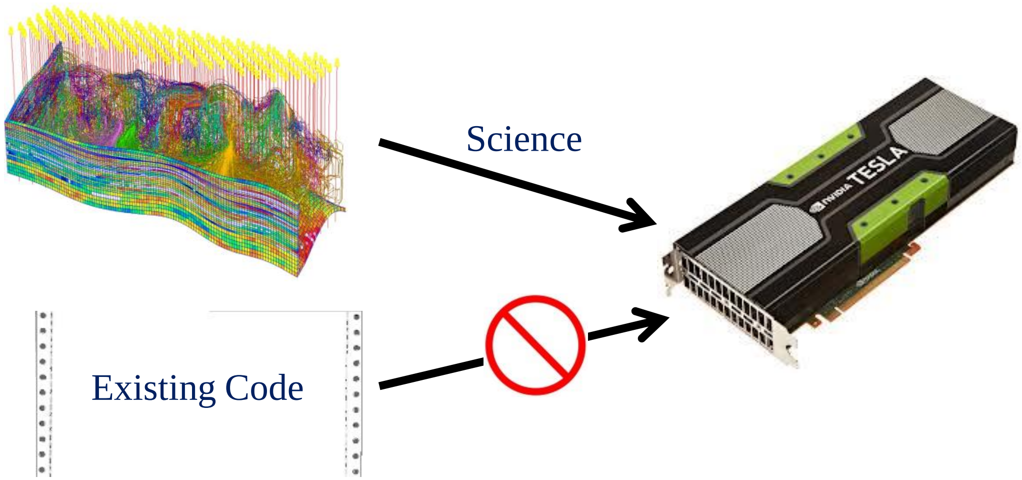
HP-CAST, June 21, 2014

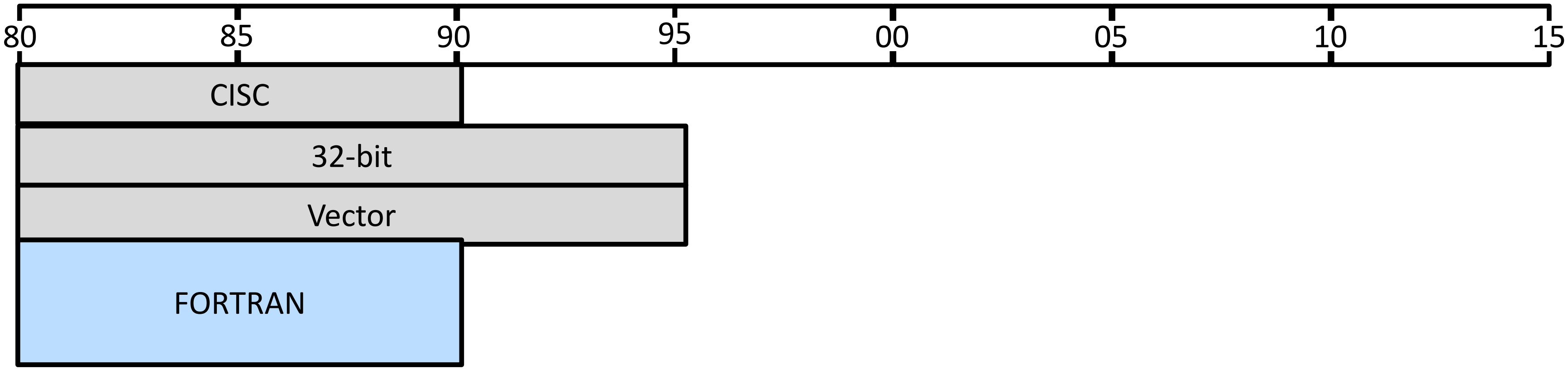
Dr. Ronald Young

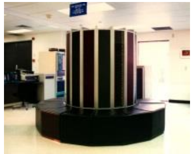
Port the Science, Not the Code.

Application and library requirements for maximum accelerator performance.



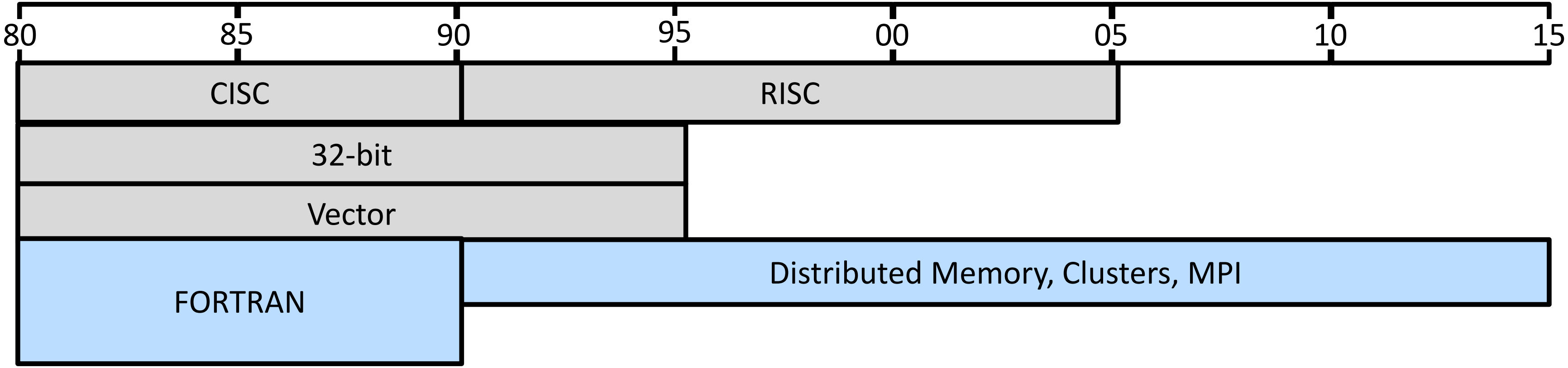
Hardware & Software History

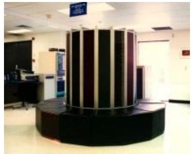




Processor Speed ~ Memory Speed

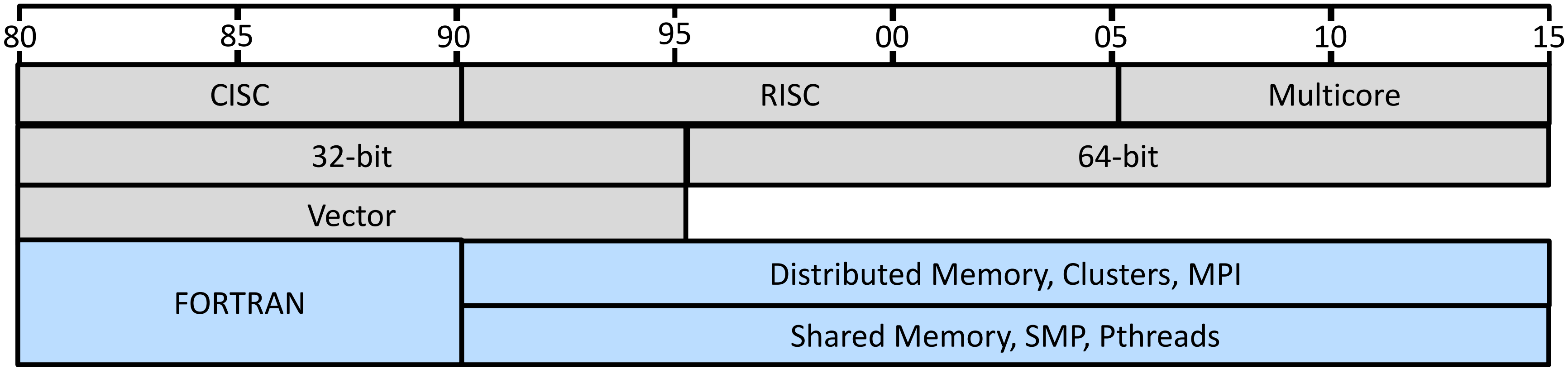
Hardware & Software History

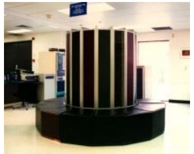




Processor on a single chip
Lower cost/calculation
Potential for increasing processor speed

Hardware & Software History



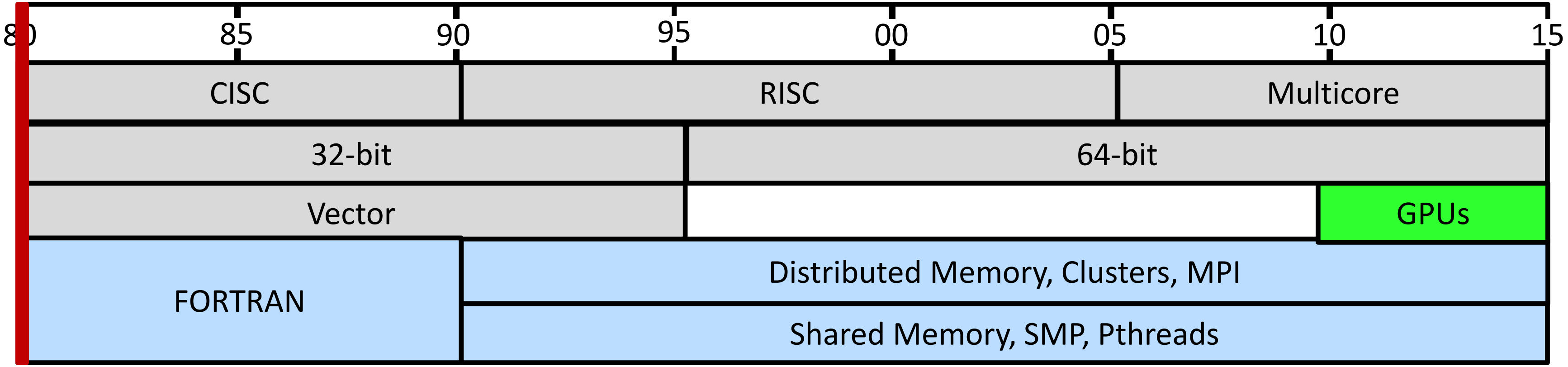


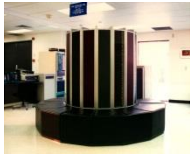
2 programming models emerge:

- Distributed memory
- Shared memory

64-bit addressing becomes available

Hardware & Software History





Floating point limited

86,000,000 Times Faster
166,000,000 Times Cheaper

Data limited

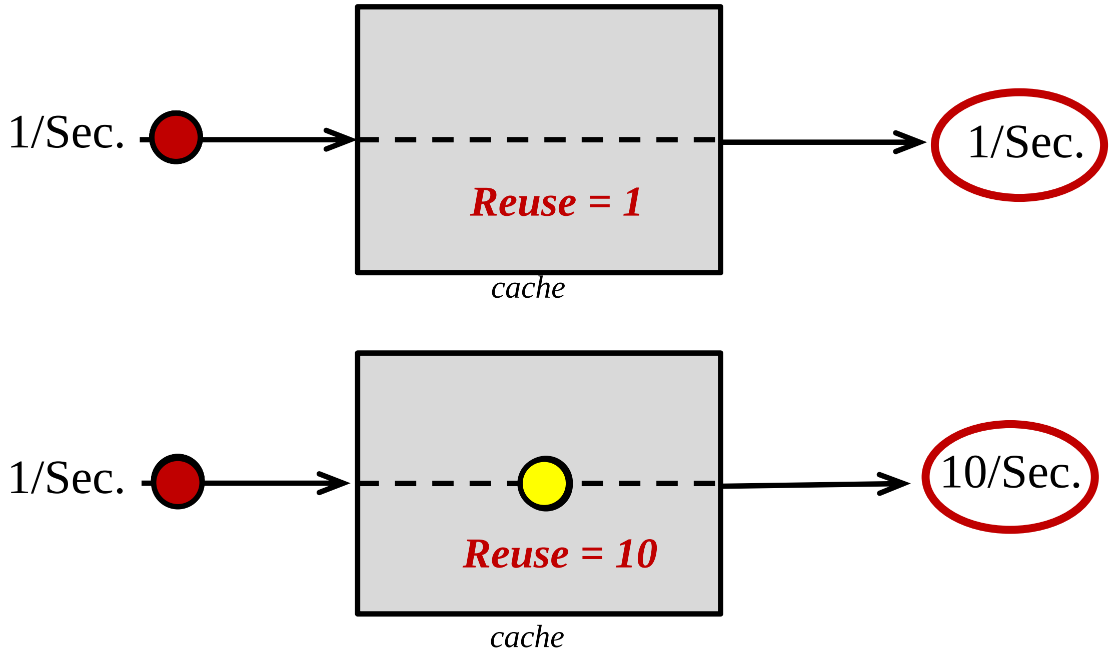
When was your software designed?

Reuse *Efficiently linking slower input to faster requirement*

- Reuse = Number of times a data element is used in a computation.

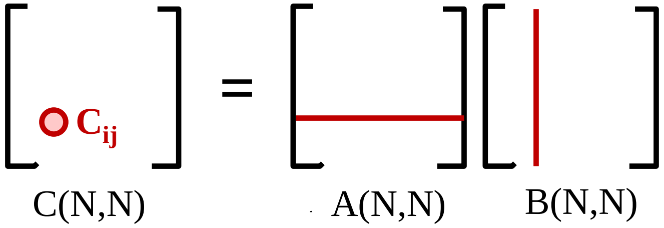
Old machines: Low Reuse

New machines: High Reuse



BLAS 1 Reuse

When M=N=K

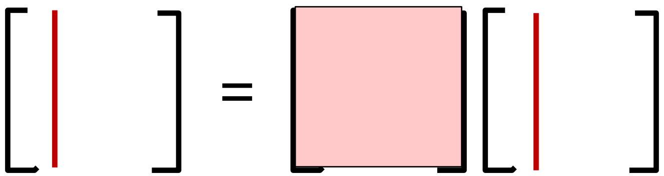



```
    DO 30 J=1,N
     DO 20 I=1,M
      DO 10 L=1,K
       C(I,J) = A(I,L)*B(L,J)
10     CONTINUE
20   CONTINUE
30  CONTINUE
```

BLAS	Function	Data	Flops	Reuse
1	DDOT	2N+1	2N	**1**

$$\frac{\text{Flops}}{\text{Data}} = \frac{2N}{(2N+1)} \sim 1$$

BLAS 2 Reuse



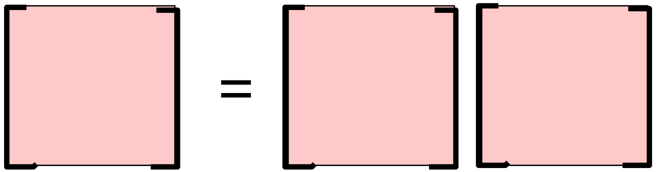
C(N,N) A(N,N) B(N,N)

```
    DO 30 J=1,N
     DO 20 I=1,M
      DO 10 L=1,K
       C(I,J) = A(I,L)*B(L,J)
10     CONTINUE
20   CONTINUE
30 CONTINUE
```

BLAS	Function	Data	Flops	Reuse
1	DDOT	2N+1	2N	**1**
2	DGEMV	N^2+2N	$2N^2$	**2**

$$\frac{\text{Flops}}{\text{Data}} = \frac{2N^2}{(N^2+2N)} \sim 2$$

BLAS 3 Reuse




```
   DO 30 J=1,N
    DO 20 I=1,M
     DO 10 L=1,K
      C(I,J) = A(I,L)*B(L,J)
10     CONTINUE
20   CONTINUE
30  CONTINUE
```

BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	**1**
2	DGEMV	N^2+2N	$2N^2$	**2**
3	DGEMM	$3N^2$	$2N^3$	**2N/3**

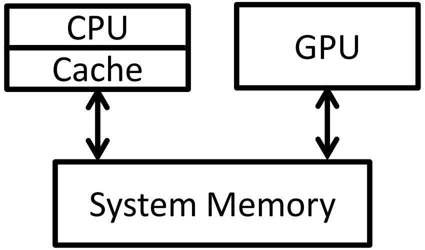
$$\frac{\text{Flops}}{\text{Data}} = \frac{2N^3}{3N^2} = \frac{2N}{3}$$

BLAS 3 Reuse

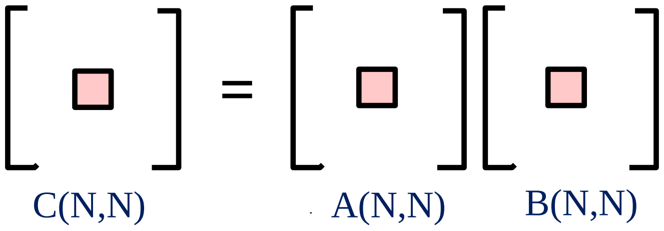
$$C(N,N) = A(N,N)\,B(N,N)$$

```
 DO 30 J=1,N
  DO 20 I=1,M
   DO 10 L=1,K
    C(I,J) = A(I,L)*B(L,J)
10    CONTINUE
20  CONTINUE
30 CONTINUE
```

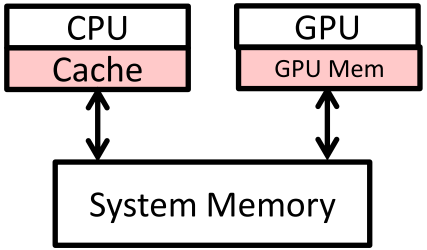
BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	**1**
2	DGEMV	N^2+2N	$2N^2$	**2**
3	DGEMM	$3N^2$	$2N^3$	**2N/3**



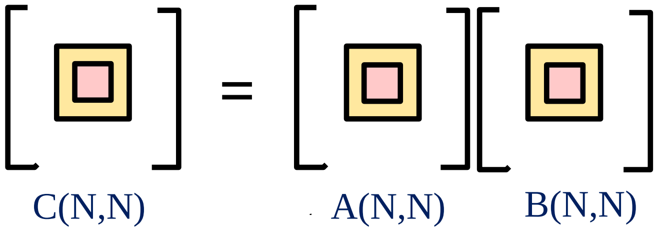
Reuse – Small Data



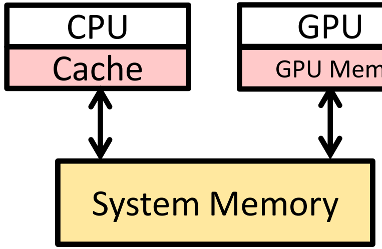
BLAS	Function	Data	Flops	Reuse
1	DDOT	$2N+1$	$2N$	1
2	DGEMV	N^2+2N	$2N^2$	2
3	DGEMM	$3N^2$	$2N^3$	$2N/3$



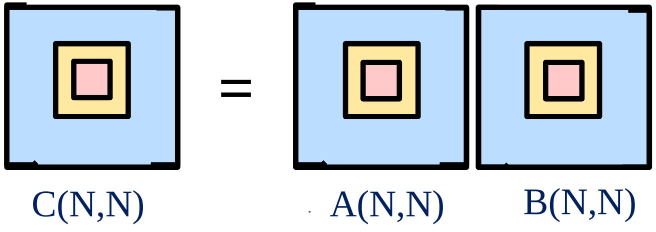
Reuse - More Data



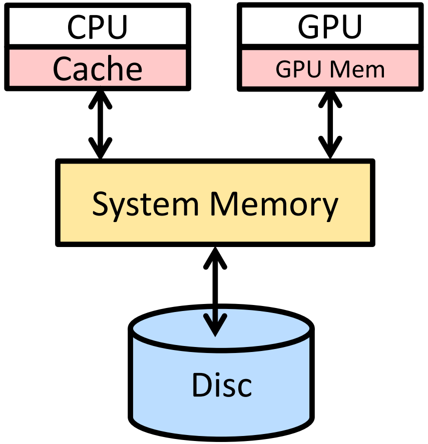
BLAS	Function	Data	Flops	Reuse
~~1~~	~~DDOT~~	~~2N+1~~	~~2N~~	~~1~~
~~2~~	~~DGEMV~~	~~N^2+2N~~	~~$2N^2$~~	~~2~~
3	DGEMM	$3N^2$	$2N^3$	2N/3



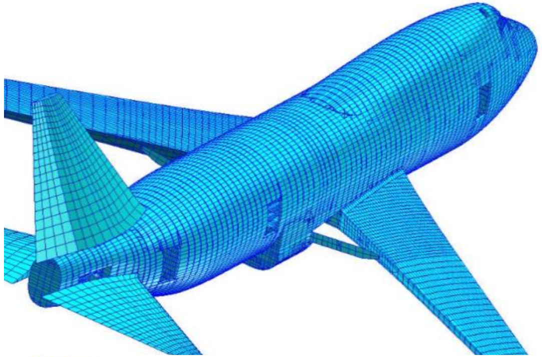
Reuse – Big Data

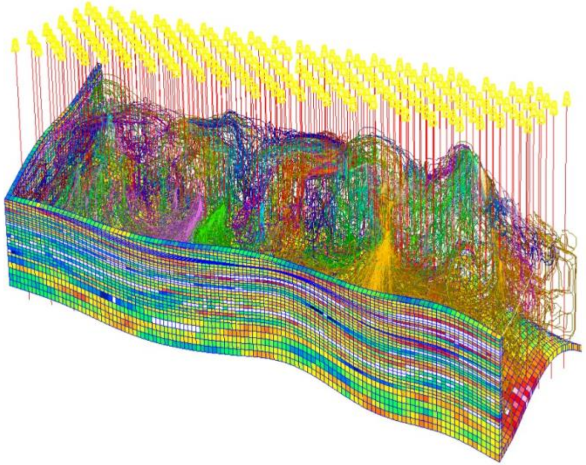


BLAS	Function	Data	Flops	Reuse
~~1~~	~~DDOT~~	~~$2N+1$~~	~~$2N$~~	~~1~~
~~2~~	~~DGEMV~~	~~N^2+2N~~	~~$2N^2$~~	~~2~~
3	DGEMM	$3N^2$	$2N^3$	$2N/3$



Application Development





Discretization

1. Domain is divided into a large number of pieces.
2. Equations are formed for each piece **[a], {b}**
3. Pieces are assembled into a global system of equations

   [A]{X} = {B}

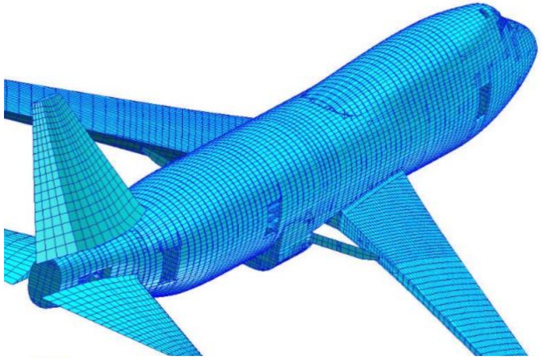
4. The system is then solved for the solution **{X}**

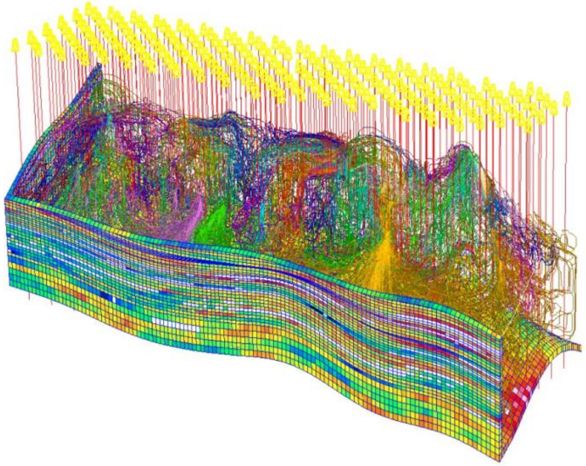
More pieces gives better answer

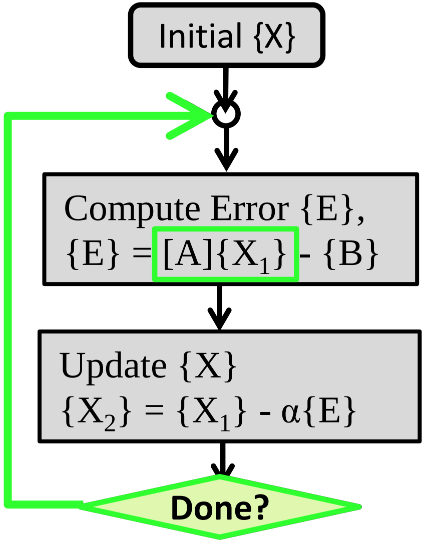
More pieces = more data and processing

∴ Machine (speed, storage) determines accuracy

Solving [A]{X}={B} *Explicitly*





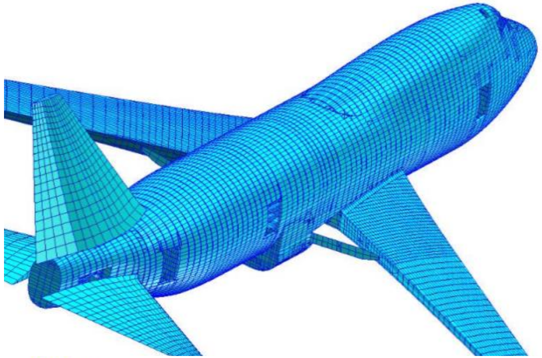


(+) Minimum data
(-) Low reuse: BLAS 2
(-) Many iterations
(-) Convergence risk
(-) Data dependent
(-) Unpredictable run times
(-) Stability -> small increments
(-) Multiple solutions
(-) Indirect addressing

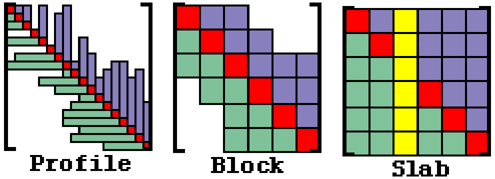
<u>Expertise required for:</u>

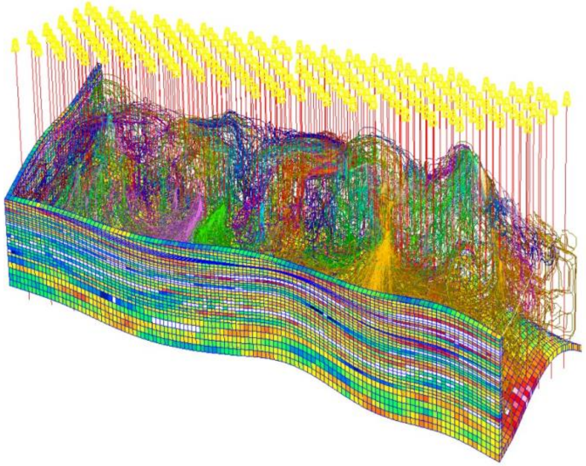
- Convergence criteria
- Selecting initial {X}
- Selecting α

Solving [A]{X}={B} *Implicitly*



Solve
[A]{X} = {B}
Factor [A]
Solve for {X}



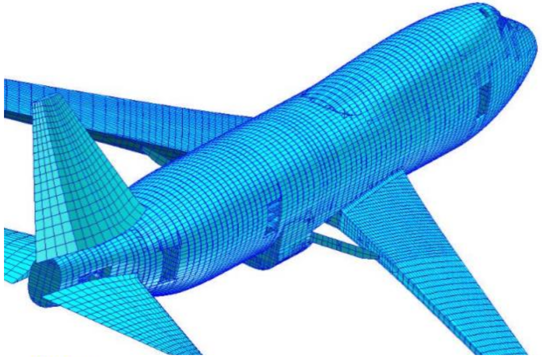


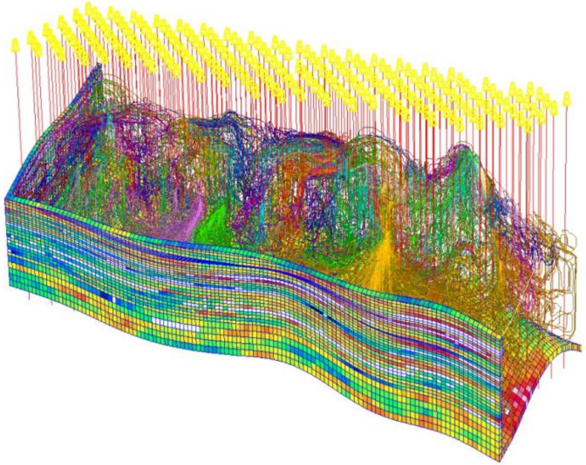
(-) Fill data
(+) High reuse: BLAS 3
(+) One iteration
(+) No convergence risk
(+) Data independent
(+) Predictable run times
(+) Multiple solutions
(+) Unconditionally stable
(+) Incremental addressing

<u>Additional advantages:</u>

- Production applications (no "expertise" required)
- Modern computers (hierachical memory systems)

Solving [A]{X}={B} *Implicitly*





1. Form [a] and {b} for each piece
   - Rows or Columns
   - Blocks
   - Finite Elements
2. Build global model **[A], {B}**
3. Solve **[A]{X}** = **{B}**
4. Process solution **{X}**

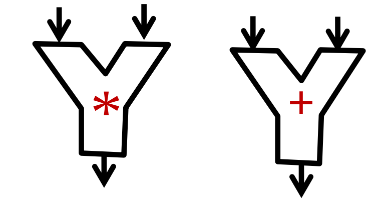
***FMS*lib**

Store **[A]**, **{X}**, **{B}**

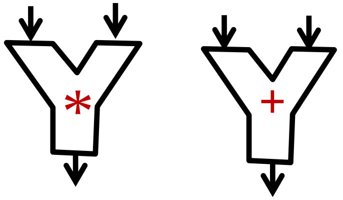
Explicit vs Implicit

	Explicit	Implicit
Data	Minimum	+ Fill
Reuse	Low (2)	High (BLAS 3)
Iterations	Many	1
Convergence	Expert required	Automatic
Data Independent	No	Yes
Run times	Variable	Predictable
Multiple solutions	No	Yes
Stability	Small increments	Unconditional
Addressing	Slow (Indirect)	Fast (Incremental)

Hardware Reuse

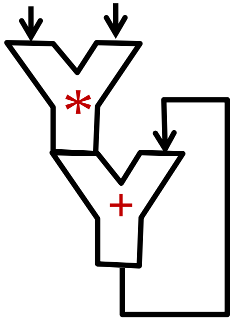


(6)(8)/2 = 24 Bytes/flop



Fused Multiply-Add

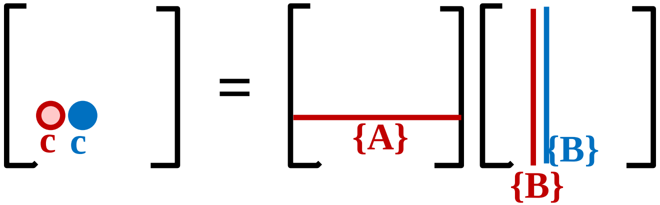
(4)(8)/2 = 16 Bytes/flop

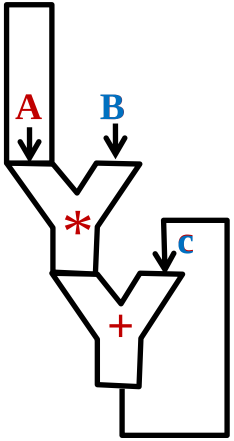


Dot Product

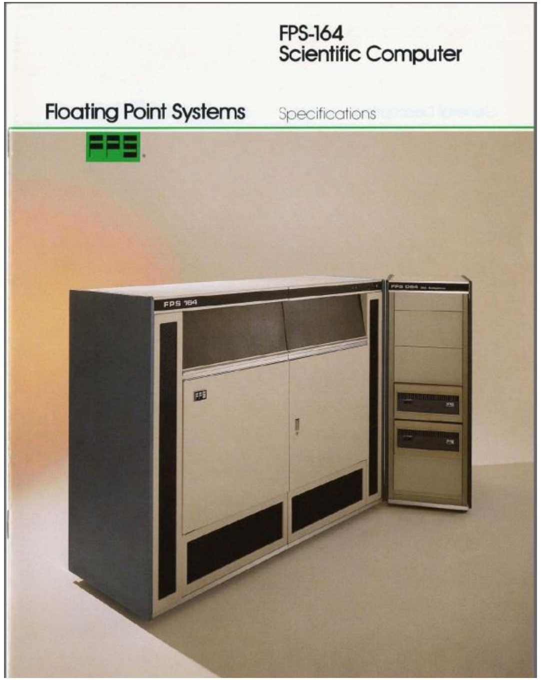
(2)(8)/2 = 8 Bytes/flop

Hardware Reuse



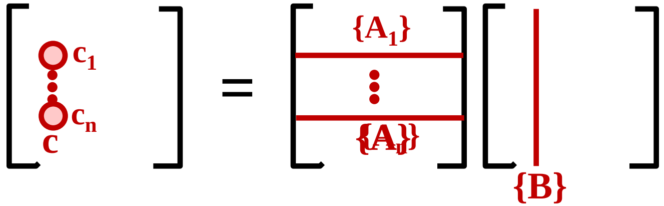


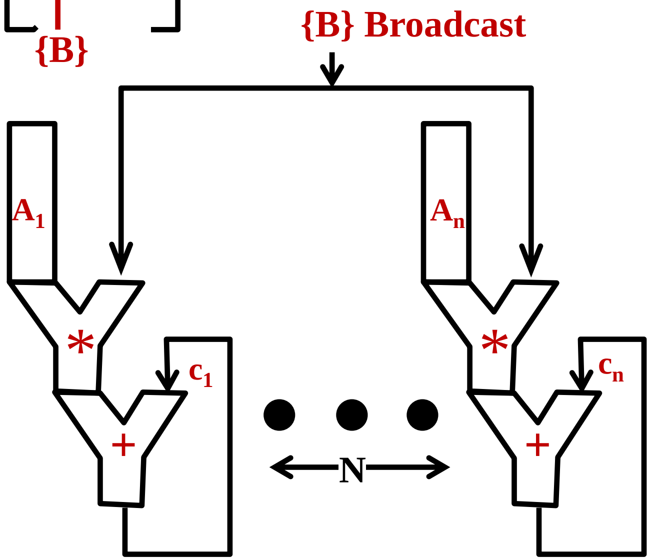
Vector Dot Product
(1)(8)/4 = 4 Bytes/flop



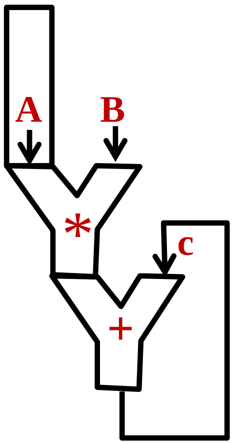
1981 Floating Point Systems 164
First 64-bit accelerator
First version of FMSlib

Hardware Reuse





1985 FMSlib Accelerator



Vector Dot Product

(1)(8)/4 = 4 Bytes/flop

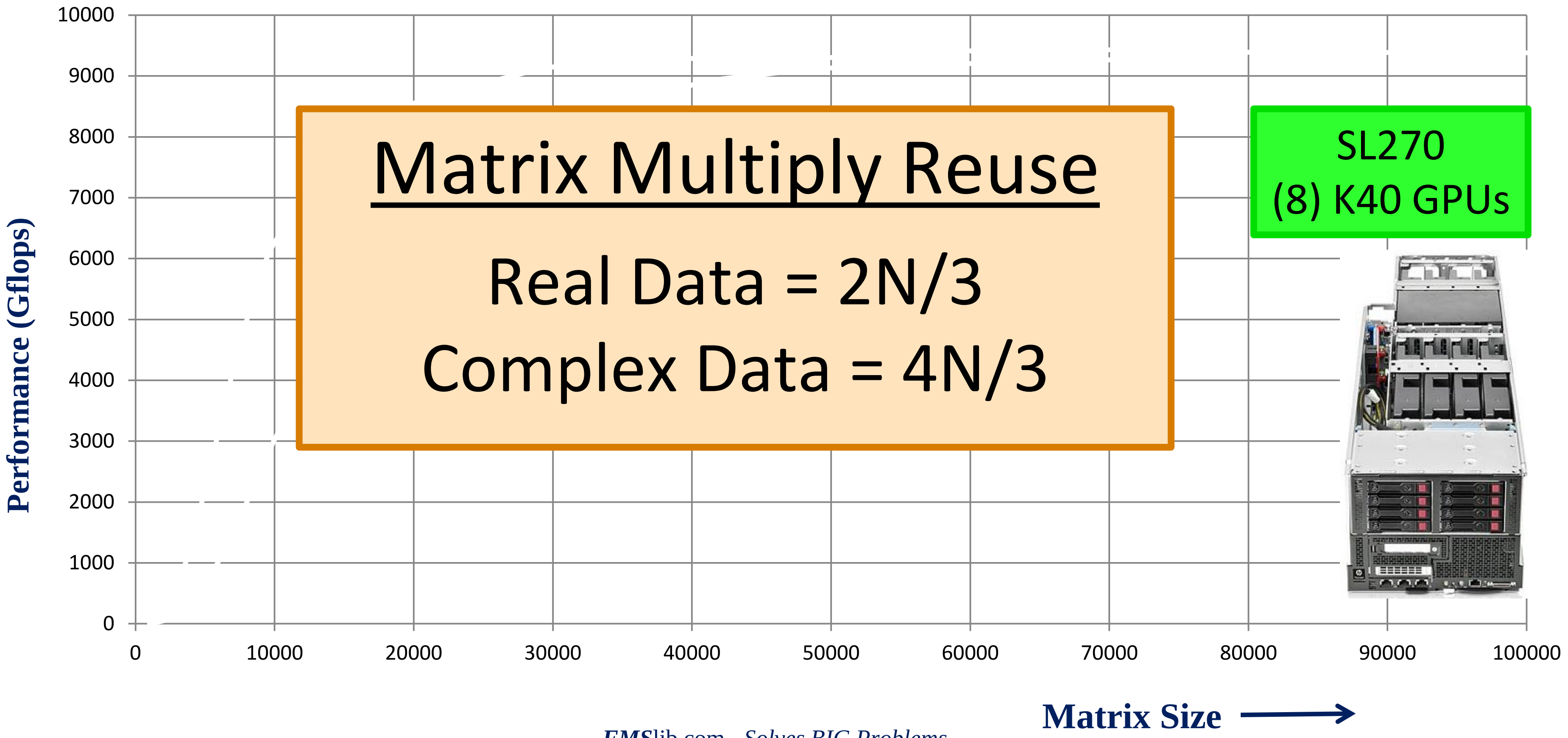
Parallel Vector Dot Product

(1/N)(4) = (4/N) Bytes/flop

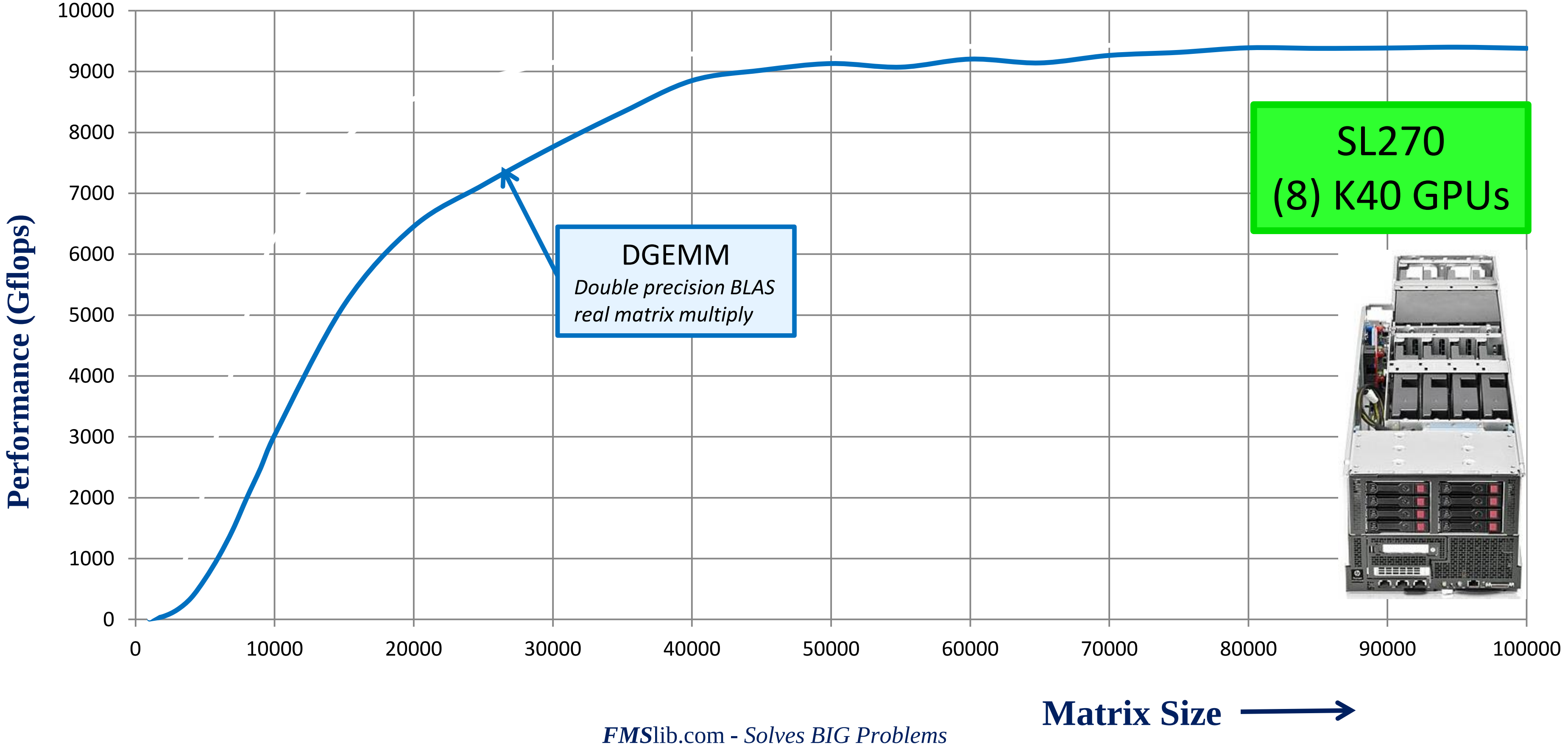
Hardware Reuse

Design	Bytes/Flop	*For 1 Tflop,* Data Rate required *(Tbytes/sec.)*	Data Rate in *(Gbytes/Sec)*	Reuse Required
General Multiplier, Adder	24	24	8	3000
Fused Multiply-Add	16	16	8	2000
Dot Product	8	8	8	1000
Vector Dot Product	4	4	8	500
N Parallel Vector Dot Product	4/N	4/N	8	500/N

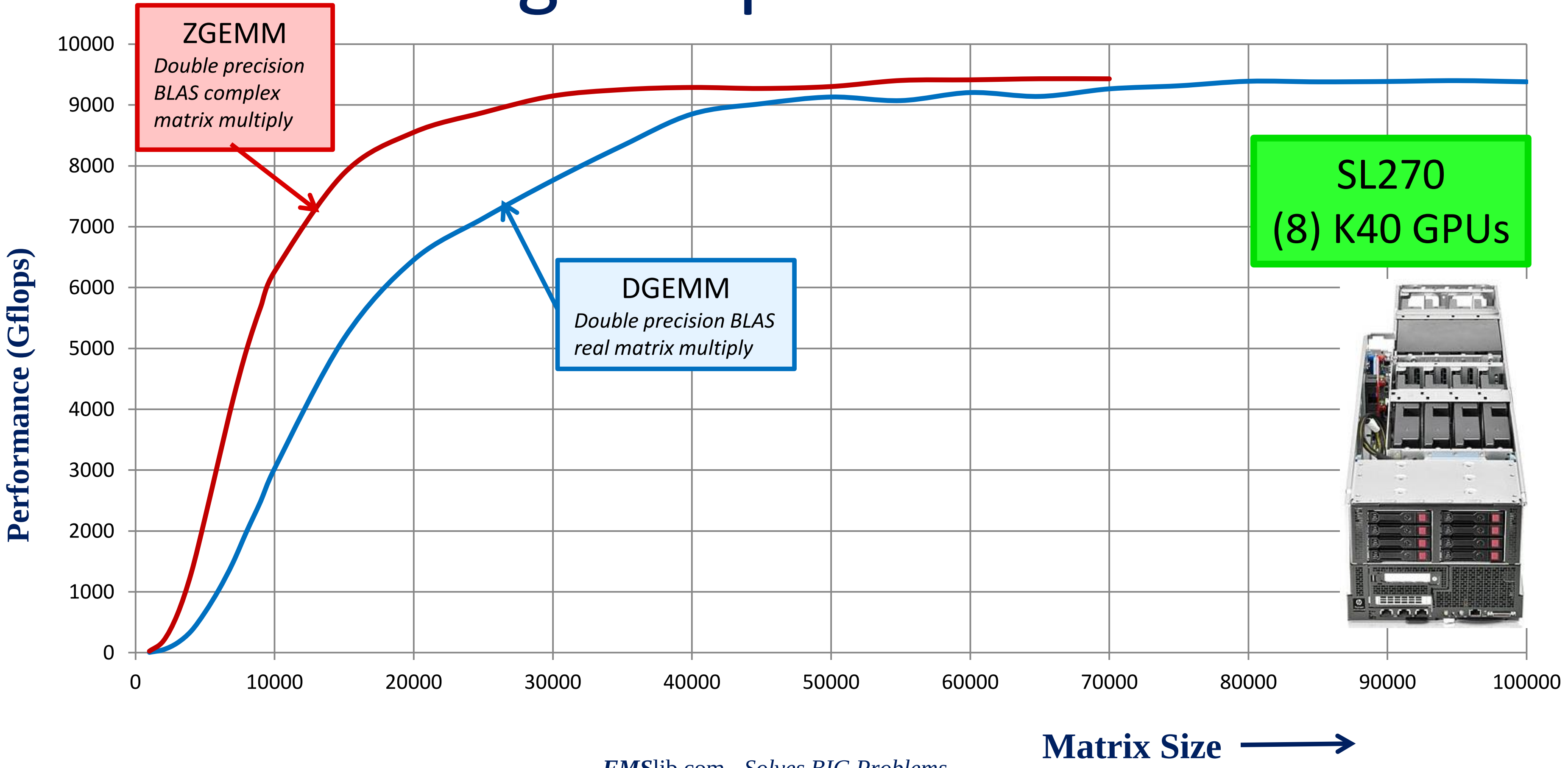
Testing “Required Reuse”



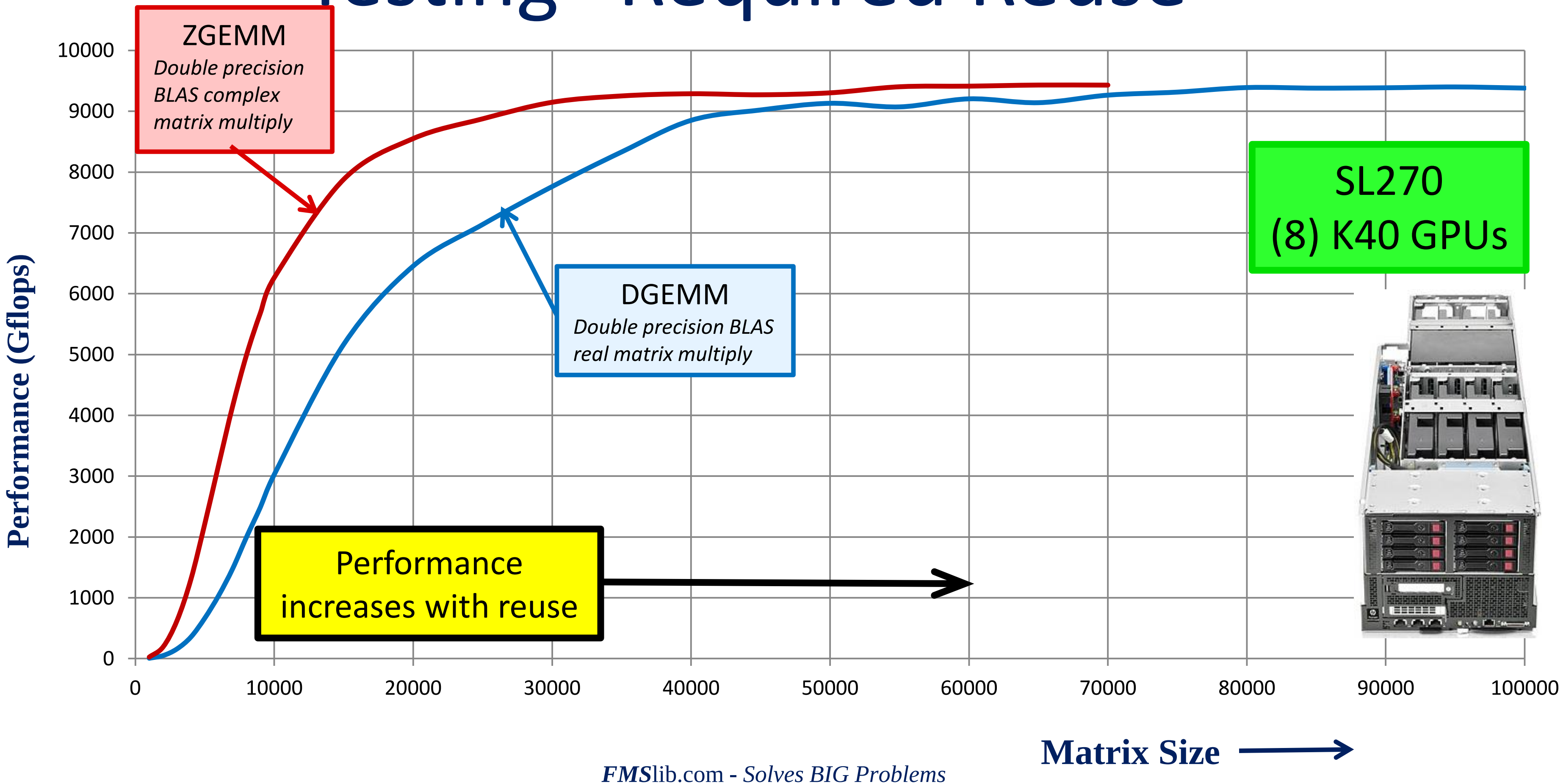
Testing "Required Reuse"



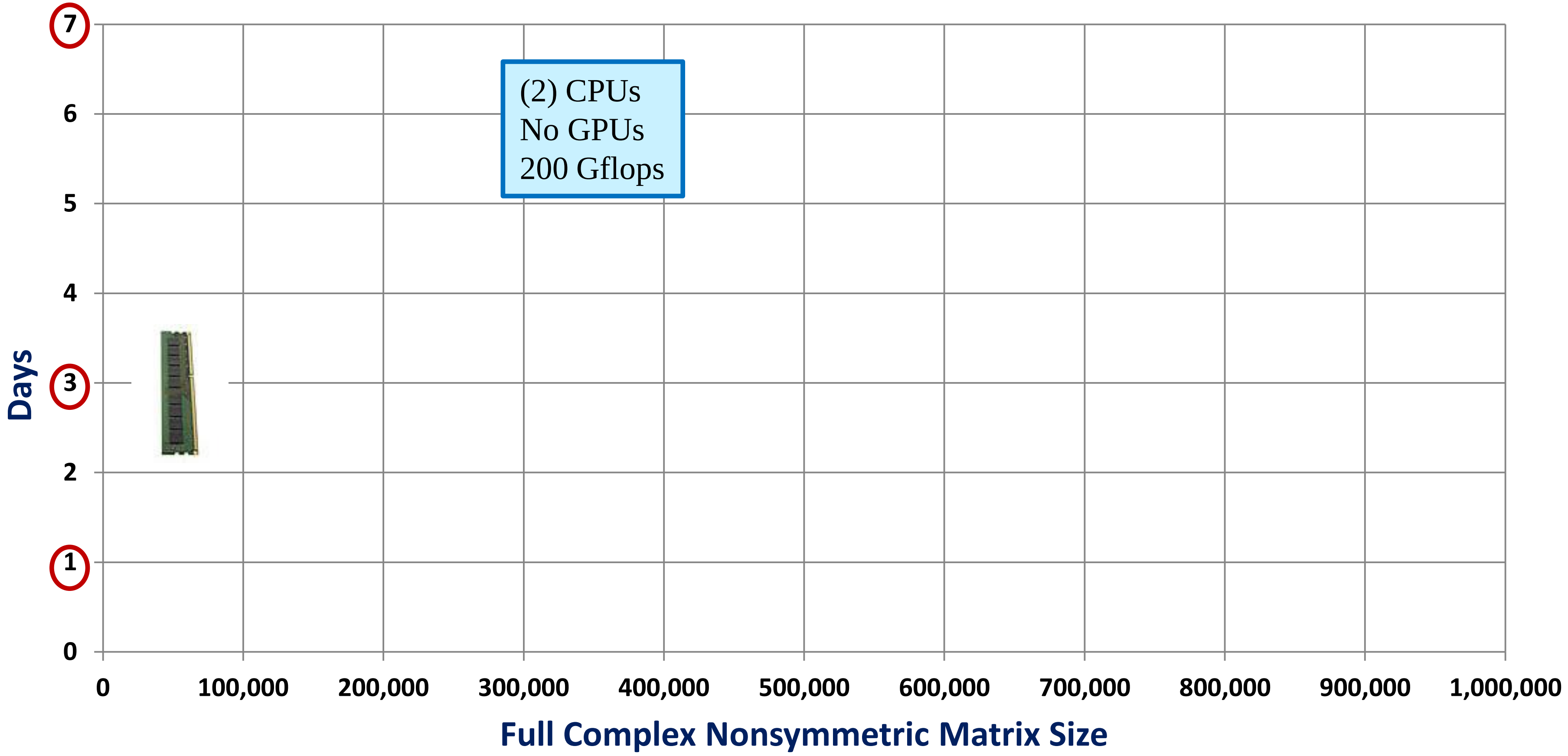
Testing "Required Reuse"



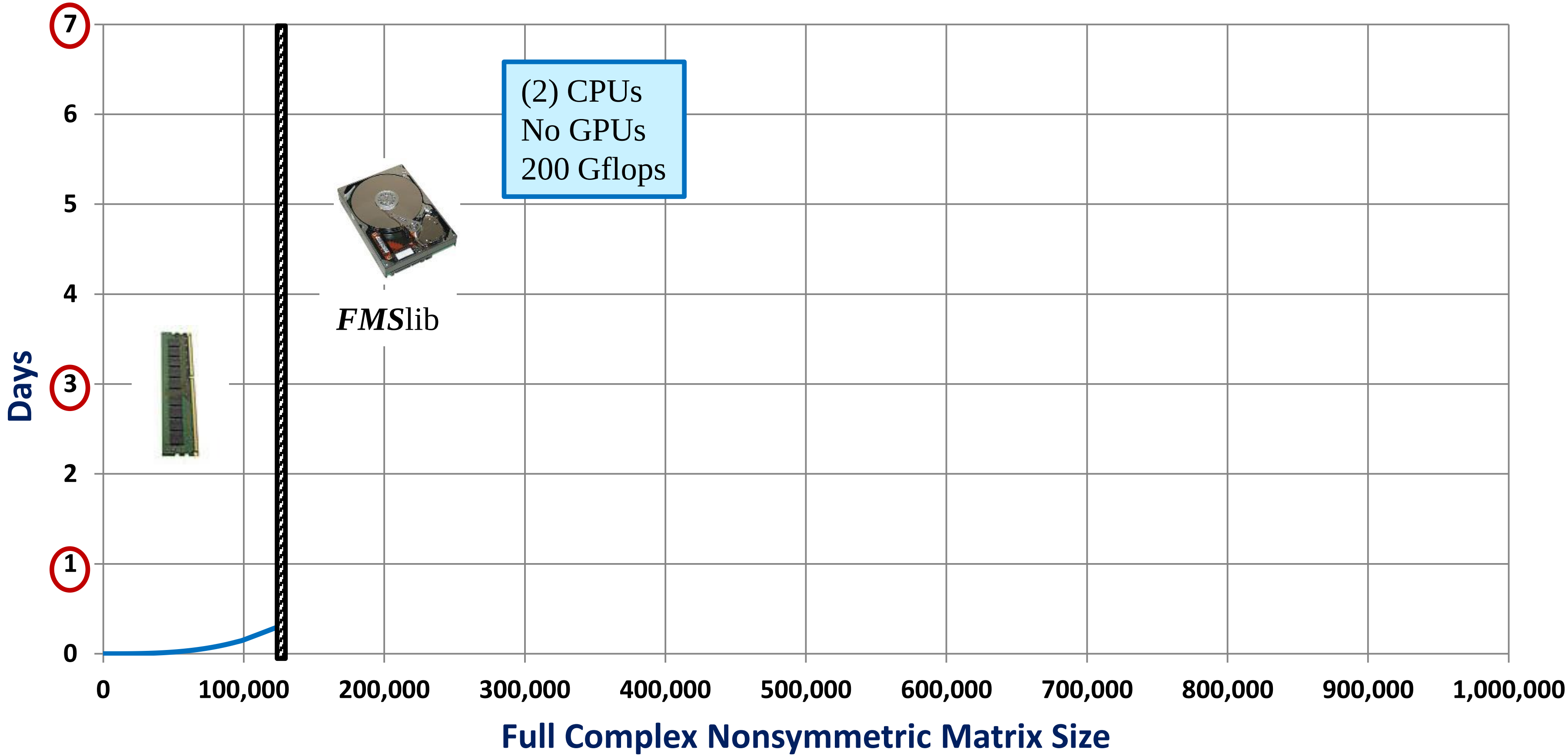
Testing "Required Reuse"



FMSlib *Performance*

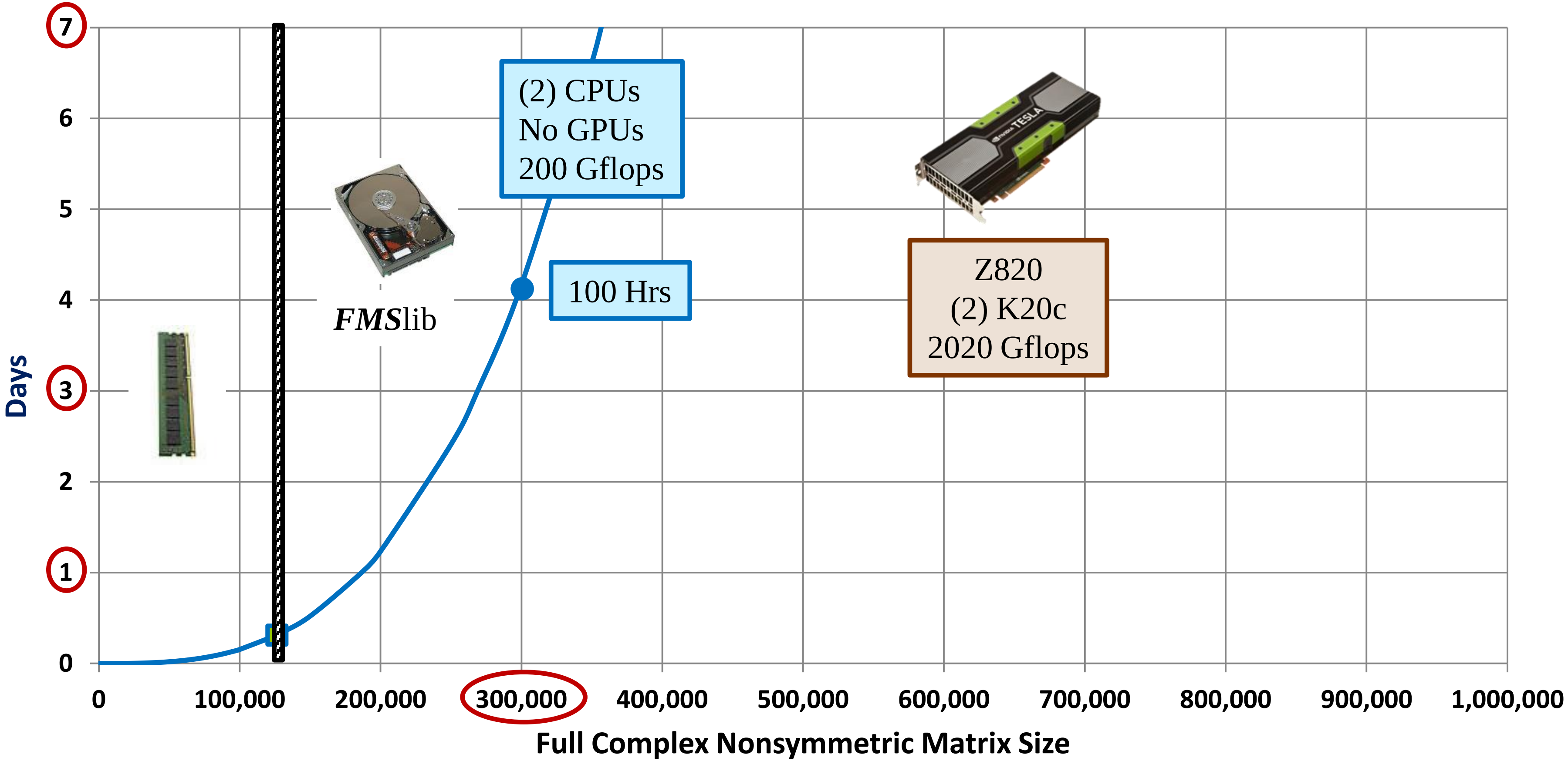


FMSlib *Performance*



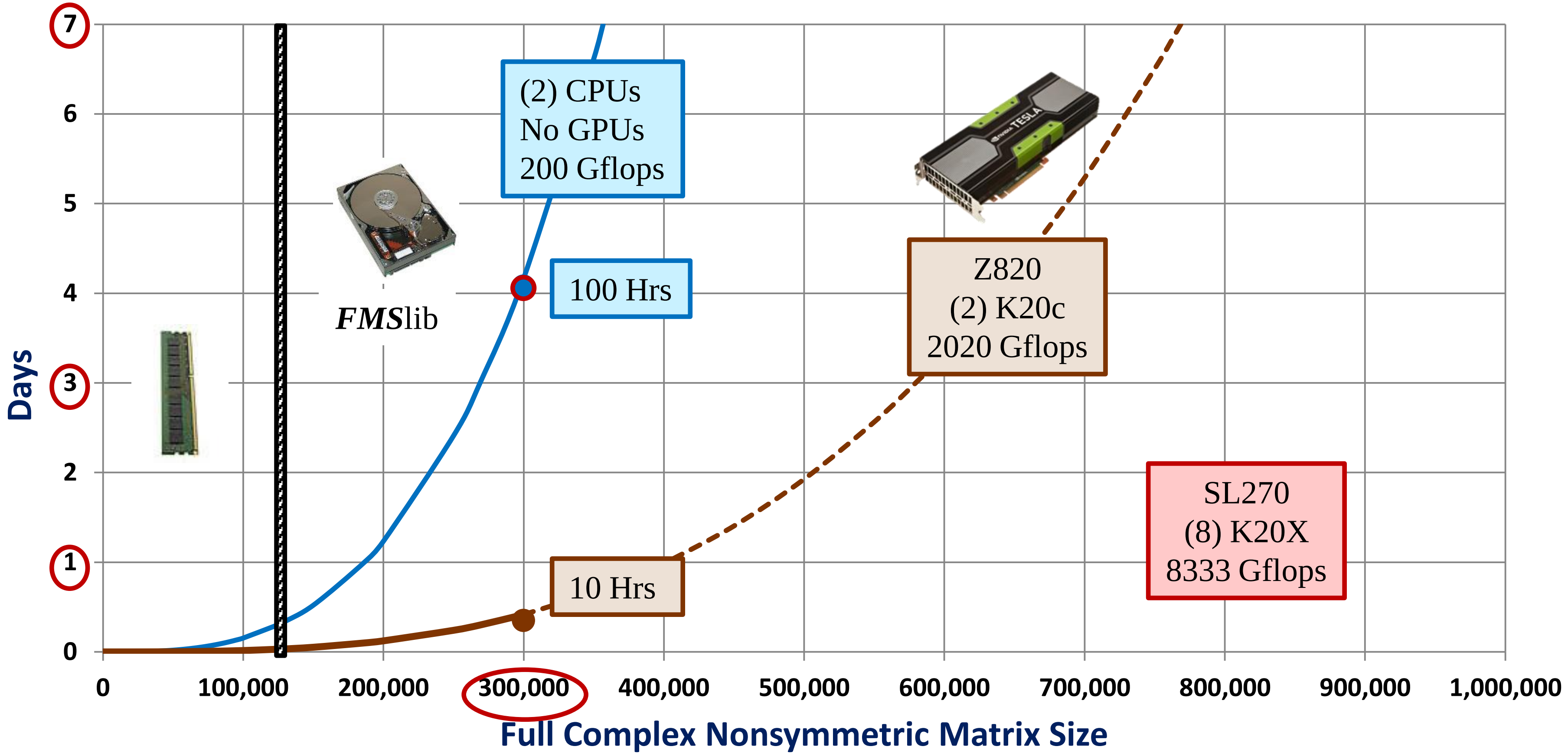
*FMS*lib.com - *Solves BIG Problems*

FMSlib *Performance*



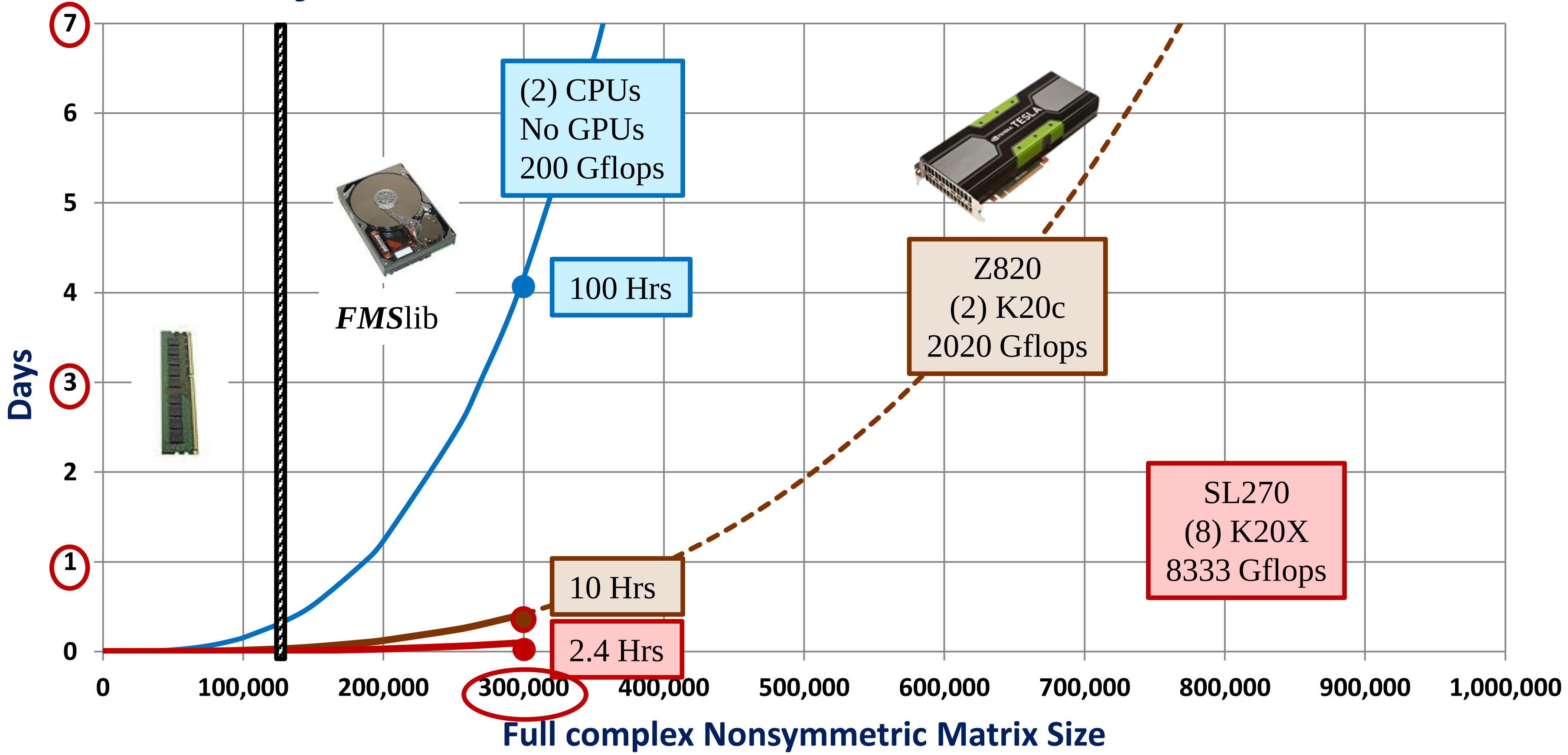
*FMS*lib.com - *Solves BIG Problems*

FMSlib *Performance*

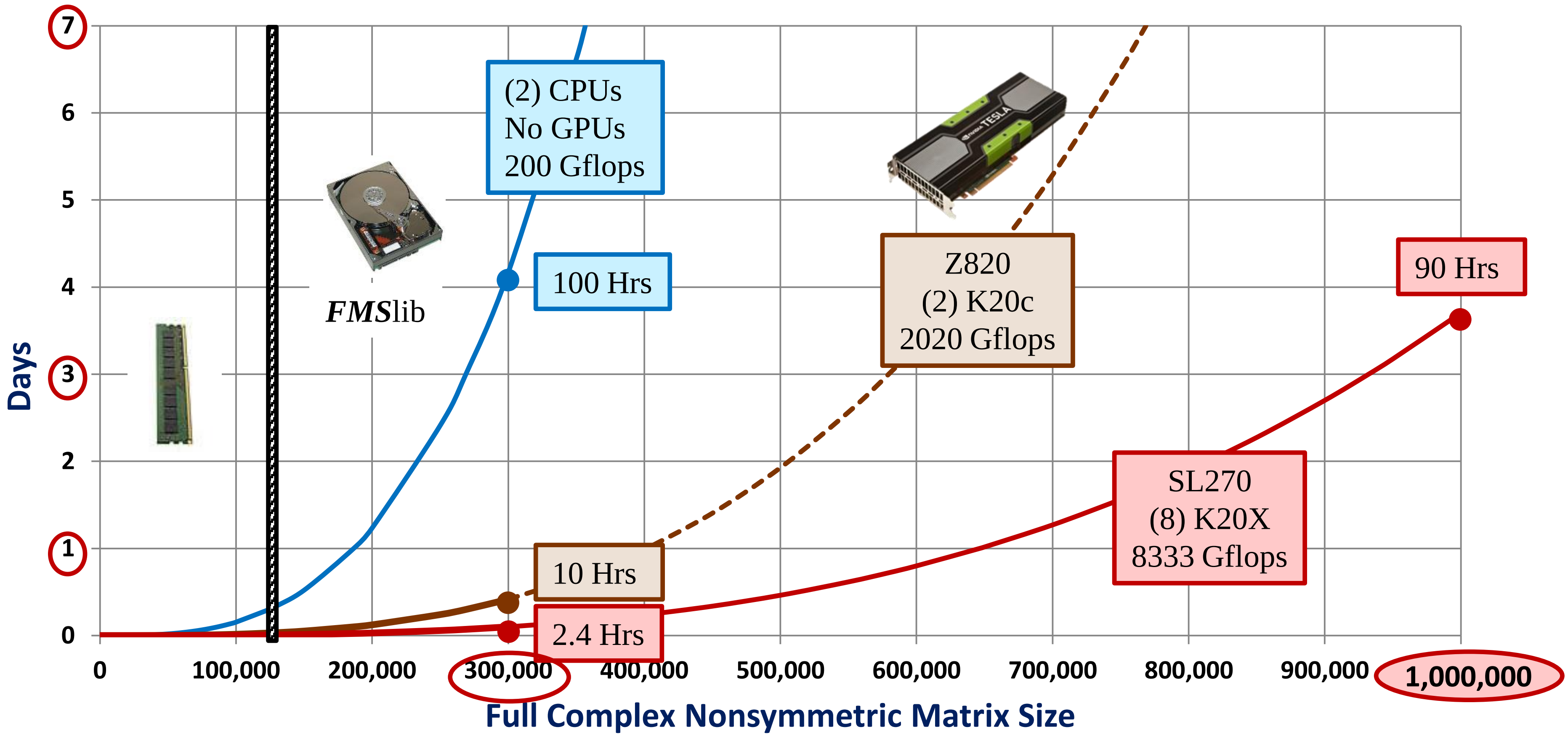


*FMS*lib.com - *Solves BIG Problems*

FMSlib *Performance*



FMSlib *Performance*



Summary

- “REUSE” is required to achieve peak performance
- Implicit solutions provide high reuse
- Reuse can be implemented at a low level in hardware
- Floating point operations are inexpensive and getting cheaper.
- Data staging determines performance and cost

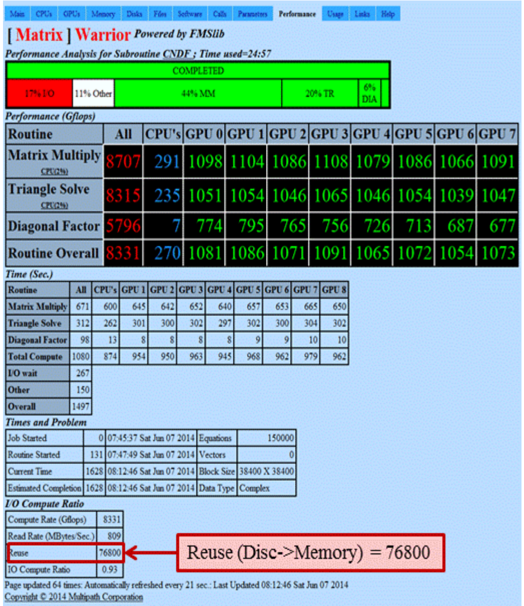
Next Steps

<u>FREE</u>

1. Download and run [Matrix]Warrior , an application which benchmarks and demonstrates FMSlib

[Matrix]Warrior
***FMS**lib*

- *Form **[A]**, **{B}** with test data*
- *Use FMSlib to solve **[A]{X}={B}***
- *Display reports as WEB pages*



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=24:57

COMPLETED

17% I/O | 11% Other | 44% MM | 20% TR | 6% DIA

Performance (Gflops)

Routine	All	CPU's	GPU 0	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7
Matrix Multiply CPU(2%)	8707	291	1098	1104	1086	1108	1079	1086	1066	1091
Triangle Solve CPU(2%)	8315	235	1051	1054	1046	1065	1046	1054	1039	1047
Diagonal Factor	5796	7	774	795	765	756	726	713	687	677
Routine Overall	8331	270	1081	1086	1071	1091	1065	1072	1054	1073

Time (Sec.)

Routine	All	CPU's	GPU 1	GPU 2	GPU 3	GPU 4	GPU 5	GPU 6	GPU 7	GPU 8
Matrix Multiply	671	600	645	642	652	640	657	653	665	650
Triangle Solve	312	262	301	300	302	297	302	300	304	302
Diagonal Factor	98	13	8	8	8	8	9	9	10	10
Total Compute	1080	874	954	950	963	945	968	962	979	962
I/O wait	267									
Other	150									
Overall	1497									

Times and Problem

Job Started	0	07:45:37 Sat Jun 07 2014	Equations	150000
Routine Started	131	07:47:49 Sat Jun 07 2014	Vectors	0
Current Time	1628	08:12:46 Sat Jun 07 2014	Block Size	38400 X 38400
Estimated Completion	1628	08:12:46 Sat Jun 07 2014	Data Type	Complex

I/O Compute Ratio

Compute Rate (Gflops)	8331
Read Rate (MBytes/Sec.)	809
Reuse	76800
IO Compute Ratio	0.93

Reuse (Disc->Memory) = 76800

Page updated 64 times: Automatically refreshed every 21 sec.: Last Updated 08:12:46 Sat Jun 07 2014

Copyright © 2014 Multipath Corporation

2. Install FMSlib in your application
3. Run FMSlib with data in memory
4. Test FMSlib with data on disc

<u>When you are ready:</u>

1. Purchase a system with GPUs (Z820, SL270)
2. License FMSlib

More Information

fmslib.com

FREE Download

[Matrix]Warrior

NVIDIA GTC Presentations

[Matrix]Warrior

FMSlib