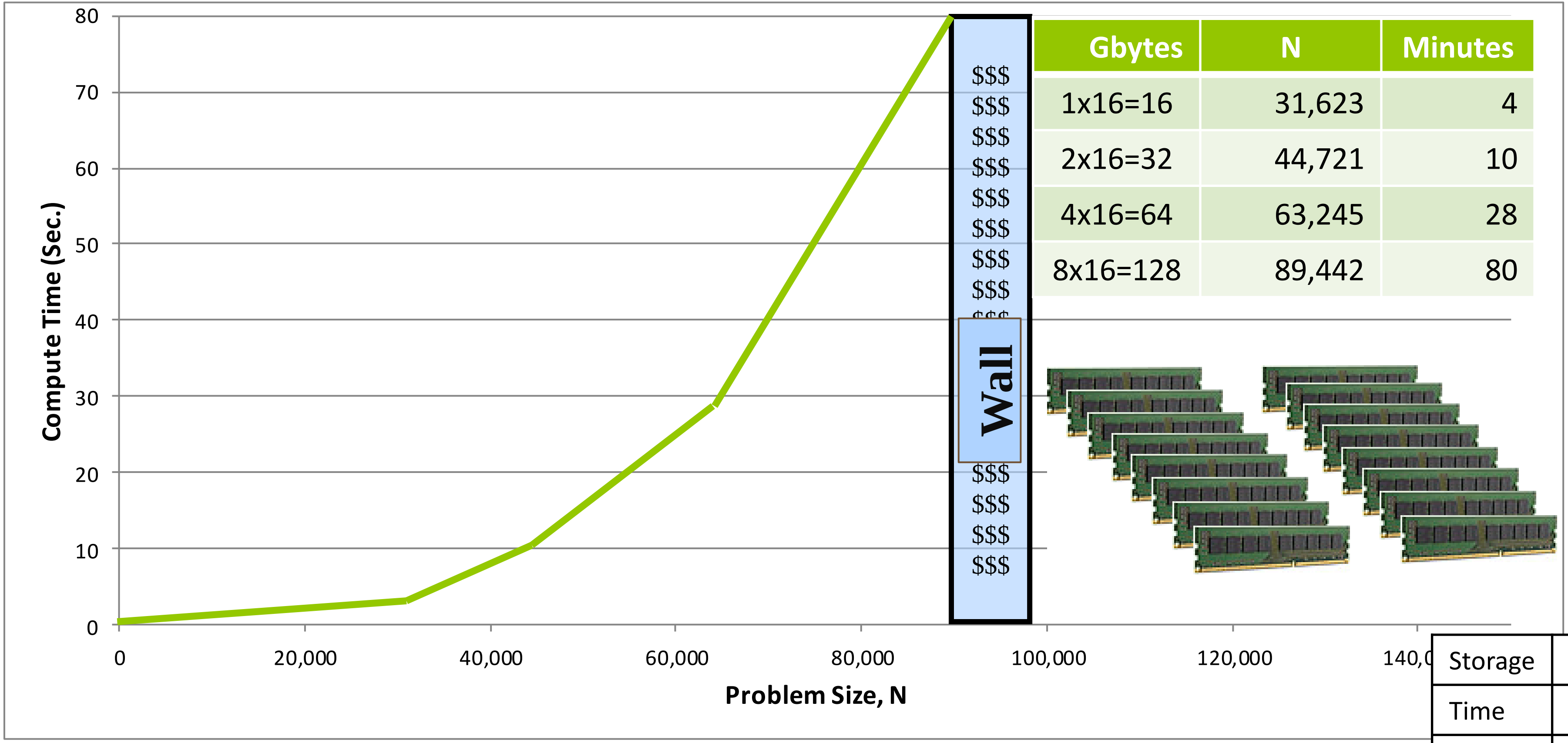


As N becomes large, it encounters three obstacles:

1. **<u>Storage</u>** for matrix A(N,N) increases as <u>N^2</u>
   - Limited by size of memory or disk
2. **<u>Computing time</u>** for [A]{X}={B} increases as <u>N^3</u>
   - Limited by processing power (CPUs, GPUs)
3. **<u>Cost/Performance</u>**
   - $/Gflop increases with performance

Storage	
Time	
Cost	

Storage: first stop-server memory



Gbytes	N	Minutes
1x16=16	31,623	4
2x16=32	44,721	10
4x16=64	63,245	28
8x16=128	89,442	80

Storage	
Time	
Cost	

Three Ways to Overcome the Storage Obstacle:

1. **Use larger very expensive memory modules**

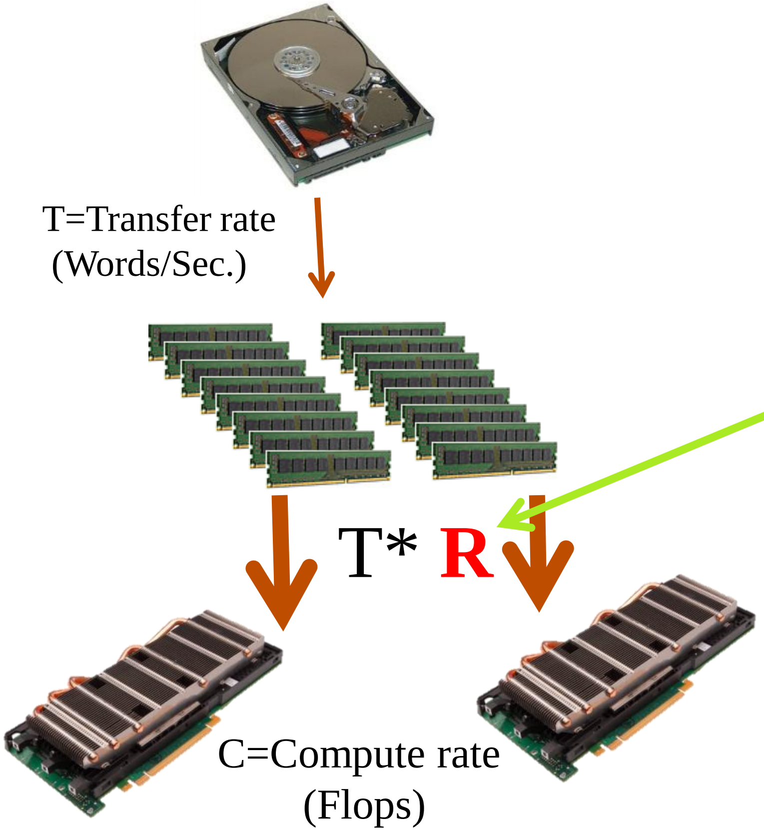
Gbytes	N	Minutes	GPU $
16x16=256	126,491	255	12X
32x16=512	178,885	636	40X

2. **Build a cluster**
   A. Replicate server
   B. Reprogram in MPI
   C. Scales storage (N^2) and compute (N^3) equally
   D. Only extends the "Wall"

3. **Store data on disk**
   A. Inexpensive (100 x less than memory)
   B. Practically unlimited size; easily added
   C. Independent scaling of storage and compute
   D. No "Wall"

Storage	
Time	
Cost	

Are Disks Fast Enough? Yes, because of ***Reuse***



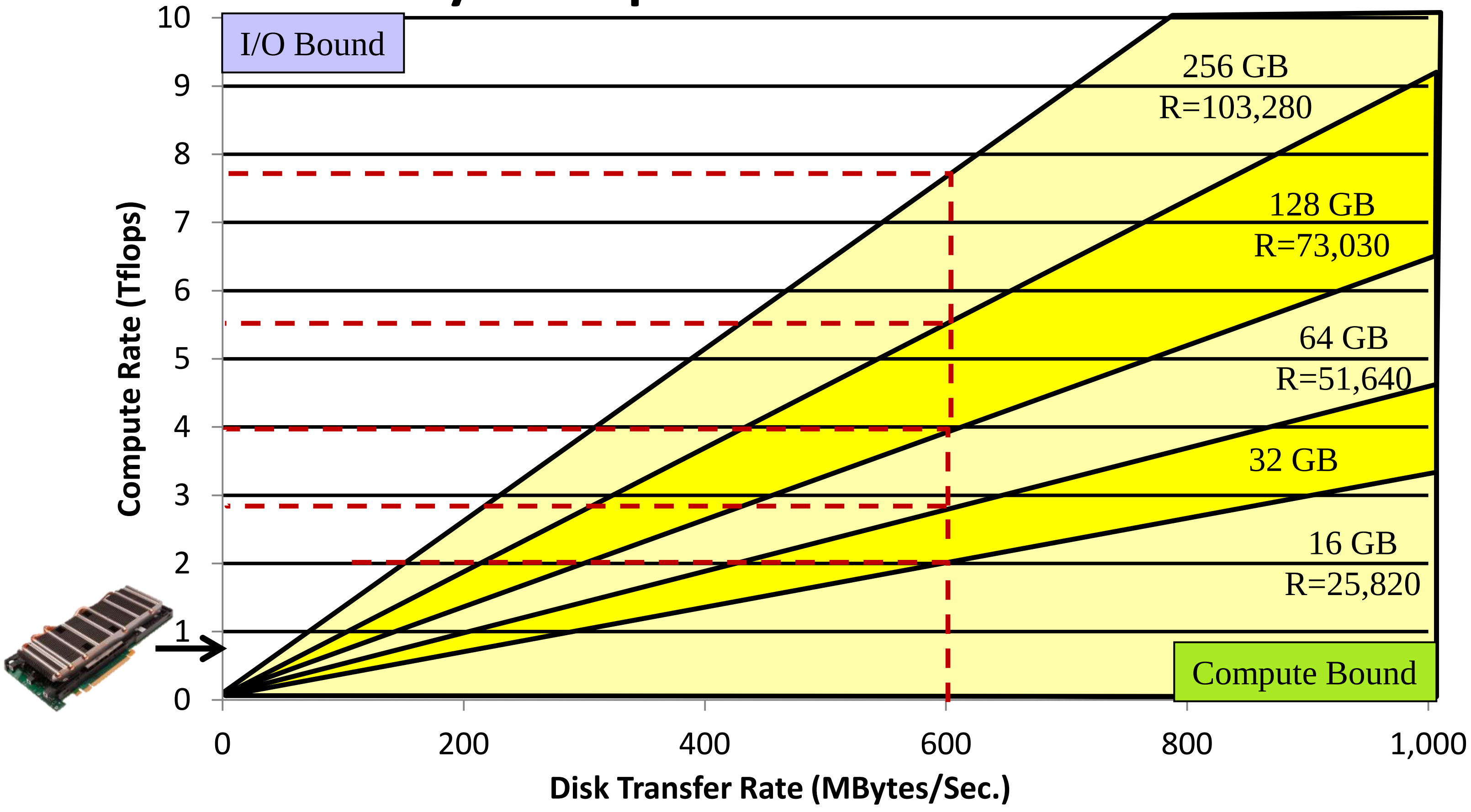
	Real	Complex
Compute $[C]=\sum[A_i][B_i]$ (Flops)	$2N^3$	$8N^3$
Read next $[A_i]$ and $[B_i]$ (Words)	$2N^2$	$4N^2$
"Reuse" R	N	2N

IO/Compute Ratio, X=(T*R)/C

- If **X > 1**, <u>Compute bound:</u>
  - Increase processing power by X
- If **X < 1**, <u>I/O bound:</u>
  1. Increase disk transfer rate to C/R
  2. Increase reuse to C/T by increasing memory
  3. Some combination of (1) and (2)

Storage	
Time	
Cost	

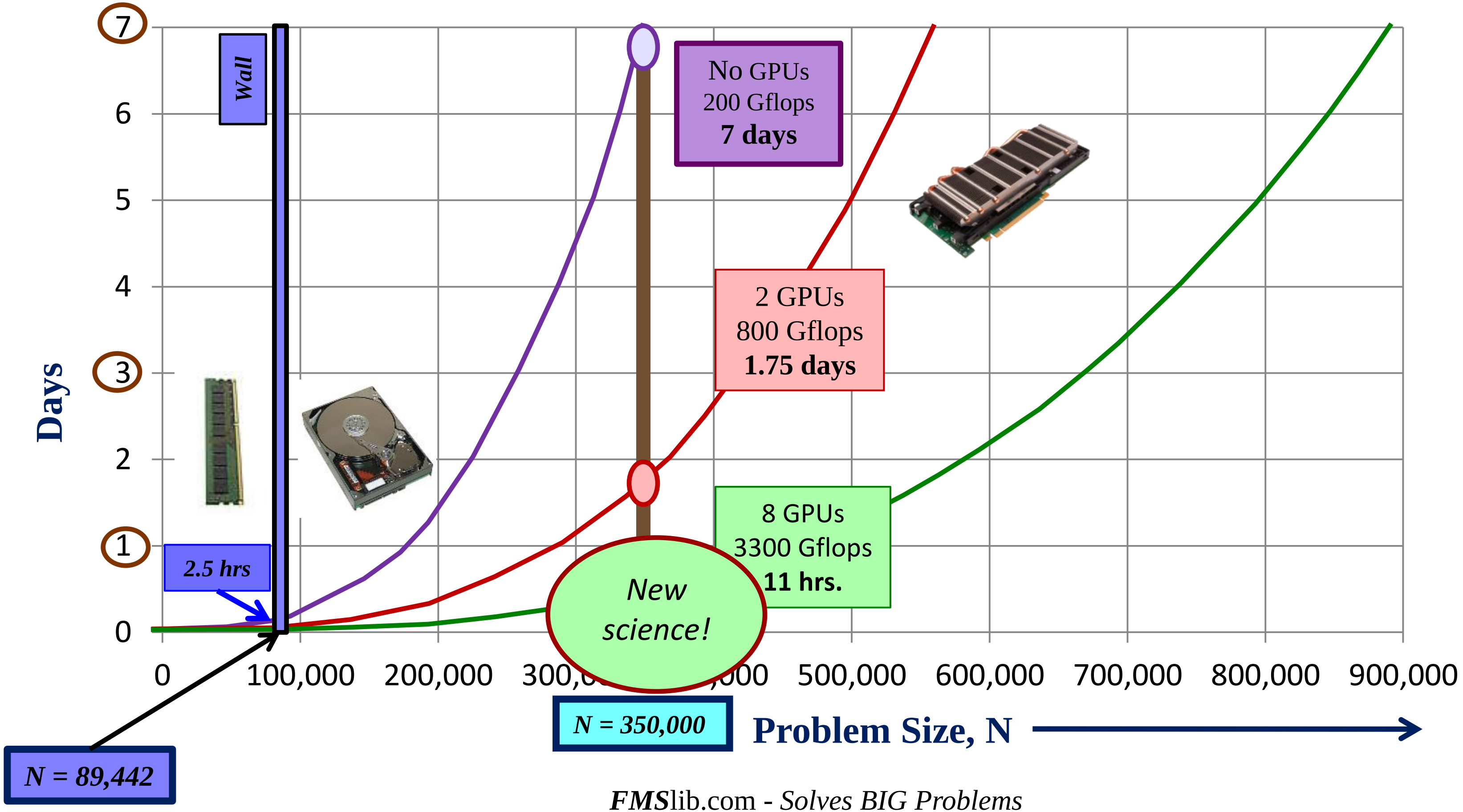
Memory Required for **Reuse** Buffer



***FMS**lib.com - Solves BIG Problems*

Storage	✔
Time	
Cost	

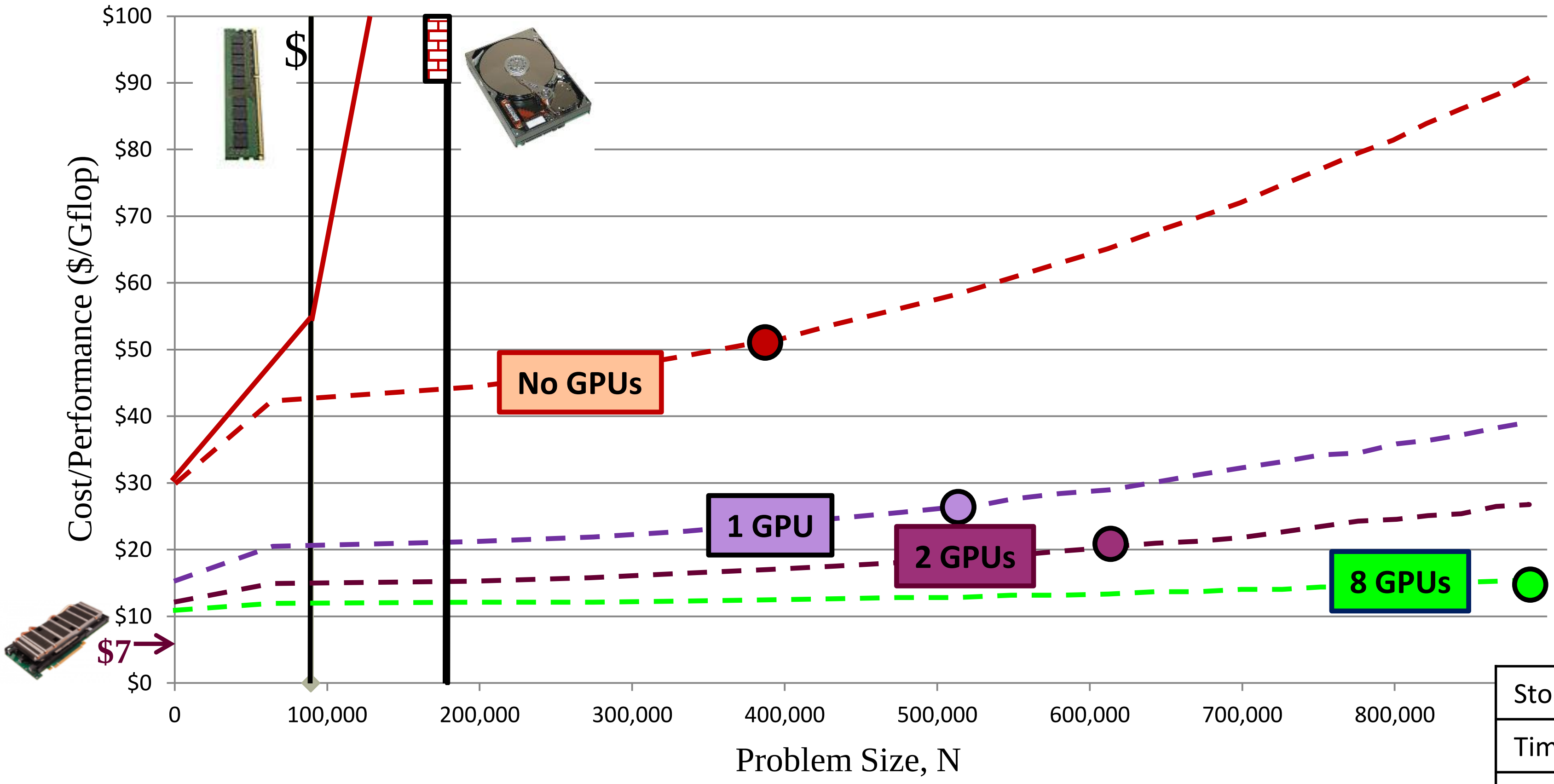
Computing Time: *What can these GPUs really DO?*



Storage	✔
Time	✔
Cost	

***FMS*lib.com** - *Solves BIG Problems*

Cost Performance *(Processing Efficiency)*



Storage	✔
Time	✔
Cost	✔

***FMS**lib.com - Solves BIG Problems*

HP Z820 Workstation

Fast Matrix Solver

Main | CPUs | GPUs | Memory | Disks | Files | Software | Calls | Parameters | Performance | Usage | Links | Help

Page updated 1304 times; Automatically refreshed every 117 sec.: Last Updated 12:02:42 Fri Apr 20 2012

Performance Analysis for Subroutine CNDF ; Time used=25:20:16

COMPLETED

85% MM | 11% TR

25:20:16

Performance (Gflops)

Routine	All	CPU's	GPU 1	GPU 2
Routine Overall	798	208	305	305

Compare CPU and GPU Performance

I/O Wait < 5%

798 Gflops

Legend

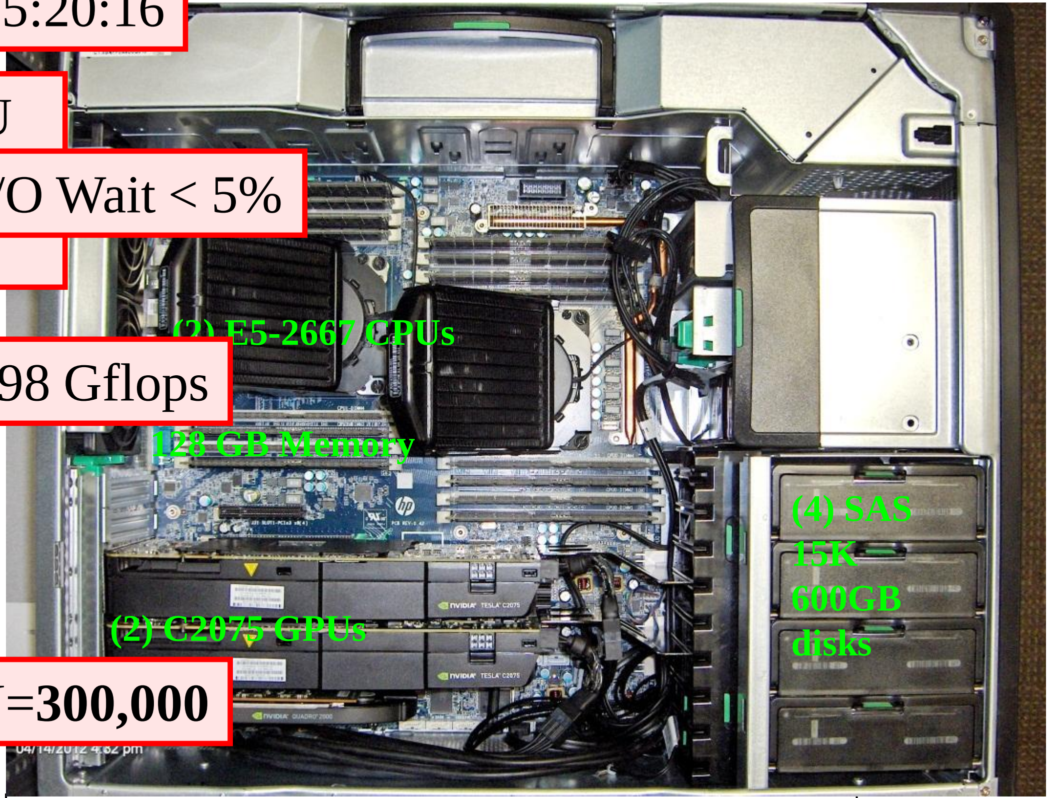
Undefined | Assembled | Factored | Reduced | Solved | Multiplied | Accessing | Computing

L1,1	U1,2	U1,3	U1,4	U1,5	U1,6	U1,7	U1,8	U1,9	U1,10	U1,11	U1,12
L2,1	L2,2	U2,3	U2,4	U2,5	U2,6	U2,7	U2,8	U2,9	U2,10	U2,11	U2,12
L3,1	L3,2	L3,3	U3,4	U3,5	U3,6	U3,7	U3,8	U3,9	U3,10	U3,11	U3,12
L4,1	L4,2	L4,3						U4,9	U4,10	U4,11	U4,12
L5,1	L5,2	L5,3						U5,9	U5,10	U5,11	U5,12
L6,1	L6,2	L6,3						U6,9	U6,10	U6,11	U6,12
L7,1	L7,2	L7,3						U7,9	U7,10	U7,11	U7,12
L8,1	L8,2	L8,3						U8,9	U8,10	U8,11	U8,12
L9,1	L9,2	L9,3	L9,4	L9,5	L9,6	L9,7	L9,8	L9,9	U9,10	U9,11	U9,12
L10,1	L10,2	L10,3	L10,4	L10,5	L10,6	L10,7	L10,8	L10,9	L10,10	U10,11	U10,12
L11,1	L11,2	L11,3	L11,4	L11,5	L11,6	L11,7	L11,8	L11,9	L11,10	L11,11	U11,12
L12,1	L12,2	L12,3	L12,4	L12,5	L12,6	L12,7	L12,8	L12,9	L12,10	L12,11	L12,12

Watch matrix being solved in real time.

Times and Problem

Job Started	0	10:41:24 Thu Apr 19 2012	Equations	300000
Routine Started	61	10:42:26 Thu Apr 19 2012	Vectors	0
Current Time	91278	12:02:42 Fri Apr 20 2012	Block Size	25088 X 25088
Estimated Completion	91278	12:02:43 Fri Apr 20 2012	Data Type	Complex



N=300,000

Reuse=(2)(25088)=**50,176**; T=(700)/8=87.5 MW/Sec.

I/O Compute Ratio = (50,176)(87.5)/798000 = **5.50**

.com - *Solves BIG Problems*

Main | CPUs | GPUs | Memory | Disks | Files | Software | Calls | Parameters | Performance | Usage | Links | Help

[Matrix] Warrior *Powered by FMSlib*

Performance Analysis for Subroutine CNDF | *Time used=28:36:47*

COMPLETED

89% MM | 7% TR

Performance (Gflops)

Routine	All	CPU's	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7	GPU 8
Routine Overall	3315	154	401	413	413	410	401	403	405	403

Legend

Undefined | Assembled | Factored | Reduced | Solved | Multiplied | Accessing | Computing

L1,1	U1,2	U1,3	U1,4	U1,5	U1,6	U1,7	U1,8	U1,9	U1,10	U1,11	U1,12	U1,13	U1,14	U1,15	U1,16	U1,17	U1,18	U1,19
L2,1	L2,2	U2,3	U2,4	U2,5	U2,6	U2,7	U2,8	U2,9	U2,10	U2,11	U2,12	U2,13	U2,14	U2,15	U2,16	U2,17	U2,18	U2,19
L3,1	L3,2	L3,3	U3,4	U3,5	U3,6	U3,7	U3,8	U3,9	U3,10	U3,11	U3,12	U3,13	U3,14	U3,15	U3,16	U3,17	U3,18	U3,19
L4,1	L4,2	L4,3	L4,4	U4,5	U4,6	U4,7	U4,8	U4,9	U4,10	U4,11	U4,12	U4,13	U4,14	U4,15	U4,16	U4,17	U4,18	U4,19
L5,1	L5,2	L5,3	L5,4	L5,5	U5,6	U5,7	U5,8	U5,9	U5,10	U5,11	U5,12	U5,13	U5,14	U5,15	U5,16	U5,17	U5,18	U5,19
L6,1	L6,2	L6,3	L6,4	L6,5	L6,6	U6,7	U6,8	U6,9	U6,10	U6,11	U6,12	U6,13	U6,14	U6,15	U6,16	U6,17	U6,18	U6,19
L7,1	L7,2	L7,3	L7,4	L7,5	L7,6	L7,7	U7,8	U7,9	U7,10	U7,11	U7,12	U7,13	U7,14	U7,15	U7,16	U7,17	U7,18	U7,19
L8,1	L8,2	L8,3	L8,4	L8,5	L8,6	L8,7	L8,8	U8,9	U8,10	U8,11	U8,12	U8,13	U8,14	U8,15	U8,16	U8,17	U8,18	U8,19
L9,1	L9,2	L9,3	L9,4	L9,5	L9,6	L9,7	L9,8	L9,9	U9,10	U9,11	U9,12	U9,13	U9,14	U9,15	U9,16	U9,17	U9,18	U9,19
L10,1	L10,2	L10,3	L10,4	L10,5	L10,6	L10,7	L10,8	L10,9	L10,10	U10,11	U10,12	U10,13	U10,14	U10,15	U10,16	U10,17	U10,18	U10,19
L11,1	L11,2	L11,3	L11,4	L11,5	L11,6	L11,7	L11,8	L11,9	L11,10	L11,11	U11,12	U11,13	U11,14	U11,15	U11,16	U11,17	U11,18	U11,19
L12,1	L12,2	L12,3	L12,4	L12,5	L12,6	L12,7	L12,8	L12,9	L12,10	L12,11	L12,12	U12,13	U12,14	U12,15	U12,16	U12,17	U12,18	U12,19
L13,1	L13,2	L13,3	L13,4	L13,5	L13,6	L13,7	L13,8	L13,9	L13,10	L13,11	L13,12	L13,13	U13,14	U13,15	U13,16	U13,17	U13,18	U13,19
L14,1	L14,2	L14,3	L14,4	L14,5	L14,6	L14,7	L14,8	L14,9	L14,10	L14,11	L14,12	L14,13	L14,14	U14,15	U14,16	U14,17	U14,18	U14,19
L15,1	L15,2	L15,3	L15,4	L15,5	L15,6	L15,7	L15,8	L15,9	L15,10	L15,11	L15,12	L15,13	L15,14	L15,15	U15,16	U15,17	U15,18	U15,19
L16,1	L16,2	L16,3	L16,4	L16,5	L16,6	L16,7	L16,8	L16,9	L16,10	L16,11	L16,12	L16,13	L16,14	L16,15	L16,16	U16,17	U16,18	U16,19
L17,1	L17,2	L17,3	L17,4	L17,5	L17,6	L17,7	L17,8	L17,9	L17,10	L17,11	L17,12	L17,13	L17,14	L17,15	L17,16	L17,17	U17,18	U17,19
L18,1	L18,2	L18,3	L18,4	L18,5	L18,6	L18,7	L18,8	L18,9	L18,10	L18,11	L18,12	L18,13	L18,14	L18,15	L18,16	L18,17	L18,18	U18,19
L19,1	L19,2	L19,3	L19,4	L19,5	L19,6	L19,7	L19,8	L19,9	L19,10	L19,11	L19,12	L19,13	L19,14	L19,15	L19,16	L19,17	L19,18	L19,19

Times and Problem

Job Started	0	08:15:22 Wed Oct 31 2012	Equations	500000
Routine Started	3	08:15:26 Wed Oct 31 2012	Vectors	0
Current Time	103010	12:52:13 Thu Nov 01 2012	Block Size	26624 X 26624
Estimated Completion	103011	12:52:13 Thu Nov 01 2012	Data Type	Complex

Compute I/O Ratio

Compute Rate (Gflops)	3315
Read Rate (MBytes/Sec.)	637
Reuse	53248
IO Compute Ratio	1.28

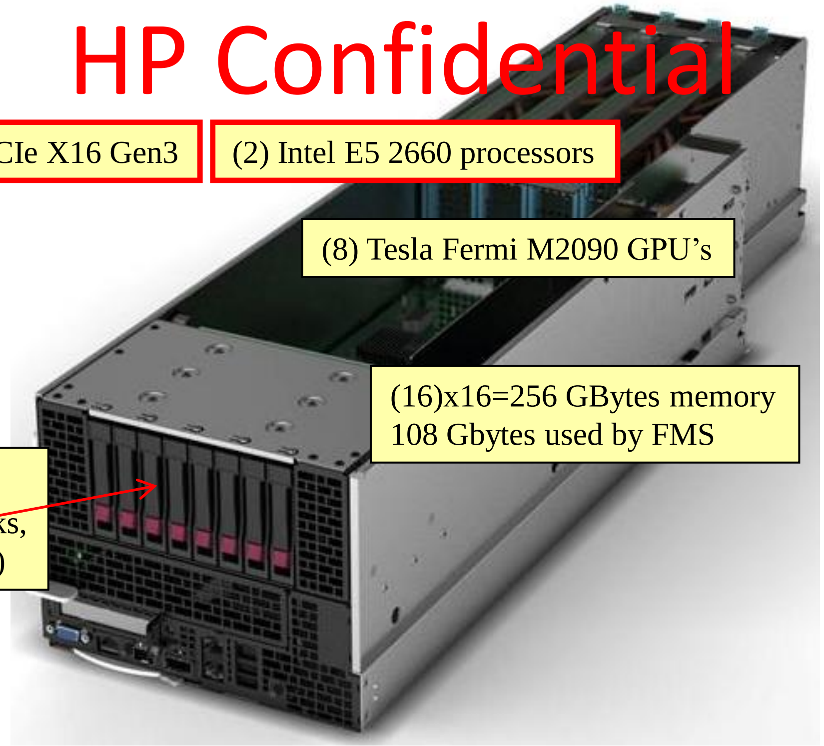
Page updated 3 times: Automatically refreshed every 11 sec.: Last Updated 12:52:13 Thu Nov 01 2012

28:36:47

I/O Wait < 5%

3.315 Tflops

HP SL270 4U Server
HP Confidential



PCIe X16 Gen3

(2) Intel E5 2660 processors

(8) Tesla Fermi M2090 GPU’s

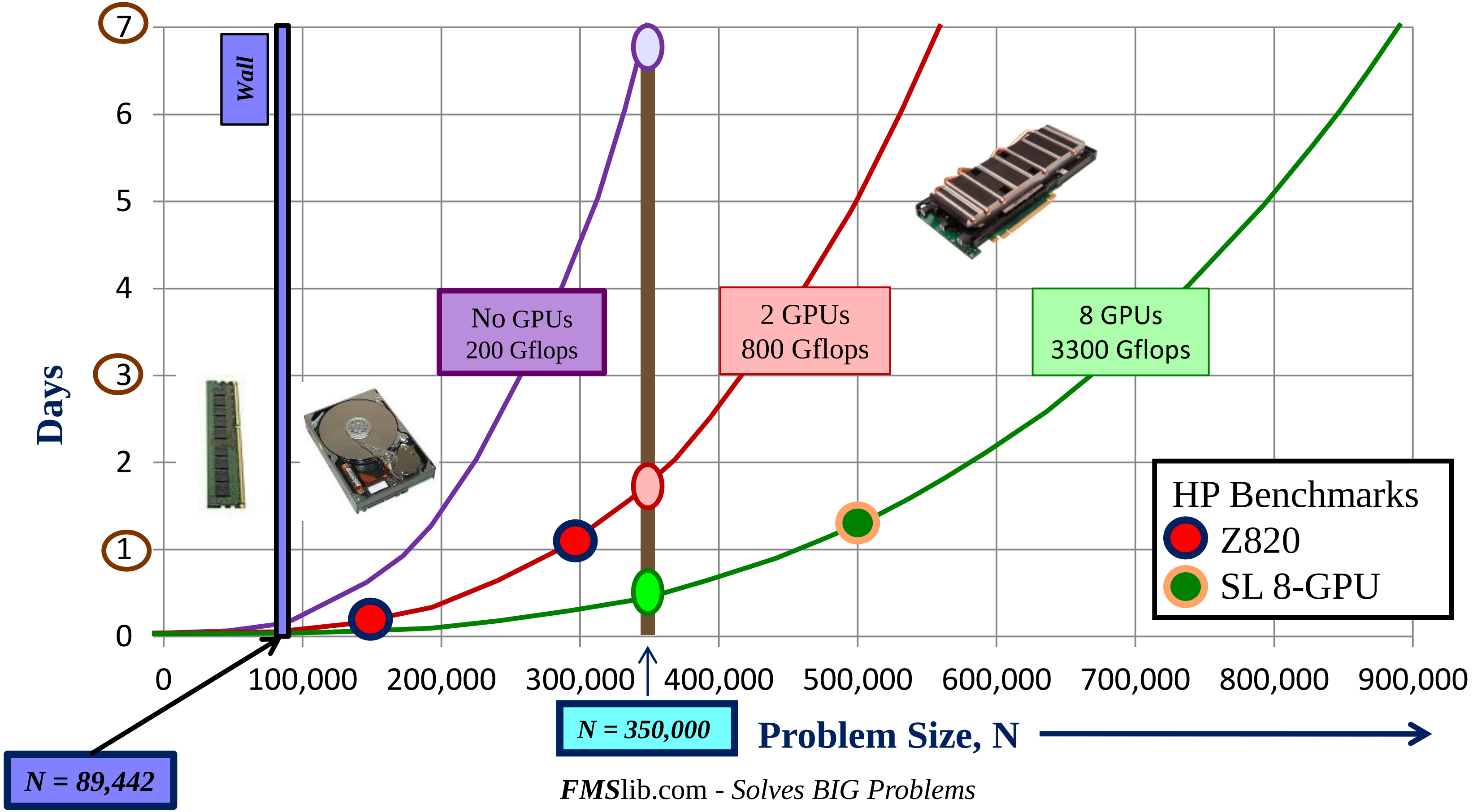
(16)x16=256 GBytes memory
108 Gbytes used by FMS

(7)x1TB=**7** TB
7.2K SATA Disks,
(637MB/Sec.)

N=500,000

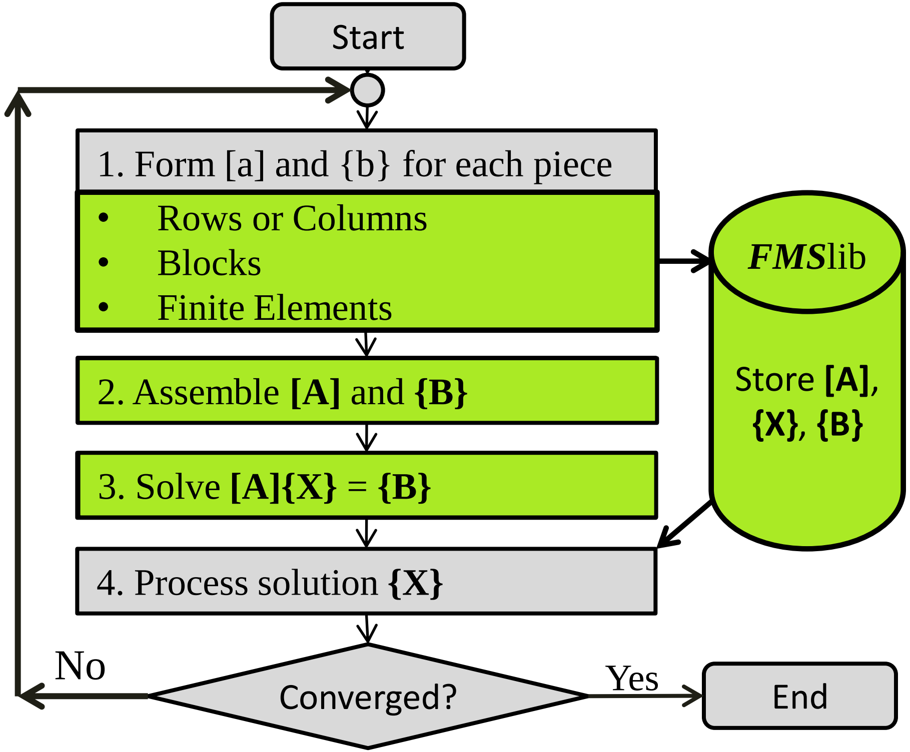
I/O Compute Ratio = (53248)(79.6)/3315000 = **1.28**

HP Benchmarks



***FMS*lib.com** - *Solves BIG Problems*

Application Program Interface: *Disk-Based*



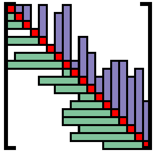
Why ***FMS*lib**?

1. ***FMS*lib** is based on an in-depth understanding of mathematics and computer architecture, incorporating no shortcuts. *Performance is obtained by exploiting all hardware features.*

2. ***FMS*lib** was the first linear algebra library, initially introduced in 1982 by Floating Point Systems to accelerate their array processors.

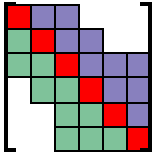
3. ***FMS*lib** includes only those routines that have proven commercial value.

***FMS*lib** Solvers



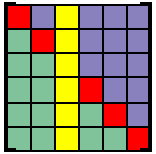
Profile

PROFILE SOLVER: Accounts for the sparsity of matrix **[A]** on an equation by equation basis



Block

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



Slab

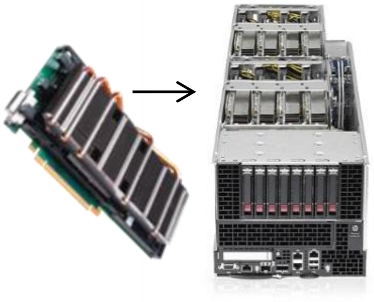
SLAB SOLVER: Extends the functionality of the Block Solver by providing full column partial pivoting for full nonsymmetric matrices.

Data Type	8-byte Real 16-byte Complex
Matrix Symmetry	Symmetric Nonsymmetric Hermitian
Direct	No iteration Predictable performance
Dense	No indirect addressing Maximum GPU performance
Out-of-core	Option to use disk for data storage
Multiple Solutions	Efficiently solves for multiple {X}
GPUs	**Plug-and-play**
OS	**Linux, Windows**

***FMS*lib** Performance History 1978-present

Machine	Year	Flops	N*	$/Gflop
DEC VAX	**1978**	97,000	1,465	2,000,000,000
FPS 164	**1982**	11,000,000	7,090	50,000,000
FPS 164-MAX	**1985**	341,000,000	22,272	2,500,000
(8) NVIDIA GPUs	**2012**	3,300,000,000,000	474,627	15

* Factor full complex nonsymmetric matrix in 1 day



34 Million times faster!
133 Million times cheaper!

Demonstrate your GPU Performance with

[Matrix] Warrior

Powered by ***FMSlib***

1. Demonstrate new computer technology
2. Benchmark performance studies (CPUs, GPUs, Memory, Disks)
3. Assess existing machine performance (your laptop to GPU server).

Free download from ***FMS*lib** .com

Conclusions

1. Larger matrices result from more detailed analysis.

2. Matrix Algebra scales differently than most applications:
   - Storage as N2
   - Processing as N3

3. High **<u>Reuse</u>** in matrix algebra allows efficient use of multilevel memory systems:
   - Inexpensive disks can be used for storage
   - Overlapped transfers from Disk → Memory → (GPUs) → Cache → Registers
   - Processors continuously operate at near peak speed

4. GPUs have an ideal architecture for matrix algebra:
   - High performance
   - Lower capital and operational costs

5. [Matrix]Warrior may be used for
   - Machine benchmarking
   - Demonstrating performance
   - Machine burn in

It’s a new day in scientific computing

FMSlib.com

[Matrix] Warrior

Solve **[A]{X} = {B}**

3 + Tflops Sustained!, Low $/Gflop

Solves BIGGER Problems