



Impact of Architecture and Technology for Extreme Scale on Software and Algorithm Design

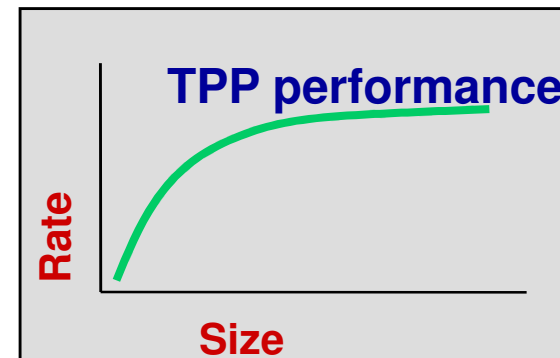
Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

H. Meuer, H. Simon, E. Strohmaier, & JD

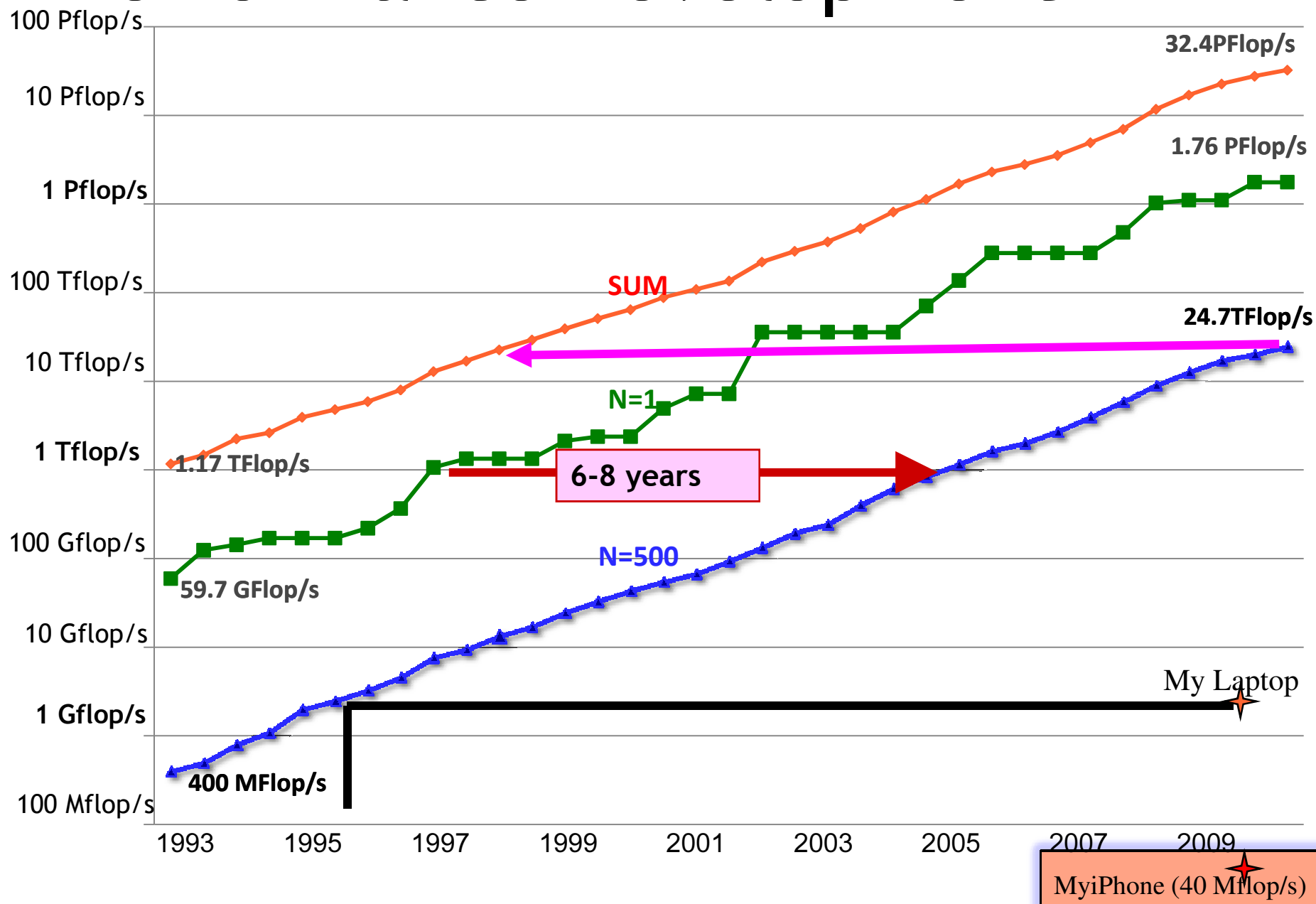
- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP

$Ax=b$, *dense problem*



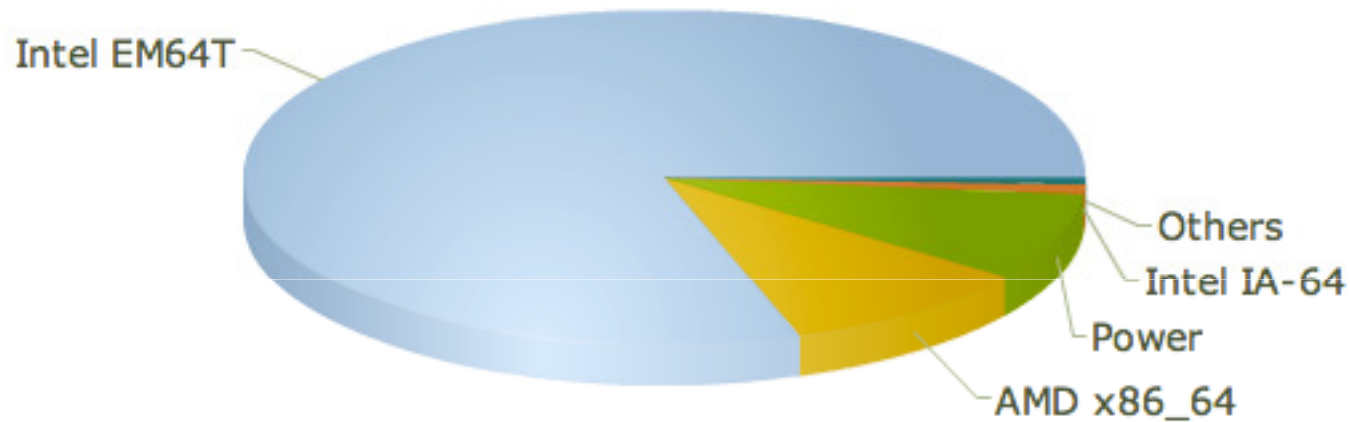
- Updated twice a year
SC'xy in the States in November
Meeting in Germany in June
- All data available from www.top500.org

Performance Development





Processors Used in the Top500 Systems



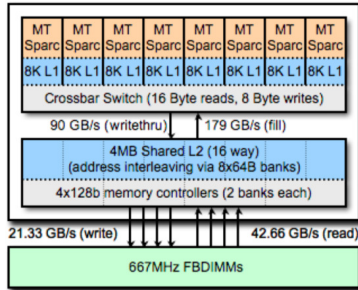
Intel 81%
AMD 10%
IBM 8%



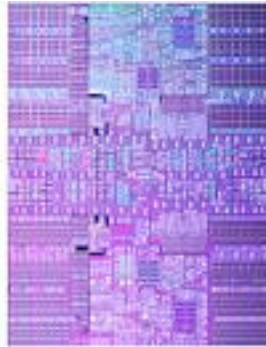
Today's Multicores

99% of Top500 Systems Are Based on Multicore

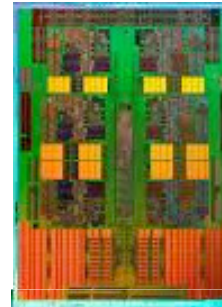
Of the Top500, 499 are multicore.



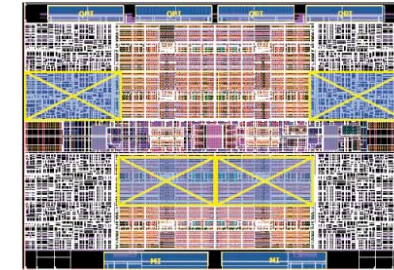
Sun Niagara2 (8 cores)



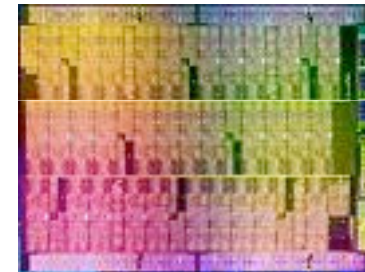
IBM Power 7 (8 cores)



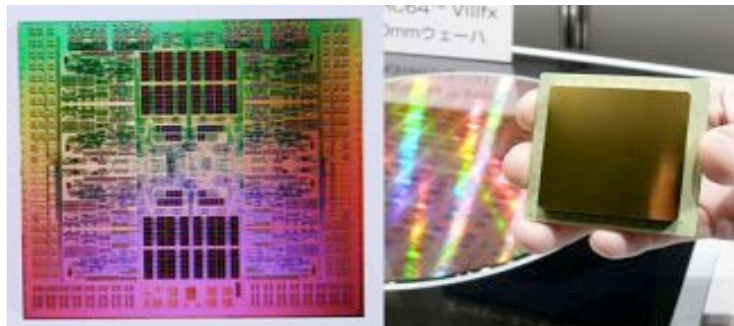
AMD MagnyCours (12 cores)



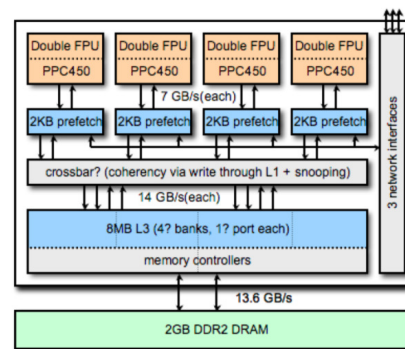
Intel Xeon(8 cores)



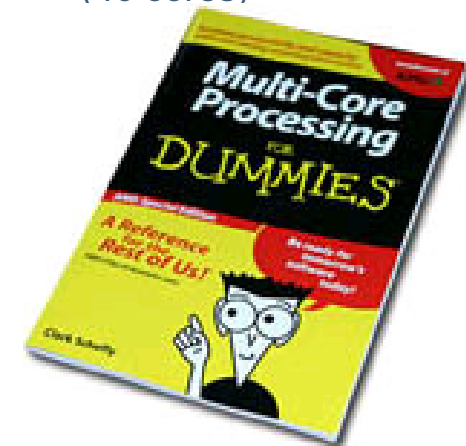
Intel Knight's Corner (40 cores)



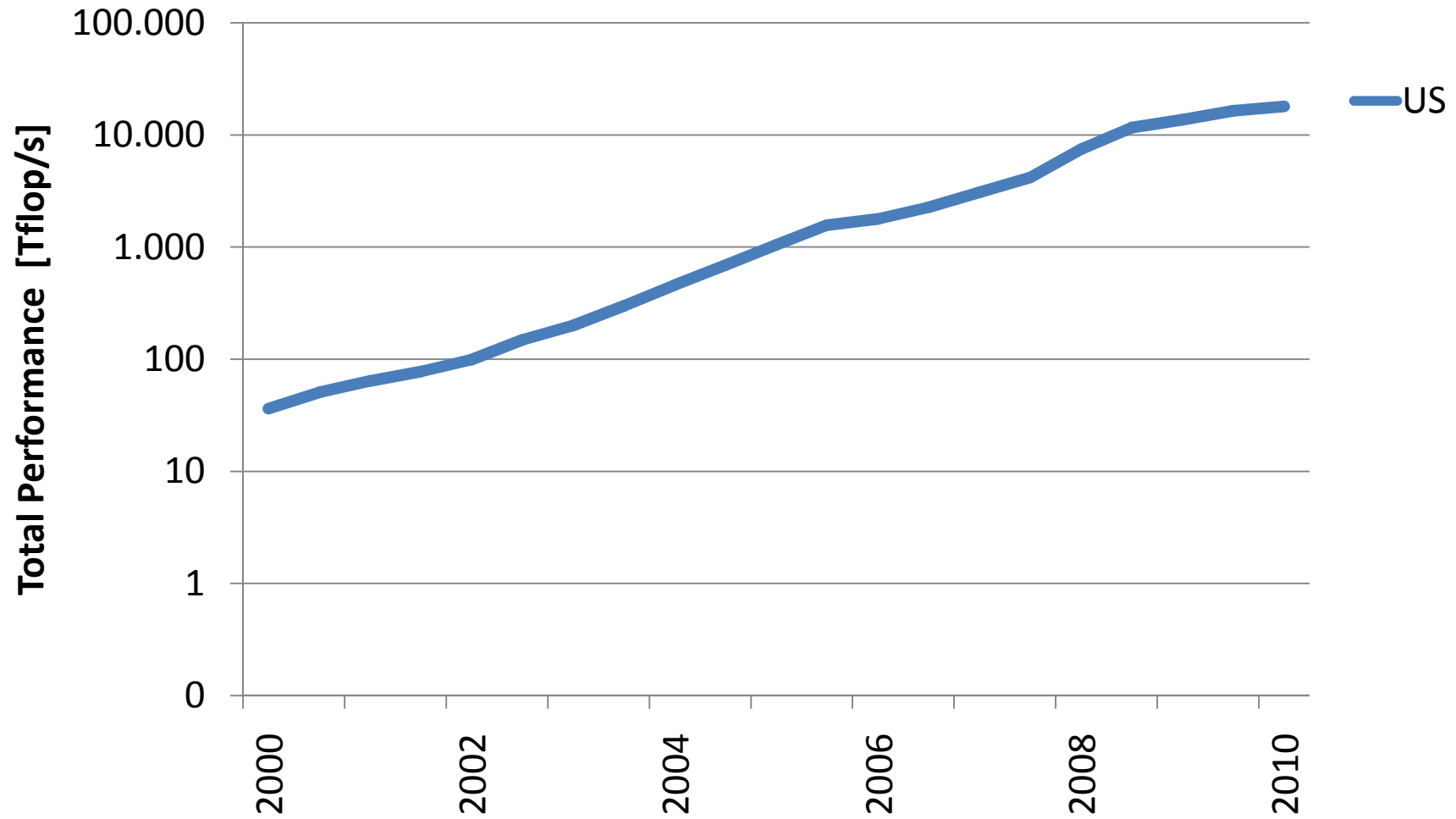
Fujitsu Venus (8 cores)



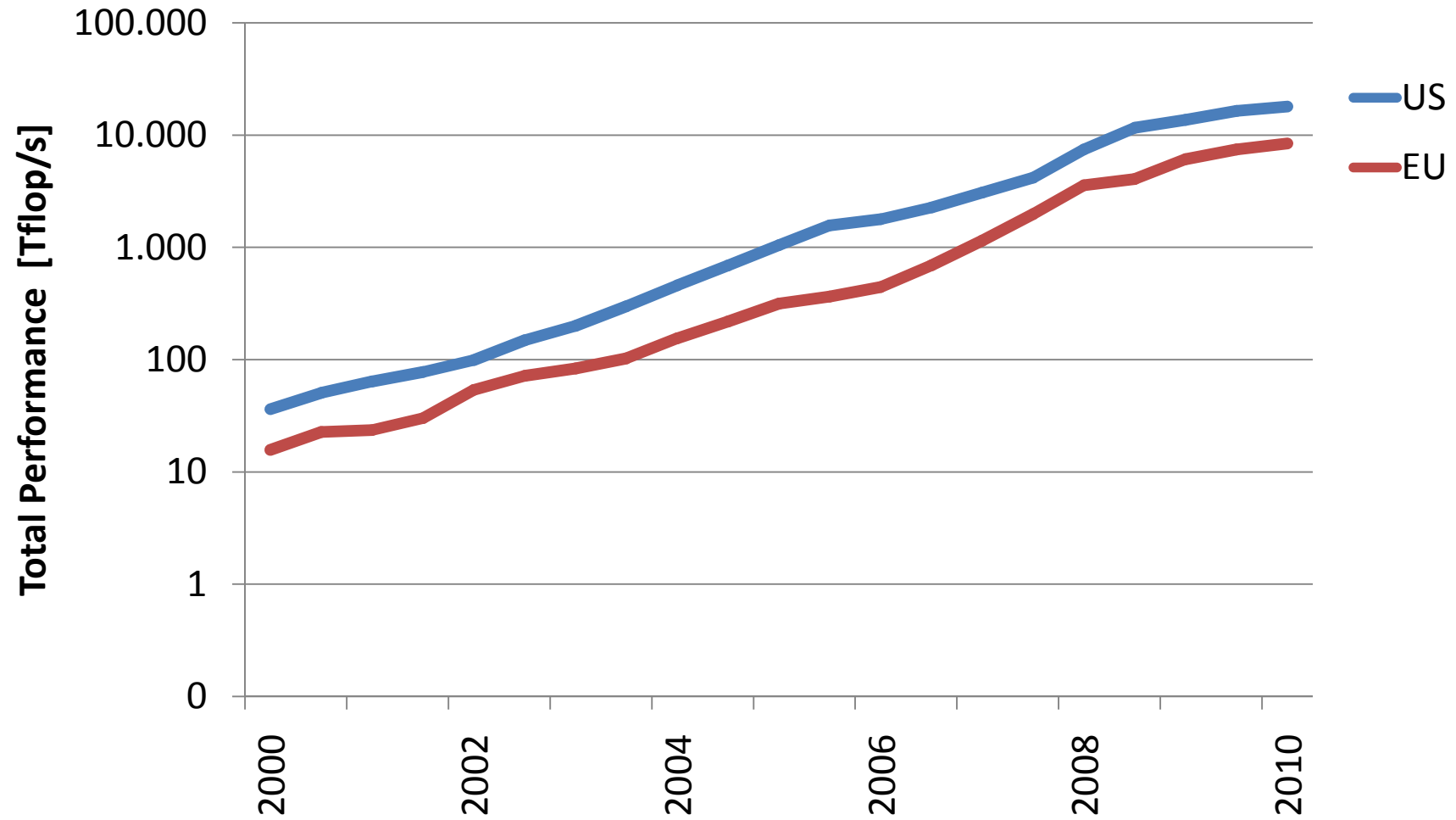
IBM BG/P (4 cores)



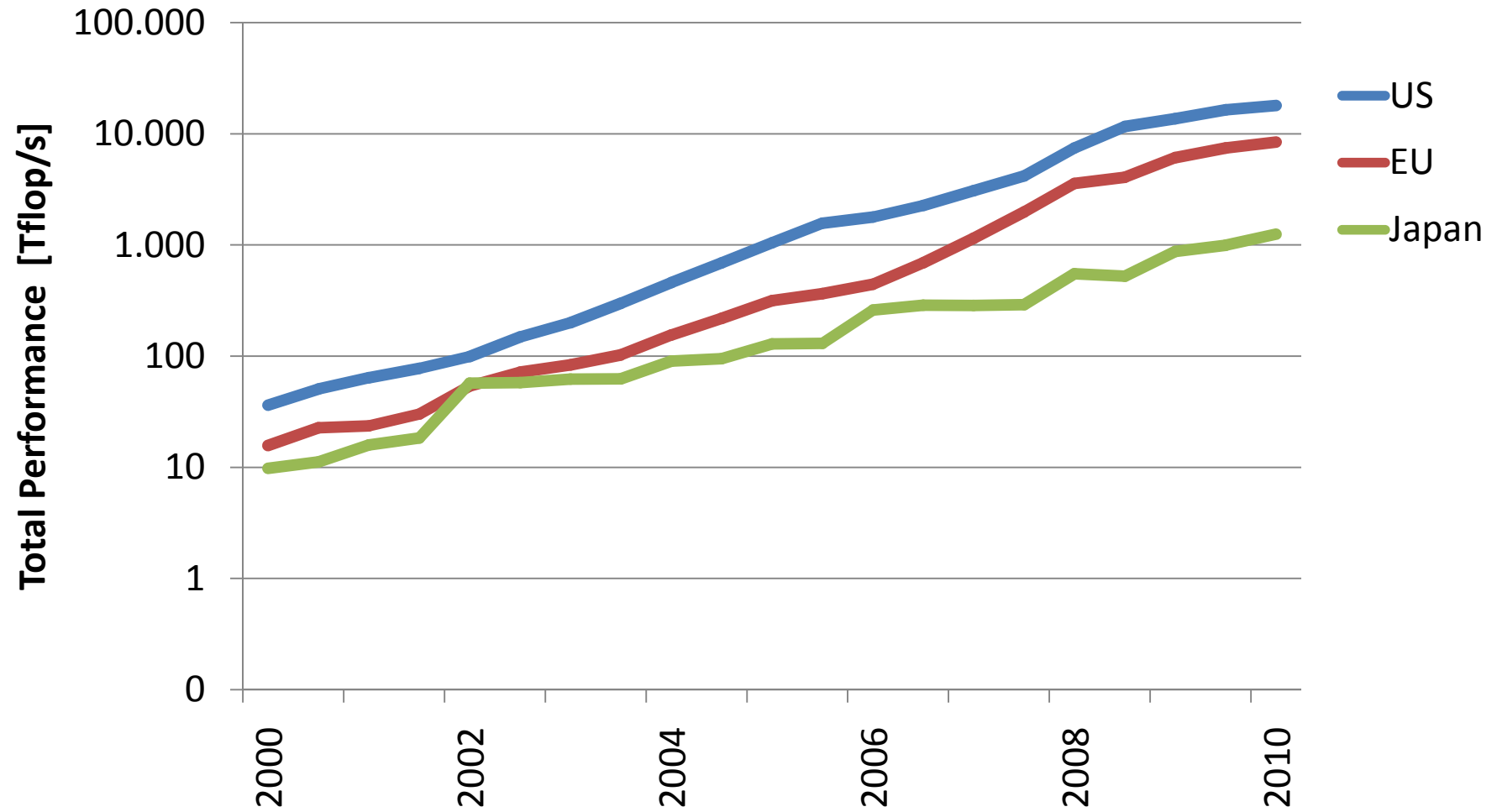
Performance of Countries



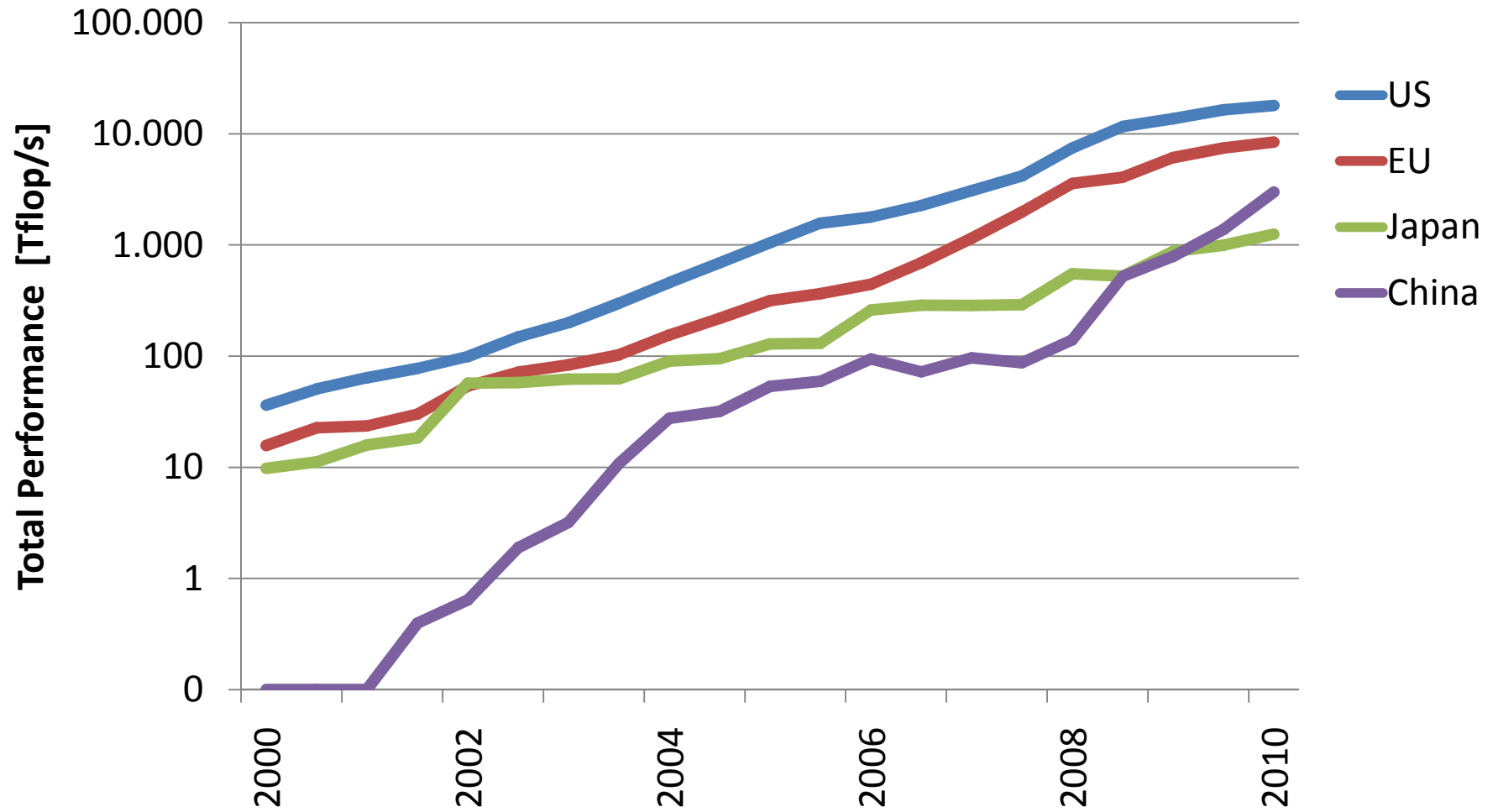
Performance of Countries



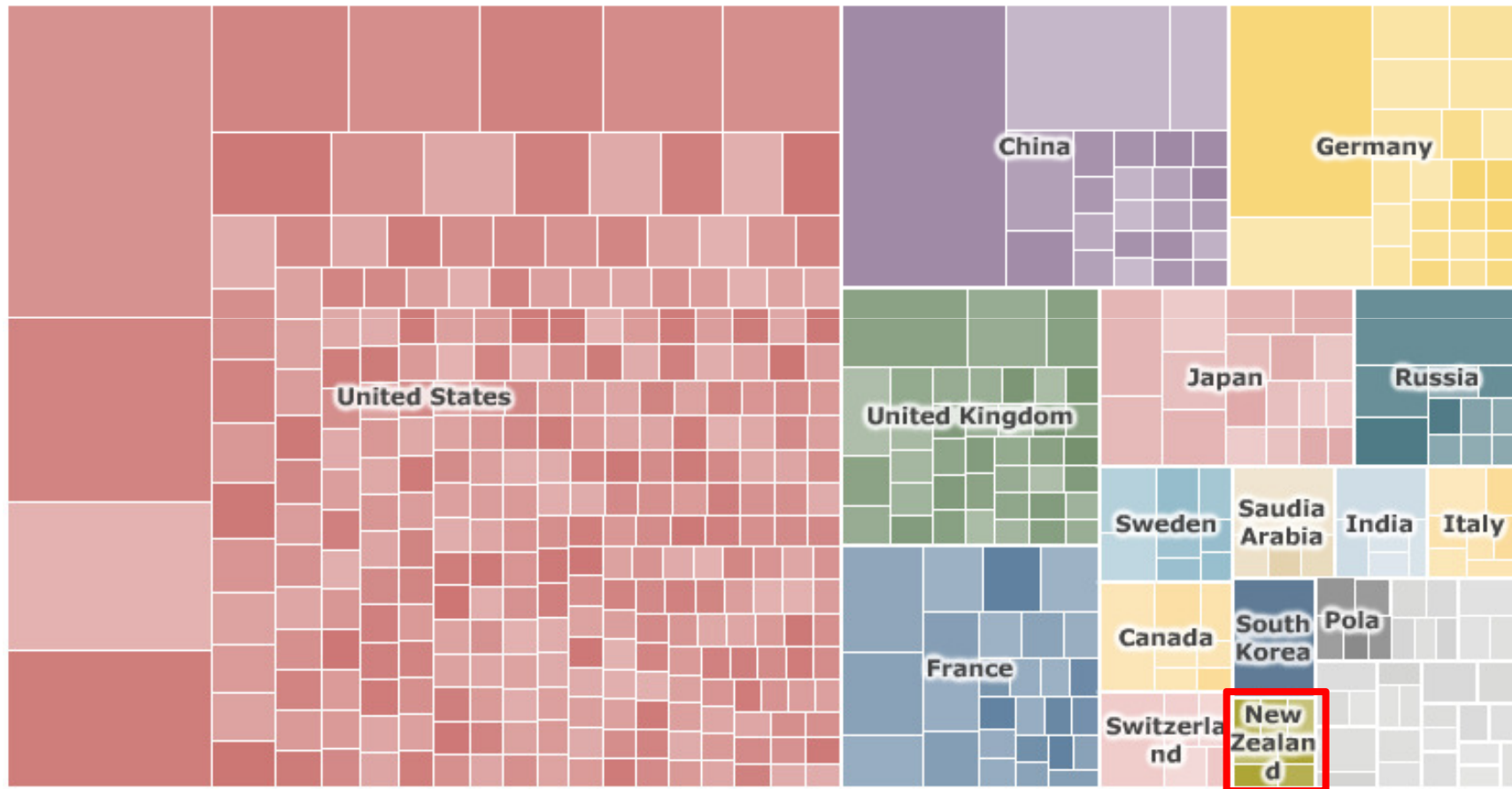
Performance of Countries



Performance of Countries



Countries / System Share



7 systems in the Italy



June 2010: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak
1	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5sixCore 2.6 GHz	USA	224,162	1.76	75
2	Nat. Supercomputer Center in Shenzhen	Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU	China	120,640	1.27	43
3	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenterQS22/LS21	USA	122,400	1.04	76
4	NSF / NICS / U of Tennessee	Kraken/ Cray Cray XT5sixCore 2.6 GHz	USA	98,928	.831	81
5	ForschungszentrumJuelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	.825	82
6	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	56,320	.544	82
7	National SC Center in Tianjin / NUDT	Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870	China	71,680	.563	46
8	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServerBlue Gene Solution	USA	212,992	.478	80
9	DOE / OS Argonne Nat Lab	Intrepid / IBM Blue Gene/P Solution	USA	163,840	.458	82
10	DOE / NNSA Sandia Nat Lab	Red Sky / Sun / SunBlade 6275	USA	42,440	.433	87



June 2010: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	MFlops /Watt
1	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5sixCore 2.6 GHz	USA	224,162	1.76	75	7.0	251
2	Nat. Supercomputer Center in Shenzhen	Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU	China	120,640	1.27	43	2.58	493
3	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenterQS22/LS21	USA	122,400	1.04	76	2.48	446
4	NSF / NICS / U of Tennessee	Kraken/ Cray Cray XT5sixCore 2.6 GHz	USA	98,928	.831	81	3.09	269
5	ForschungszentrumJuelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	.825	82	2.26	365
6	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	56,320	.544	82	3.1	175
7	National SC Center in Tianjin / NUDT	Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870	China	71,680	.563	46	1.48	380
8	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServerBlue Gene Solution	USA	212,992	.478	80	2.32	206
9	DOE / OS Argonne Nat Lab	Intrepid / IBM Blue Gene/P Solution	USA	163,840	.458	82	1.26	363
10	DOE / NNSA Sandia Nat Lab	Red Sky / Sun / SunBlade 6275	USA	42,440	.433	87	2.4	180

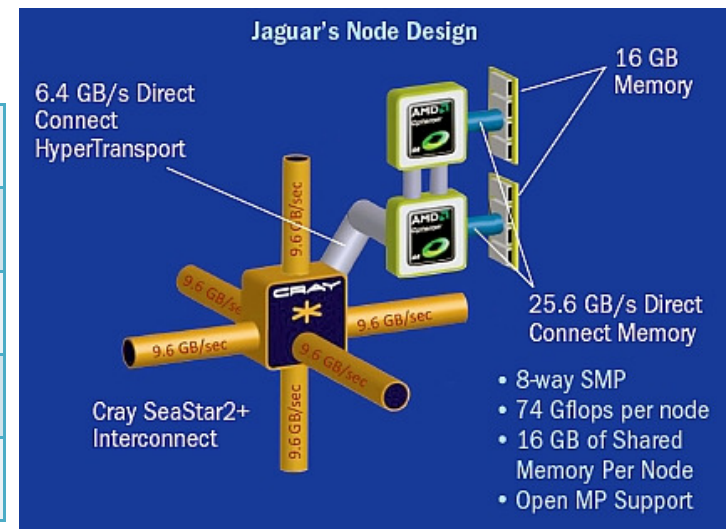


#1 ORNL's Newest System Jaguar XT5



Recently upgraded to a 2 Pflop/s system with more than 224K cores using AMD's 6 Core chip.

Peak performance	2.332 PF
System memory	300 TB
Disk space	10 PB
Disk bandwidth	240+ GB/s
Interconnect bandwidth	374 TB/s





#2 – National Supercomputer Center in Shenzhen, China – Dawning Integrator

- ◆ Nebulae
- ◆ Hybrid system, commodity + GPUs
- ◆ Theoretical peak **2.98 Pflop/s**
- ◆ Linpack Benchmark at **1.27 Pflop/s**
- ◆ 4640 nodes, each node:
 - 2 Intel 6-core Xeon5650 + Nvidia Fermi C2050 GPU (each 14 cores)
 - **120,640 cores**
 - **Infiniband connected**
 - **500 MB/s peak per link and 8 GB/s**





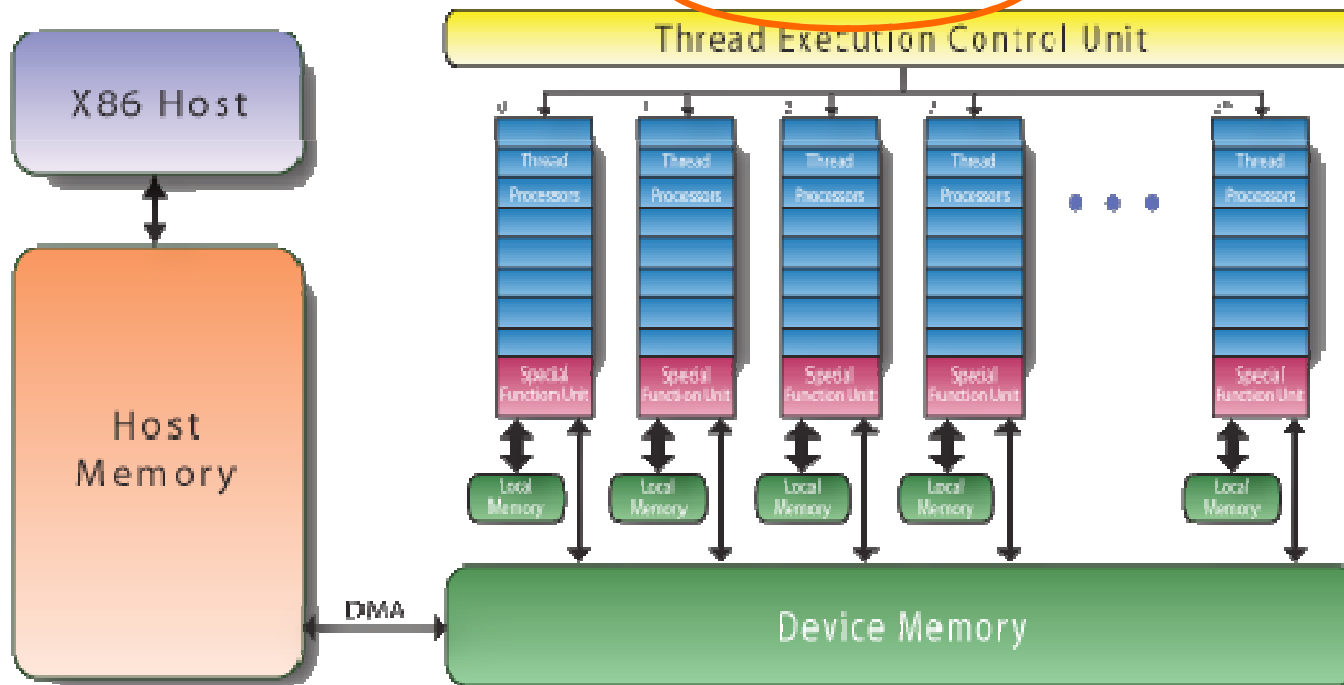
Commodity plus Accelerators

Commodity

Intel Xeon
8 cores
3 GHz
8*4 ops/cycle
96 Gflop/s (DP)

Accelerator (GPU)

Nvidia C2050 "Fermi"
448 "Cuda cores"
1.15 GHz
448 ops/cycle
515 Gflop/s (DP)



Interconnect
PCI Express

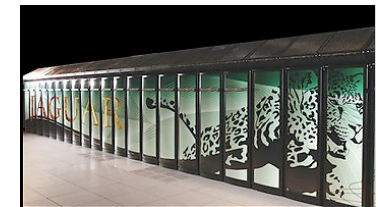
512 MB/s to 32GB/s
8 MW – 512 MW



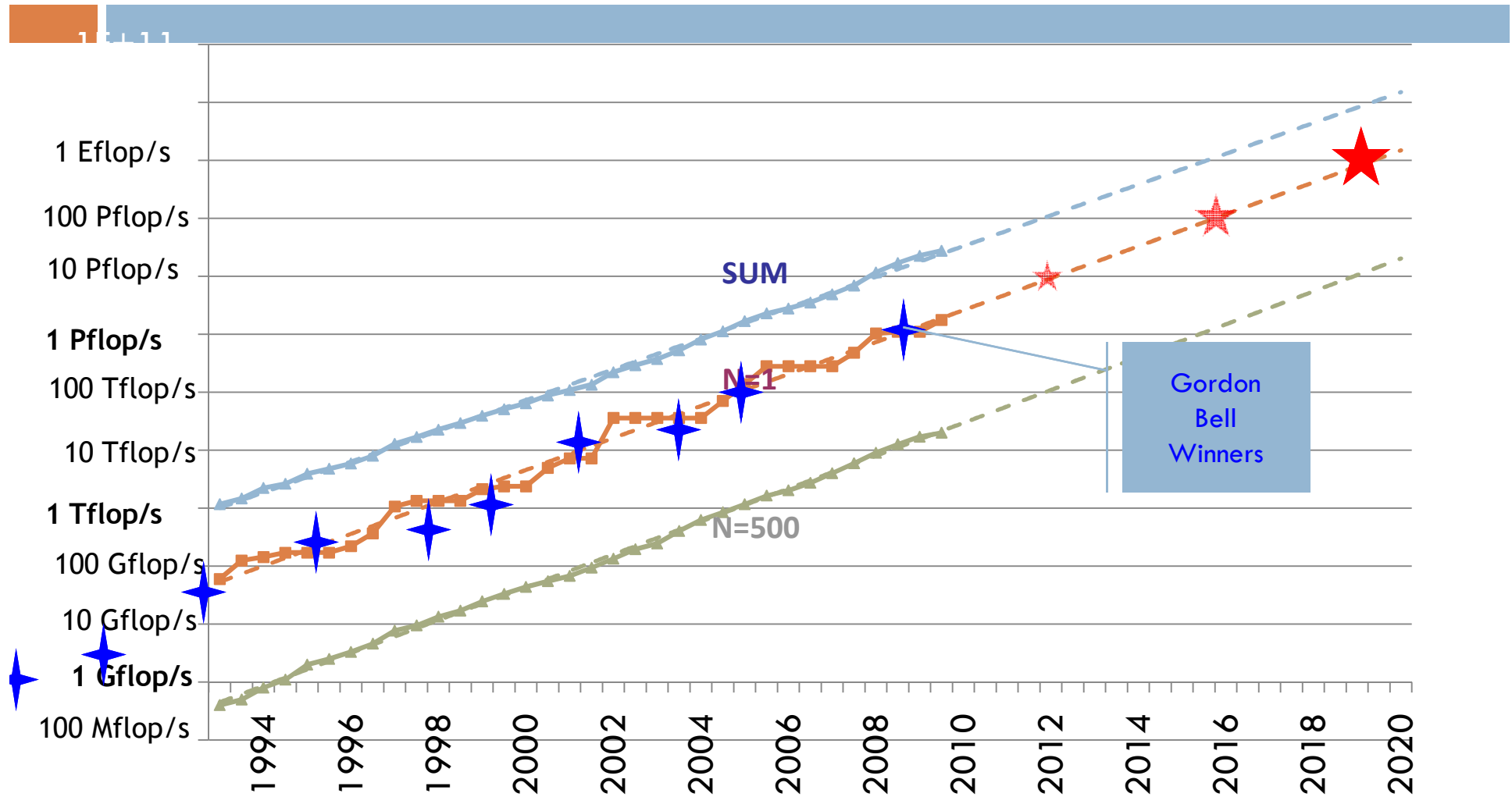
Looking at the Gordon Bell Prize

(Recognize outstanding achievement in high-performance computing applications and encourage development of parallel processing)

- 1 GFlop/s; 1988; Cray Y-MP; 8 Processors
 - ▣ Static finite element analysis
- 1 TFlop/s; 1998; Cray T3E; 1024 Processors
 - ▣ Modeling of metallic magnet atoms, using a variation of the locally self-consistent multiple scattering method.
- 1 PFlop/s; 2008; Cray XT5; 1.5×10^5 Processors
 - ▣ Superconductive materials
- 1 EFlop/s; ~ 2018 ; ?; 1×10^7 Processors (10^9 threads)



Performance Development in Top500





Potential System Architecture with a cap of \$200M and 20MW

Systems	2010
System peak	2 Pflop/s
Power	6 MW
System memory	0.3 PB
Node performance	125 GF
Node memory BW	25 GB/s
Node concurrency	12
Total Node Interconnect BW	3.5 GB/s
System size (nodes)	18,700
Total concurrency	225,000
Storage	15 PB
IO	0.2 TB
MTTI	days



Potential System Architecture with a cap of \$200M and 20MW

Systems	2010	2018
System peak	2 Pflop/s	1 Eflop/s
Power	6 MW	~20 MW
System memory	0.3 PB	32 - 64 PB [.03 Bytes/Flop]
Node performance	125 GF	1,2 or 15TF
Node memory BW	25 GB/s	2 - 4TB/s [.002 Bytes/Flop]
Node concurrency	12	O(1k) or 10k
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)
System size (nodes)	18,700	O(100,000) or O(1M)
Total concurrency	225,000	O(billion) [O(10) to O(100) for latency hiding]
Storage	15 PB	500-1000 PB (>10x system memory is min)
IO	0.2 TB	60 TB/s (how long to drain the machine)
MTTI	days	O(1 day)



Potential System Architecture with a cap of \$200M and 20MW

Systems	2010	2018	Difference Today & 2018
System peak	2 Pflop/s	1 Eflop/s	O(1000)
Power	6 MW	~20 MW	
System memory	0.3 PB	32 - 64 PB [.03 Bytes/Flop]	O(100)
Node performance	125 GF	1,2 or 15TF	O(10) - O(100)
Node memory BW	25 GB/s	2 - 4TB/s [.002 Bytes/Flop]	O(100)
Node concurrency	12	O(1k) or 10k	O(100) - O(1000)
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)	O(100)
System size (nodes)	18,700	O(100,000) or O(1M)	O(10) - O(100)
Total concurrency	225,000	O(billion) [O(10) to O(100) for latency hiding]	O(10,000)
Storage	15 PB	500-1000 PB (>10x system memory is min)	O(10) - O(100)
IO	0.2 TB	60 TB/s (how long to drain the machine)	O(100)
MTTI	days	O(1 day)	- O(10)



Exascale (10^{18} Flop/s) Systems: Two possible paths

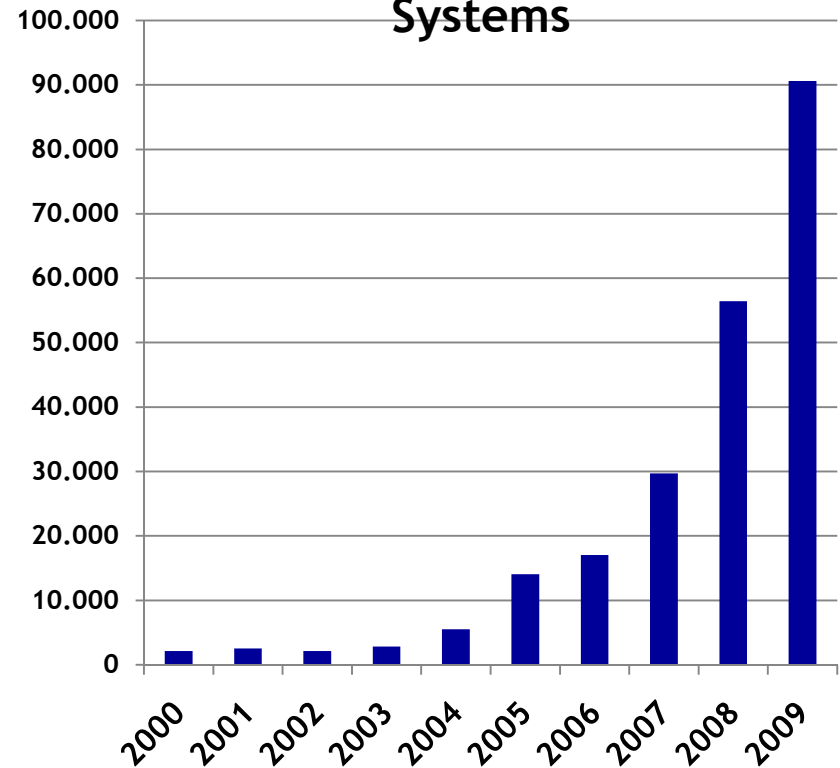
- **Light weight processors (think BG/P)**
 - ~1 GHz processor (10^9)
 - ~1 Kilo cores/socket (10^3)
 - ~1 Mega sockets/system (10^6)
- **Hybrid system (think GPU based)**
 - ~1 GHz processor (10^9)
 - ~10 Kilo FPUs/socket (10^4)
 - ~100 Kilo sockets/system (10^5)



Factors that Necessitate Redesign of Our Software

- Steepness of the ascent from terascale to petascale to exascale
- Extreme parallelism and hybrid design
 - Preparing for million/billion way parallelism
- Tightening memory/bandwidth bottleneck
 - Limits on power/clock speed implication on multicore
 - Reducing communication will become much more intense
 - Memory per core changes, byte-to-flop ratio will change
- Necessary Fault Tolerance
 - MTF will drop
 - Checkpoint/restart has limitations

Average Number of Cores Per Supercomputer for Top20 Systems



Software infrastructure does not exist today



Moore's Law reinterpreted

- Number of cores per chip will double every two years
- Clock speed will not increase (possibly decrease) because of Power
- Need to deal with systems with millions of concurrent threads
- Need to deal with inter-chip parallelism as well as intra-chip parallelism

Major Changes to Software

- **Must rethink the design of our software**
 - **Another disruptive technology**
 - Similar to what happened with cluster computing and message passing
 - **Rethink and rewrite the applications, algorithms, and software**
- **Numerical libraries for example will change**
 - **For example, both LAPACK and ScaLAPACK will undergo major changes to accommodate this**

Future Computer Systems

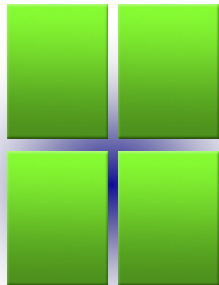
- Most likely be a hybrid design
- Think standard multicore chips and accelerator (GPUs)
- Today accelerators are attached
- Next generation more integrated
- Intel's Larrabee? Now called "Knights Corner" and "Knights Ferry" to come.
 - 48 x86 cores
- AMD's Fusion in 2011 - 2013
 - Multicore with embedded graphics ATI
- Nvidia's plans?



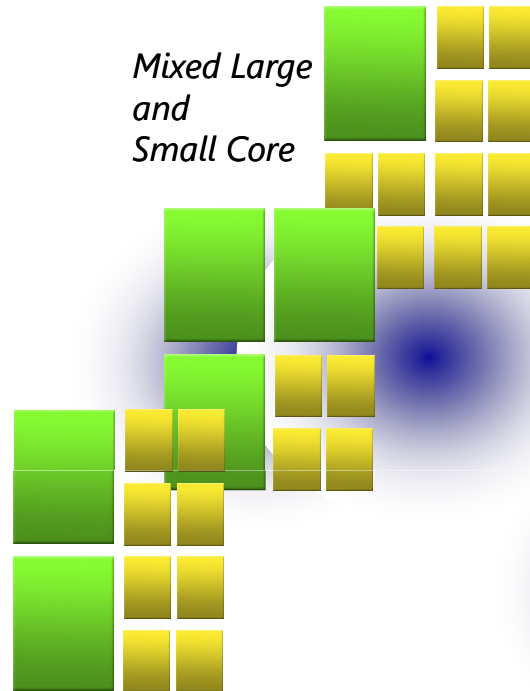


What's Next?

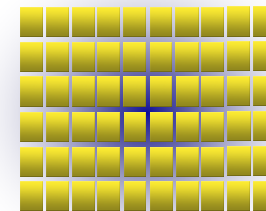
All Large Core



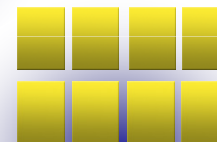
Mixed Large and Small Core



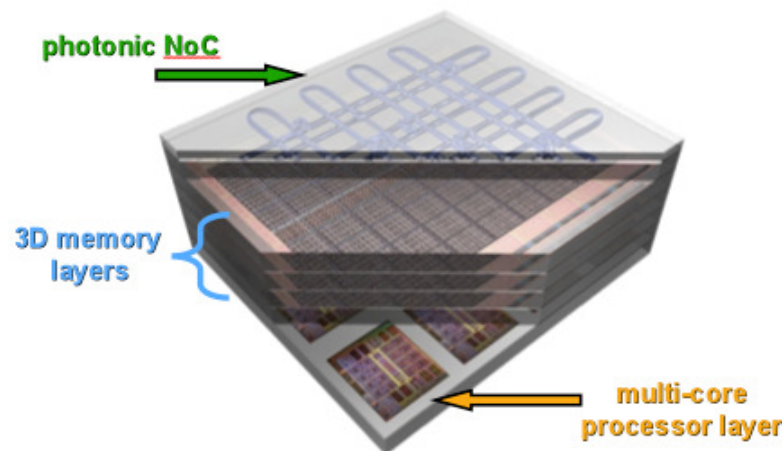
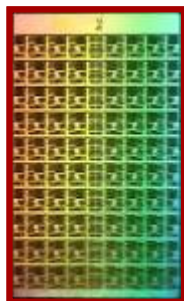
Many Small Cores



All Small Core



Many Floating-Point Cores



Different Classes of Chips

- Home
- Games / Graphics
- Business
- Scientific

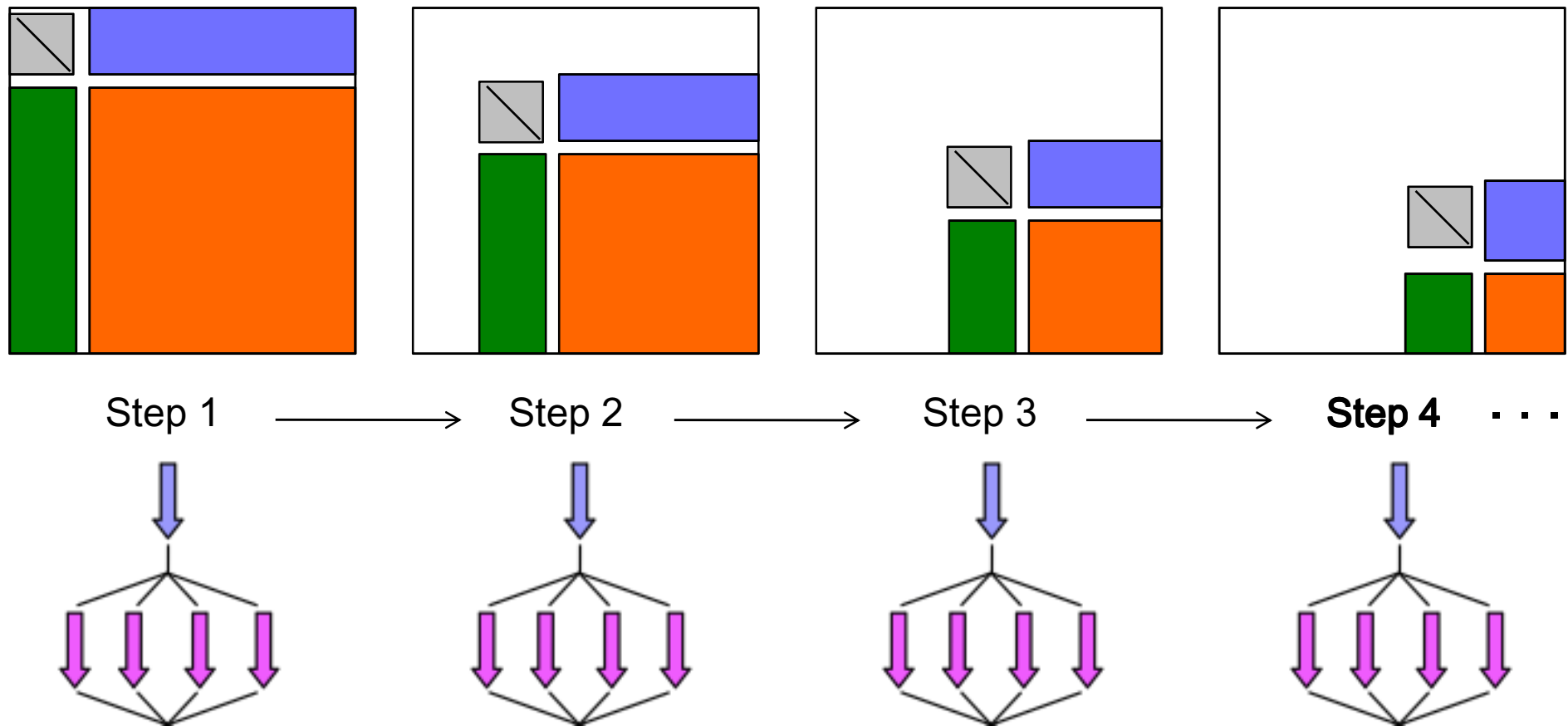




Five Important Software Features to Consider When Computing at Scale

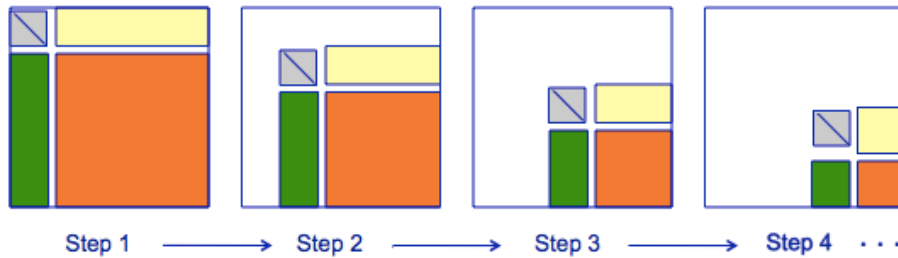
- 1. Effective Use of Many-Core and Hybrid architectures**
 - Break fork-join parallelism
 - Dynamic Data Driven Execution
 - Block Data Layout
- 2. Exploiting Mixed Precision in the Algorithms**
 - Single Precision is 2X faster than Double Precision
 - With GP-GPUs 10x
 - Power saving issues
- 3. Self Adapting / Auto Tuning of Software**
 - Too hard to do by hand
- 4. Fault Tolerant Algorithms**
 - With 1,000,000's of cores things will fail
- 5. Communication Reducing Algorithms**
 - For dense computations from $O(n \log p)$ to $O(\log p)$ communications
 - Asynchronous iterations
 - GMRES k-step compute ($x, Ax, A^2x, \dots A^kx$)

LAPACK LU/LL^T/QR

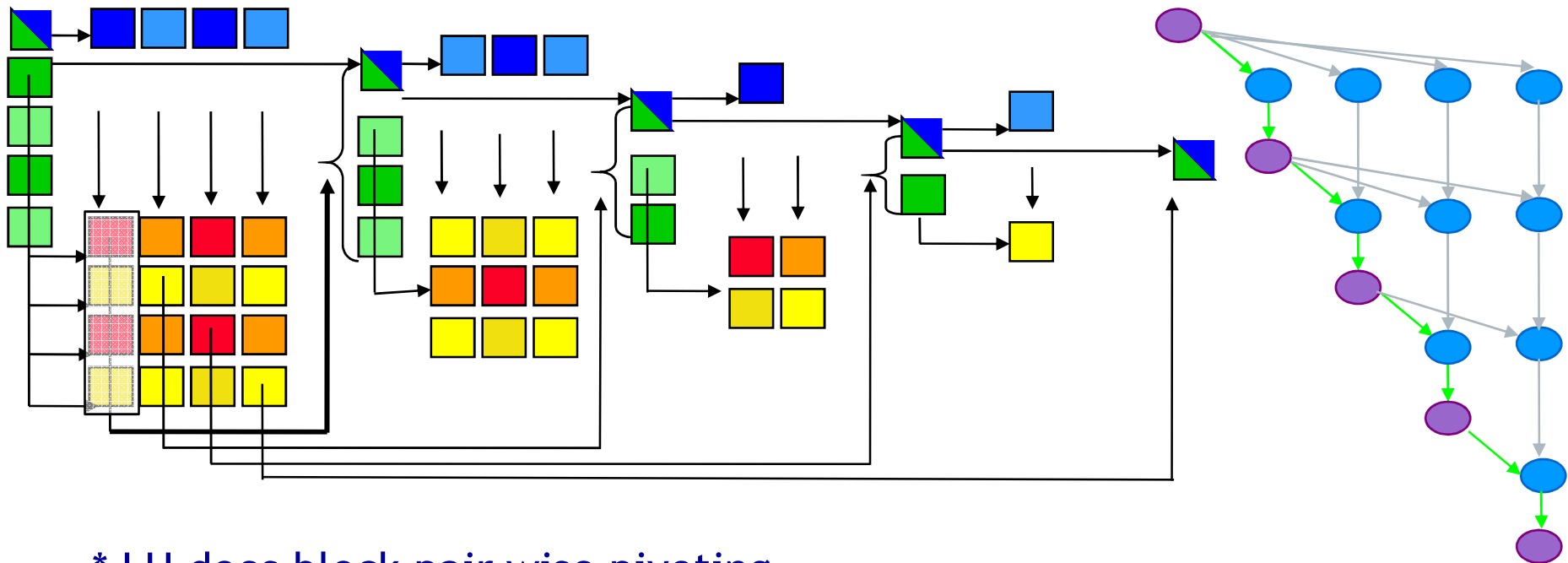


- Fork-join, bulk synchronous processing

Parallel Tasks in LU/LL^T/QR



- Break into smaller tasks and remove dependencies



* LU does block pair wise pivoting



PLASMA: Parallel Linear Algebra s/w for Multicore Architectures

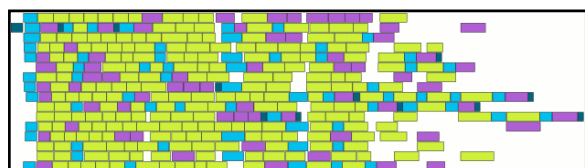
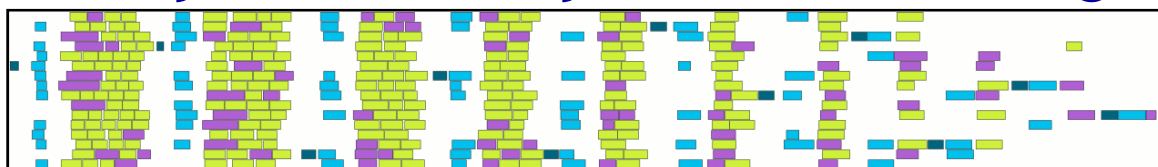
• Objectives

- High utilization of each core
- Scaling to large number of cores
- Shared or distributed memory

• Methodology

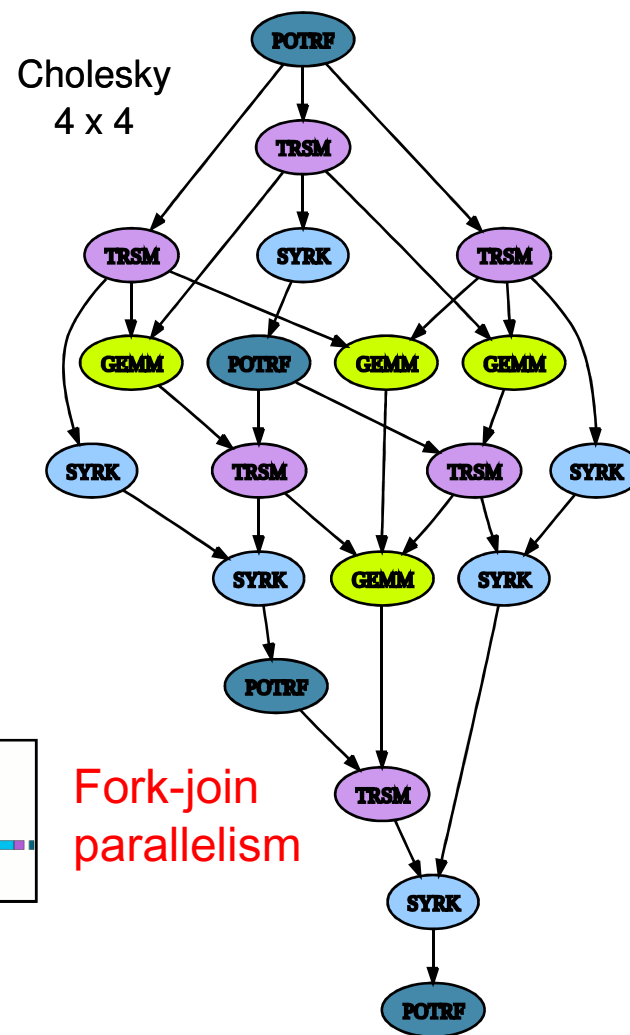
- Dynamic DAG scheduling
- Explicit parallelism
- Implicit communication
- Fine granularity / block data layout

• Arbitrary DAG with dynamic scheduling



DAG scheduled parallelism

Time





Communication Avoiding Algorithms

- Goal: Algorithms that communicate as little as possible
- Jim Demmel and company have been working on algorithms that obtain a provable minimum communication.
- Direct methods (BLAS, LU, QR, SVD, other decompositions)
 - Communication lower bounds for *all* these problems
 - Algorithms that attain them (*all* dense linear algebra, some sparse)
 - Mostly not in LAPACK or ScaLAPACK (yet)
- Iterative methods - Krylov subspace methods for $Ax=b$, $Ax=\lambda x$
 - Communication lower bounds, and algorithms that attain them (depending on sparsity structure)
 - Not in any libraries (yet)
- For QR Factorization they can show:

	Lower bound
# flops	$\Theta(mn^2)$
# words	$\Theta\left(\frac{mn^2}{\sqrt{W}}\right)$
# messages	$\Theta\left(\frac{mn^2}{W^{3/2}}\right)$



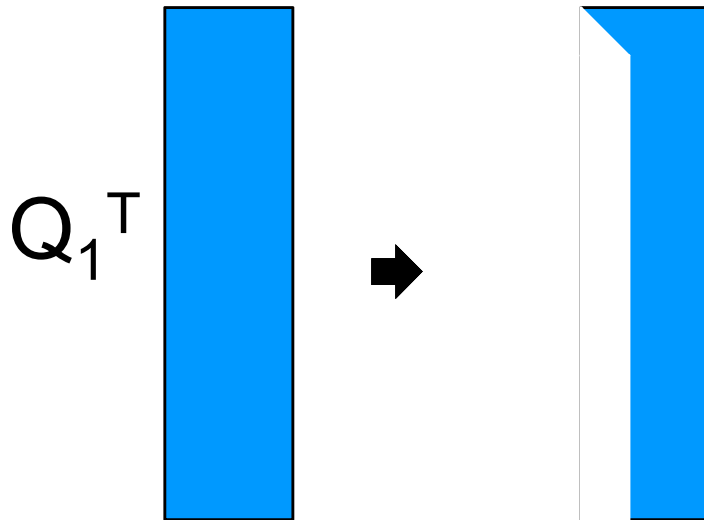
Standard QR Block Reduction

- We have a $m \times n$ matrix A we want to reduce to upper triangular form.



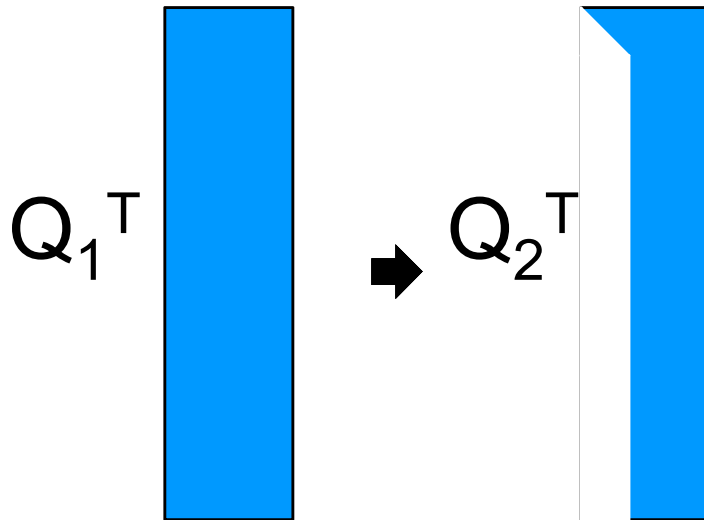
Standard QR Block Reduction

- We have a $m \times n$ matrix A we want to reduce to upper triangular form.



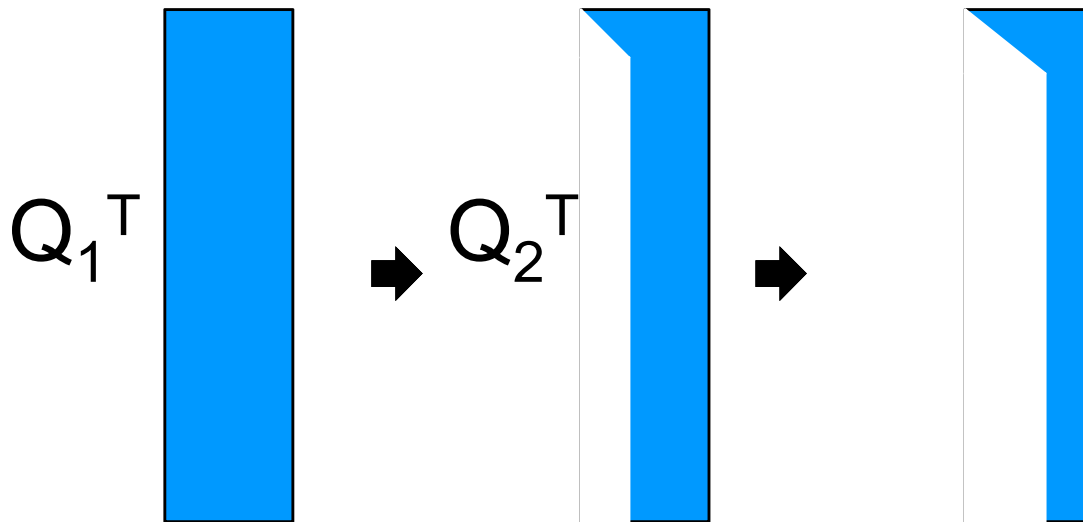
Standard QR Block Reduction

- We have a $m \times n$ matrix A we want to reduce to upper triangular form.



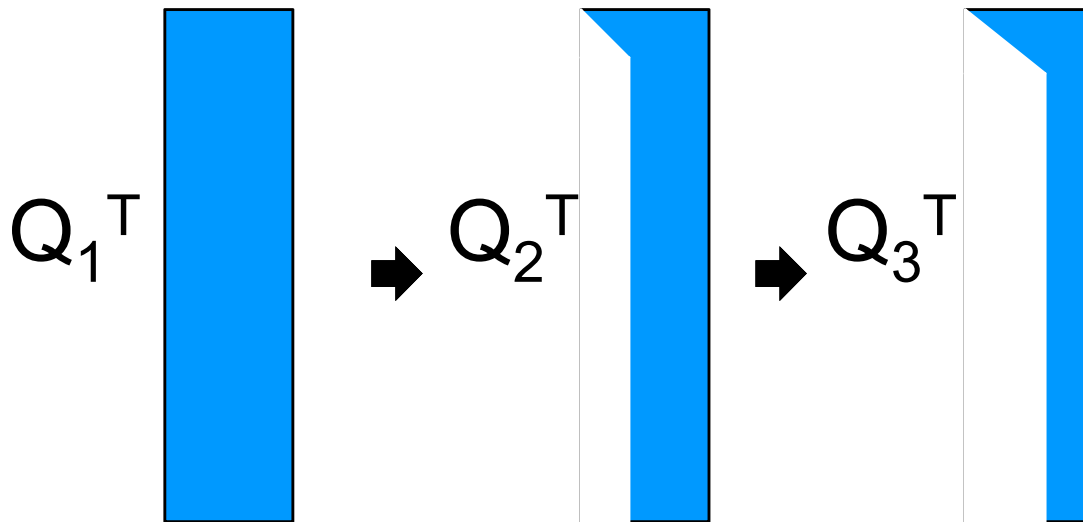
Standard QR Block Reduction

- We have a $m \times n$ matrix A we want to reduce to upper triangular form.



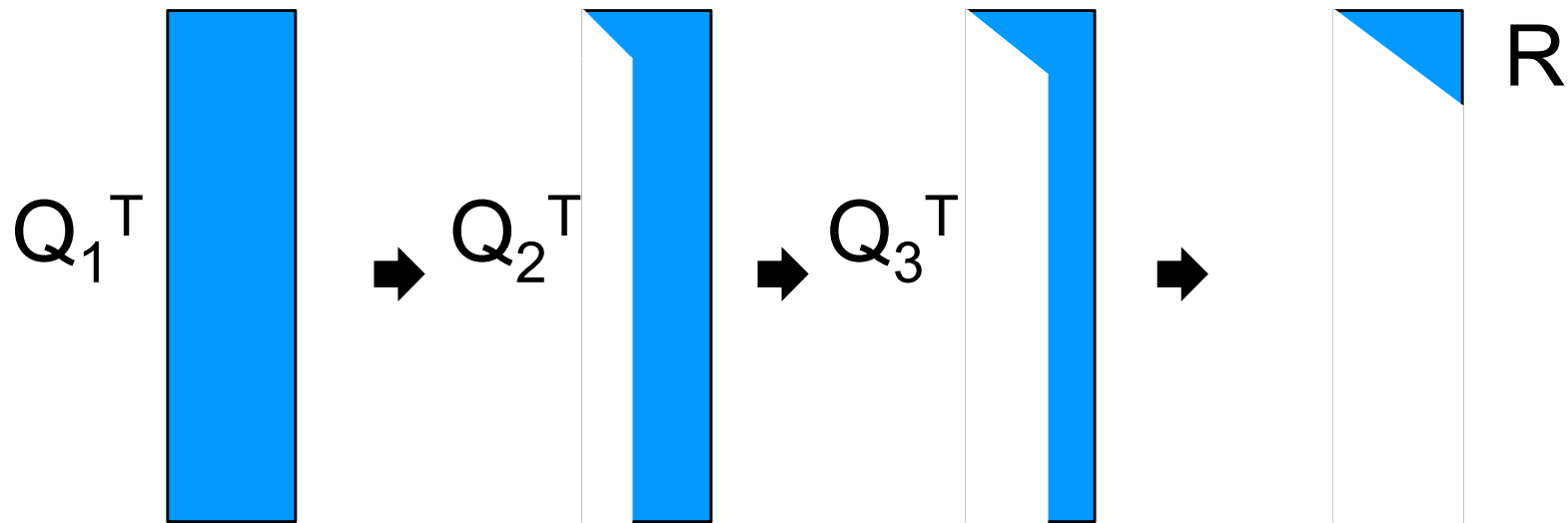
Standard QR Block Reduction

- We have a $m \times n$ matrix A we want to reduce to upper triangular form.



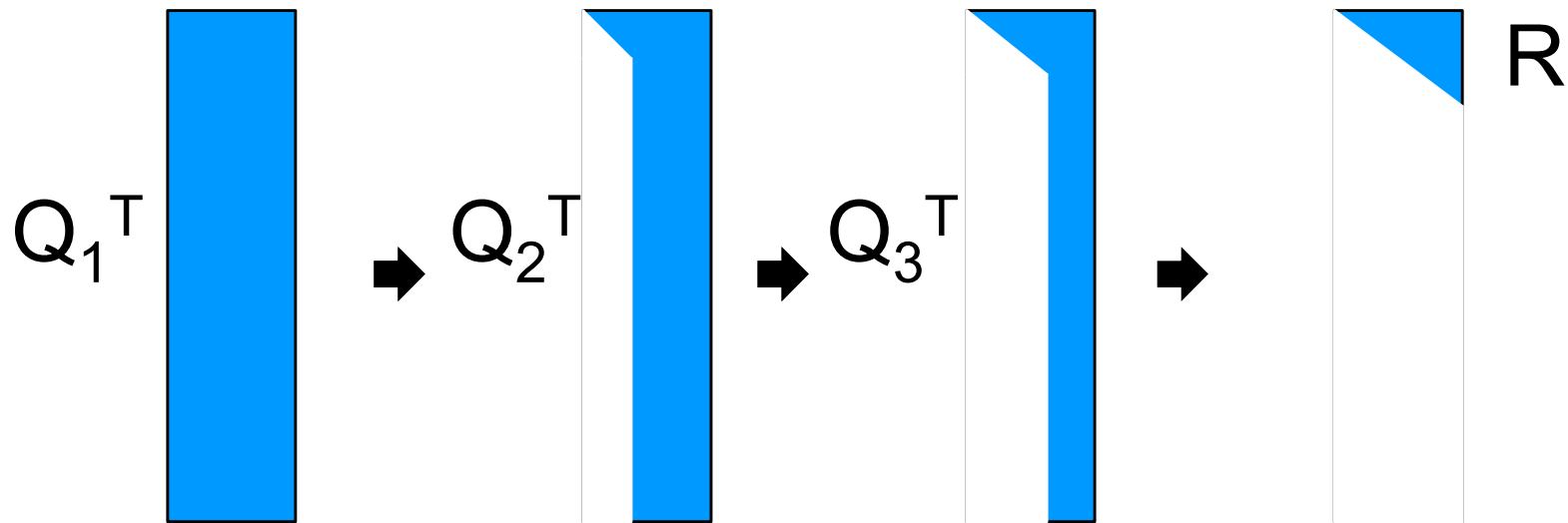
Standard QR Block Reduction

- We have a $m \times n$ matrix A we want to reduce to upper triangular form.



Standard QR Block Reduction

- We have a $m \times n$ matrix A we want to reduce to upper triangular form.



$$A = Q_1 Q_2 Q_3 R = QR$$



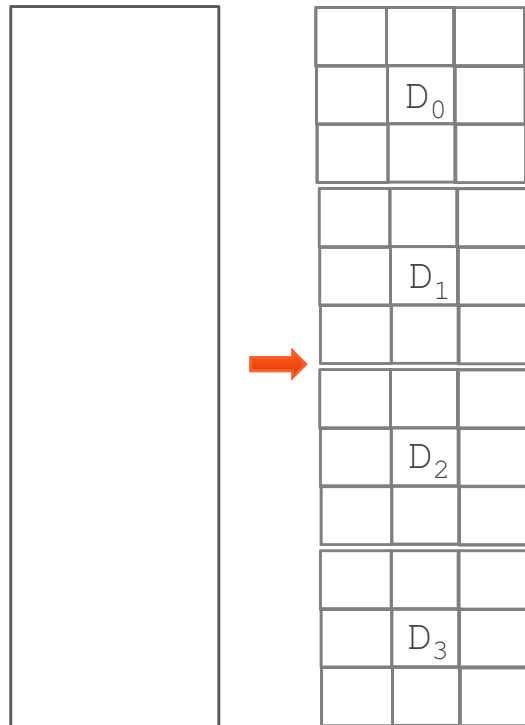
Communication Avoiding QR Example



A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications*, pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.



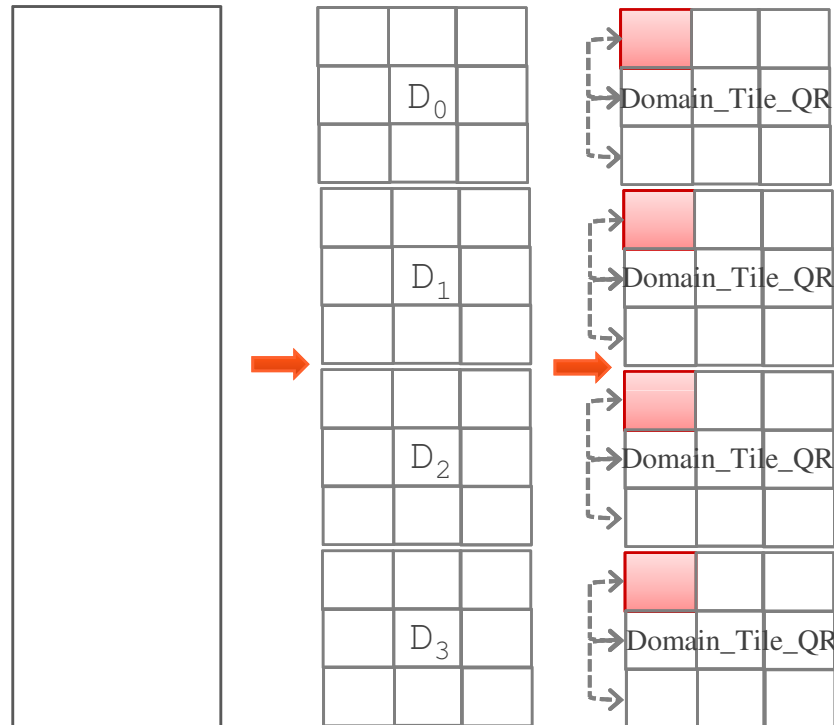
Communication Avoiding QR Example



A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications*, pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.



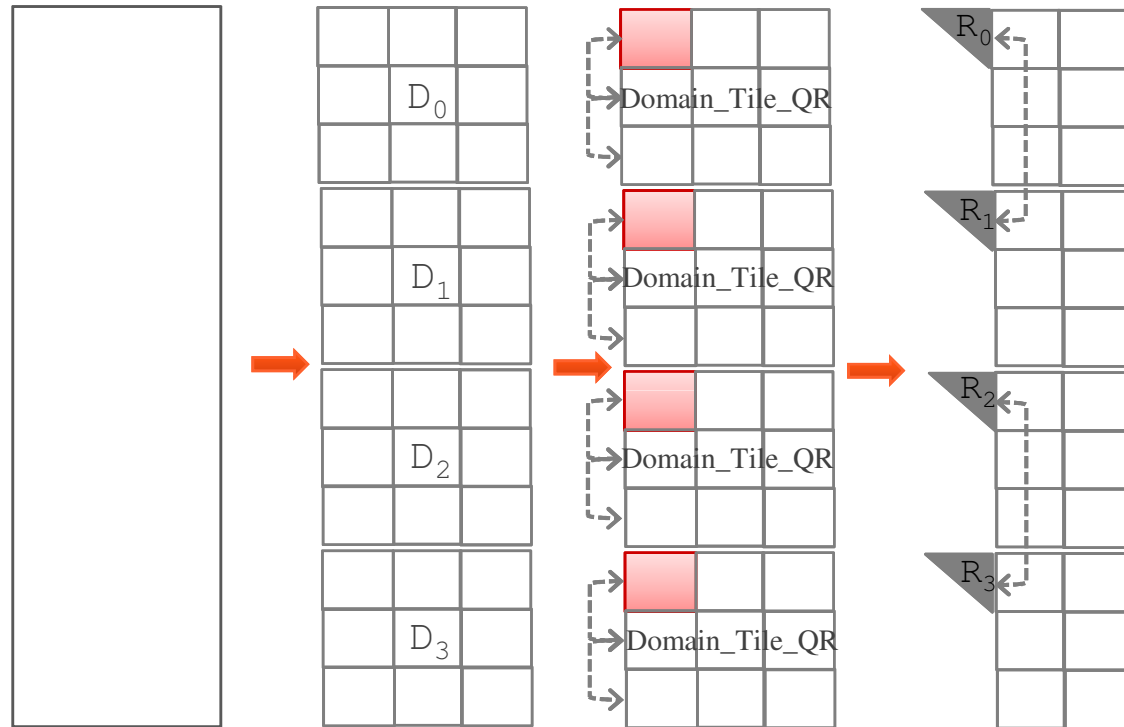
Communication Avoiding QR Example



A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications*, pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.



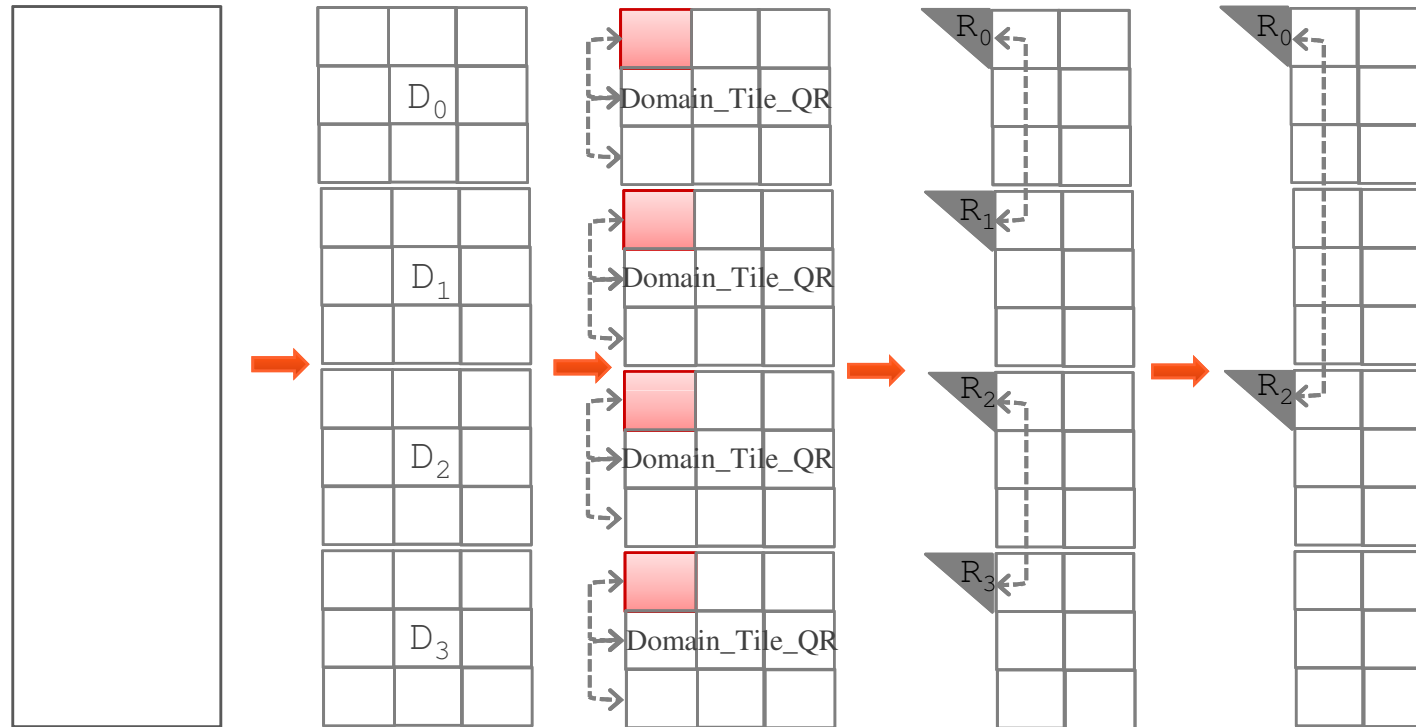
Communication Avoiding QR Example



A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications*, pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.



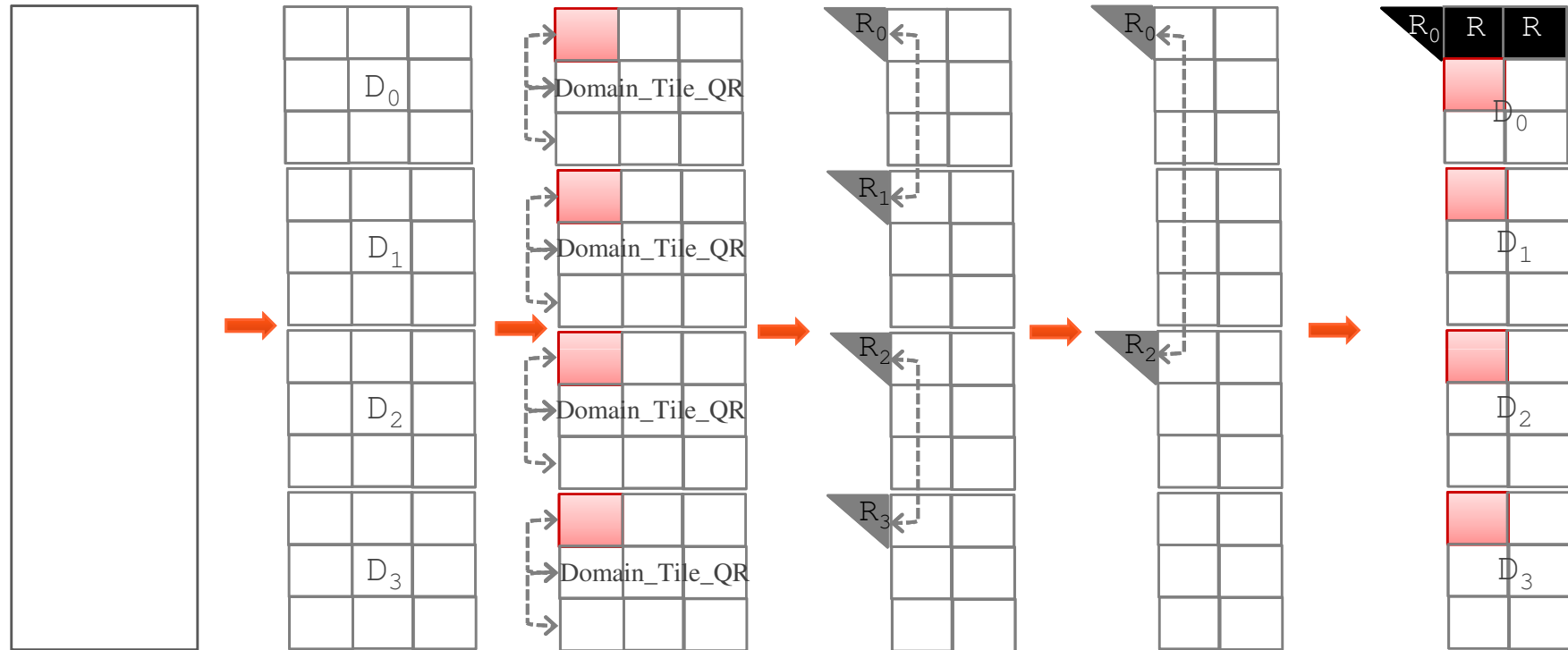
Communication Avoiding QR Example



A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications*, pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.



Communication Avoiding QR Example



A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications*, pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Challenges of using GPUs

- **High levels of parallelism**
Many GPU cores, serial kernel execution
[e.g. 240 in the Nvidia Tesla; up to 512 in *Fermi* - to have concurrent kernel execution]
- **Hybrid/heterogeneous architectures**
Match algorithmic requirements to architectural strengths
[e.g. small, non-parallelizable tasks to run on CPU, large and parallelizable on GPU]
- **Compute vs communication gap**
Exponentially growing gap; persistent challenge
[Processor speed improves 59%, memory bandwidth 23%, latency 5.5%]
[on all levels, e.g. a GPU Tesla C1070 (4 x C1060) has compute power of O(1,000) Gflop/s but GPUs communicate through the CPU using O(1) GB/s connection]

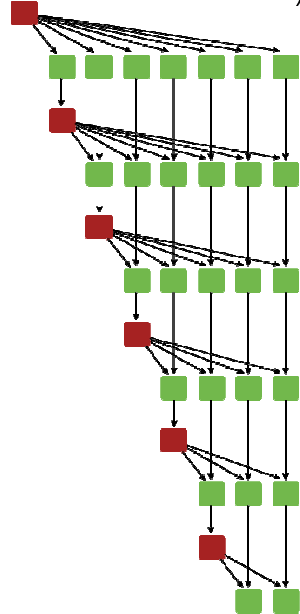
Hybrid Computing

- Match algorithmic requirements to architectural strengths of the hybrid components

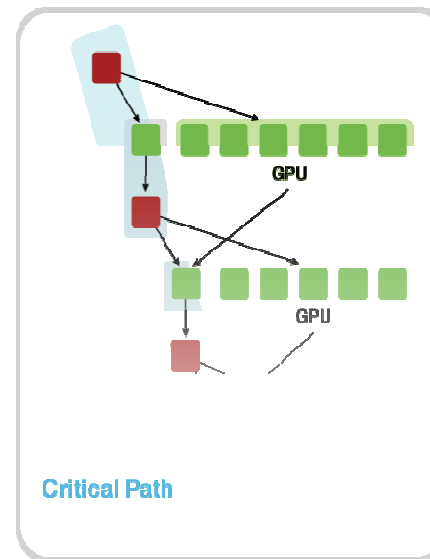
Multicore: small tasks/tiles

Accelerator: large data parallel tasks

Algorithms as DAGs
(small tasks/tiles for **multicore**)



Current hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)



- e.g. split the computation into tasks; define critical path that “clears” the way for other large data parallel tasks; proper schedule the tasks execution
- Design algorithms with well defined “*search space*” to facilitate auto-tuning



Cholesky on **multicore + multi-GPUs**

• **Hardware**

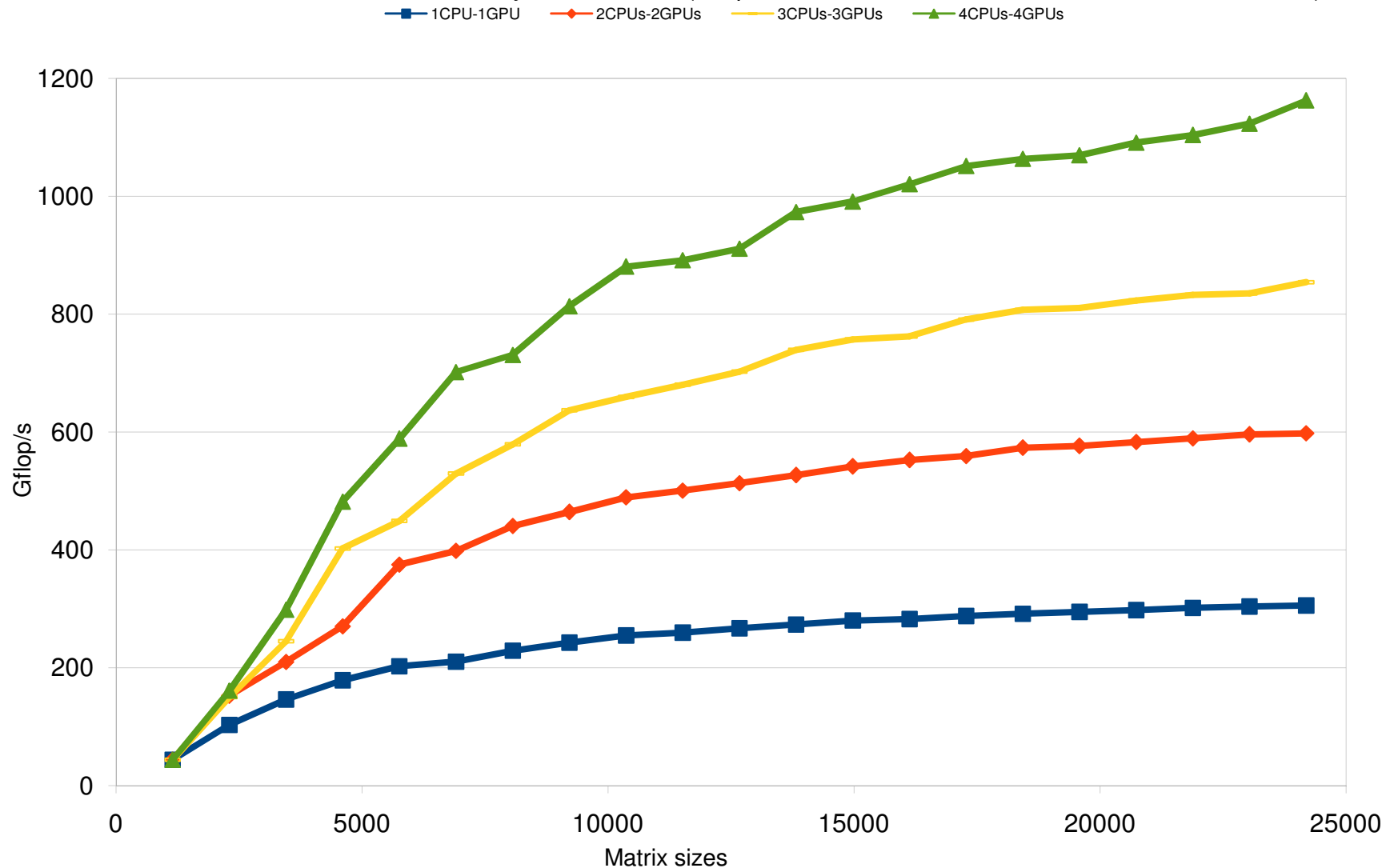
- HOST: Two-dual core AMD Opteron 1.8GHz, **2GB** memory
- DEVICE:
 - 4 GPU TESLA C1070 1.44GHz
 - 240 computing cores per GPU
 - 4GB memory per GPU
 - Single precision floating point performance (NVIDIA PEAK): 4.14 Tflop/s
 - Memory bandwidth: 408 GB/s
 - System interface: PCIeexpress

• **Memory limitations prevented runs on larger sizes**



SP Cholesky on Multicore + Multi GPUs

Parallel Performance of the hybrid SPOTRF (4 Opteron 1.8GHz and 4 GPU TESLA C1060 1.44GHz)





Performance of Single Precision on Conventional and GPU's

- Realized have the similar situation on our commodity processors.
 - That is, SP is 2X as fast as DP on many systems
- The Intel Xeon and AMD Opteron have SSE3
 - 2 flops/cycle DP
 - 4 flops/cycle SP
- IBM PowerPC has AltiVec
 - 8 flops/cycle SP
 - 4 flops/cycle DP
 - No DP on AltiVec



NVIDIA Tesla



Best case reality: 240 mul-adds per clock

Just able to do the mul-add so 2/3 or 624 Gflop/s of theoretical peak

All this is single precision

Double precision is 78 Gflop/s peak (Factor of 8 from SP; exploit mixed prec)

Single precision is faster because:

- Operations are faster
- Reduced data motion
- Larger blocks gives higher locality in cache



Idea Goes Something Like This...

- **Exploit 32 bit floating point as much as possible.**
 - **Especially for the bulk of the computation**
- **Correct or update the solution with selective use of 64 bit floating point to provide a refined results**
- **Intuitively:**
 - **Compute a 32 bit result,**
 - **Calculate a correction to 32 bit result using selected higher precision and,**
 - **Perform the update of the 32 bit results with the correction using high precision.**



Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, $Ax = b$, can work this way.

```
L U = lu(A) SINGLE  $O(n^3)$ 
x = L \ (U \ b) SINGLE  $O(n^2)$ 
r = b - Ax DOUBLE  $O(n^2)$ 
WHILE || r || not small enough
  z = L \ (U \ r)
  x = x + z DOUBLE  $O(n^1)$ 
  r = b - Ax DOUBLE  $O(n^2)$ 
END
```

2)

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.



Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, $Ax = b$, can work this way.

```
L U = lu(A) SINGLE  $O(n^3)$ 
x = L \ (U \ b) SINGLE  $O(n^2)$ 
r = b - Ax DOUBLE  $O(n^2)$ 
WHILE || r || not small enough
  z = L \ (U \ r) SINGLE  $O(n^2)$ 
  x = x + z DOUBLE  $O(n^1)$ 
  r = b - Ax DOUBLE  $O(n^2)$ 
END
```

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

- Requires extra storage, total is 1.5 times normal;
- $O(n^3)$ work is done in lower precision
- $O(n^2)$ work is done in high precision
- Problems if the matrix is ill-conditioned in sp; $O(10^8)$

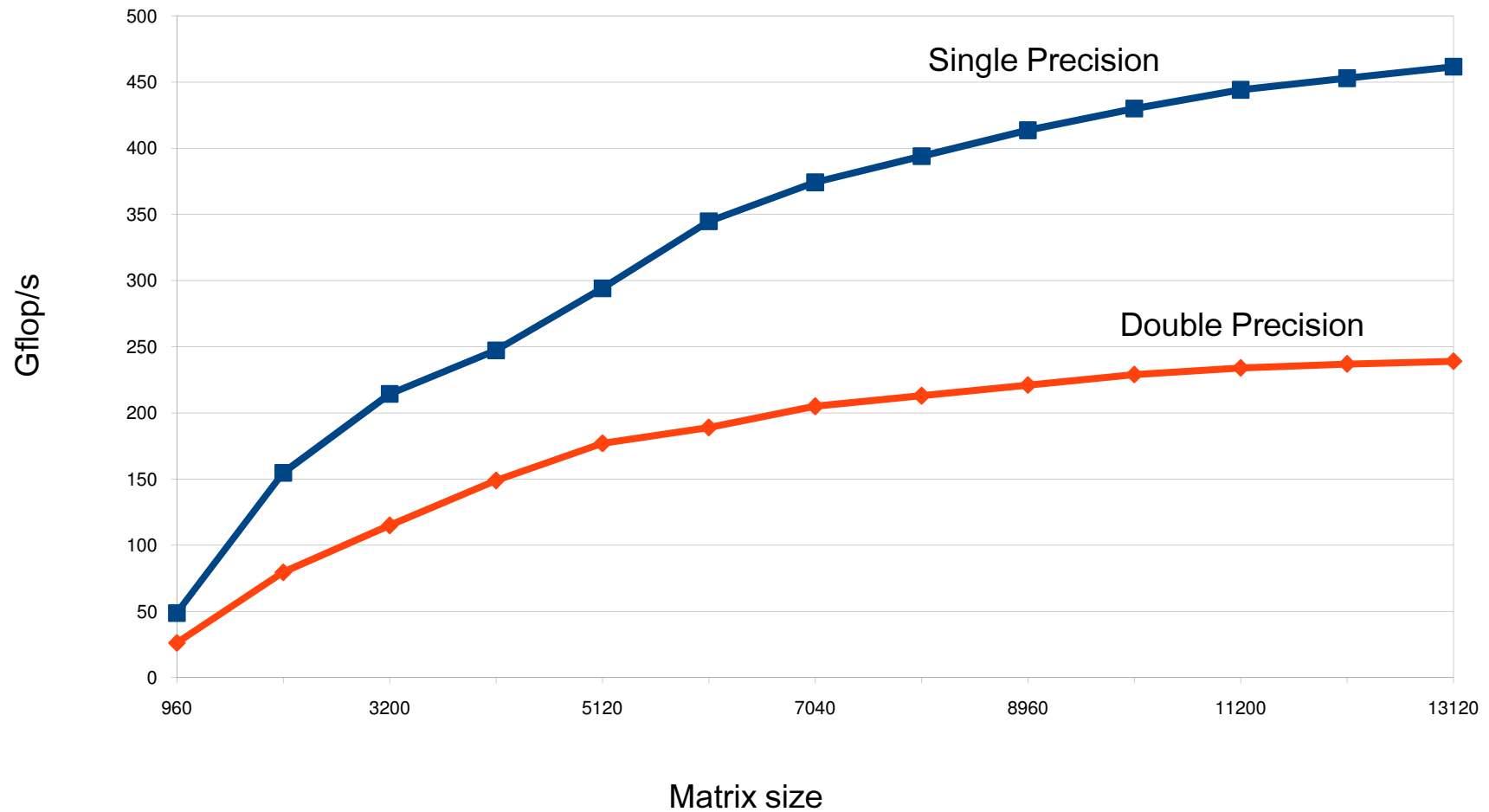


Results for Mixed Precision Iterative Refinement for Dense $Ax = b$

- Single precision is faster than DP because:
 - **Higher parallelism within floating point units**
 - 4 ops/cycle (usually) instead of 2 ops/cycle
 - **Reduced data motion**
 - 32 bit data instead of 64 bit data
 - **Higher locality in cache**
 - More data items in cache



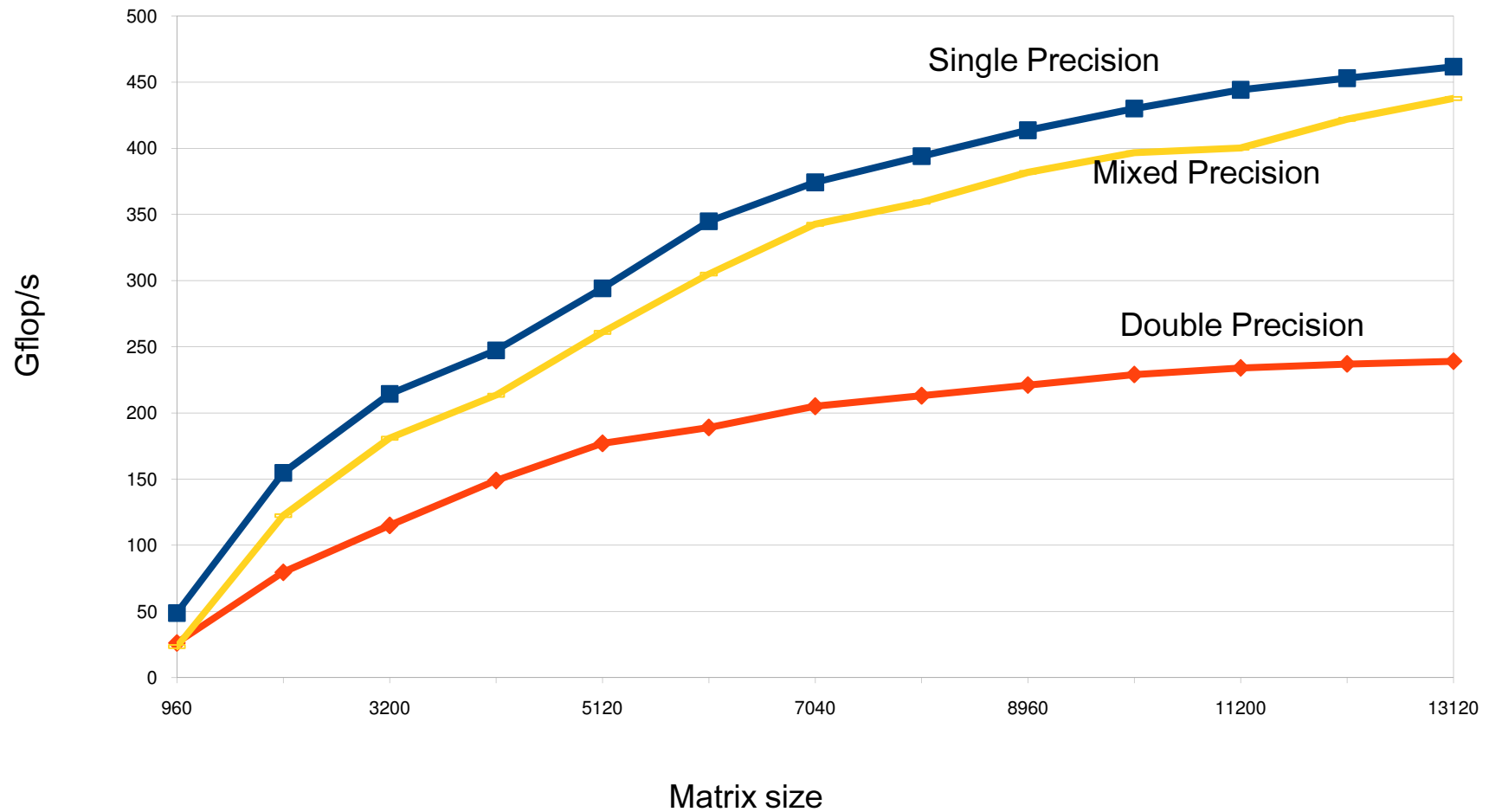
$$Ax = b$$



Tesla C2050, 448 CUDA cores (14 multiprocessors x 32) @ 1.15 GHz.,
3 GB memory, connected through PCIe to a quad-core Intel @2.5 GHz.



$$Ax = b$$

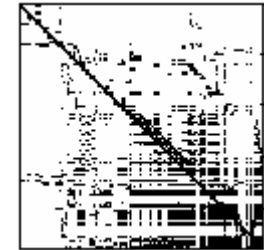
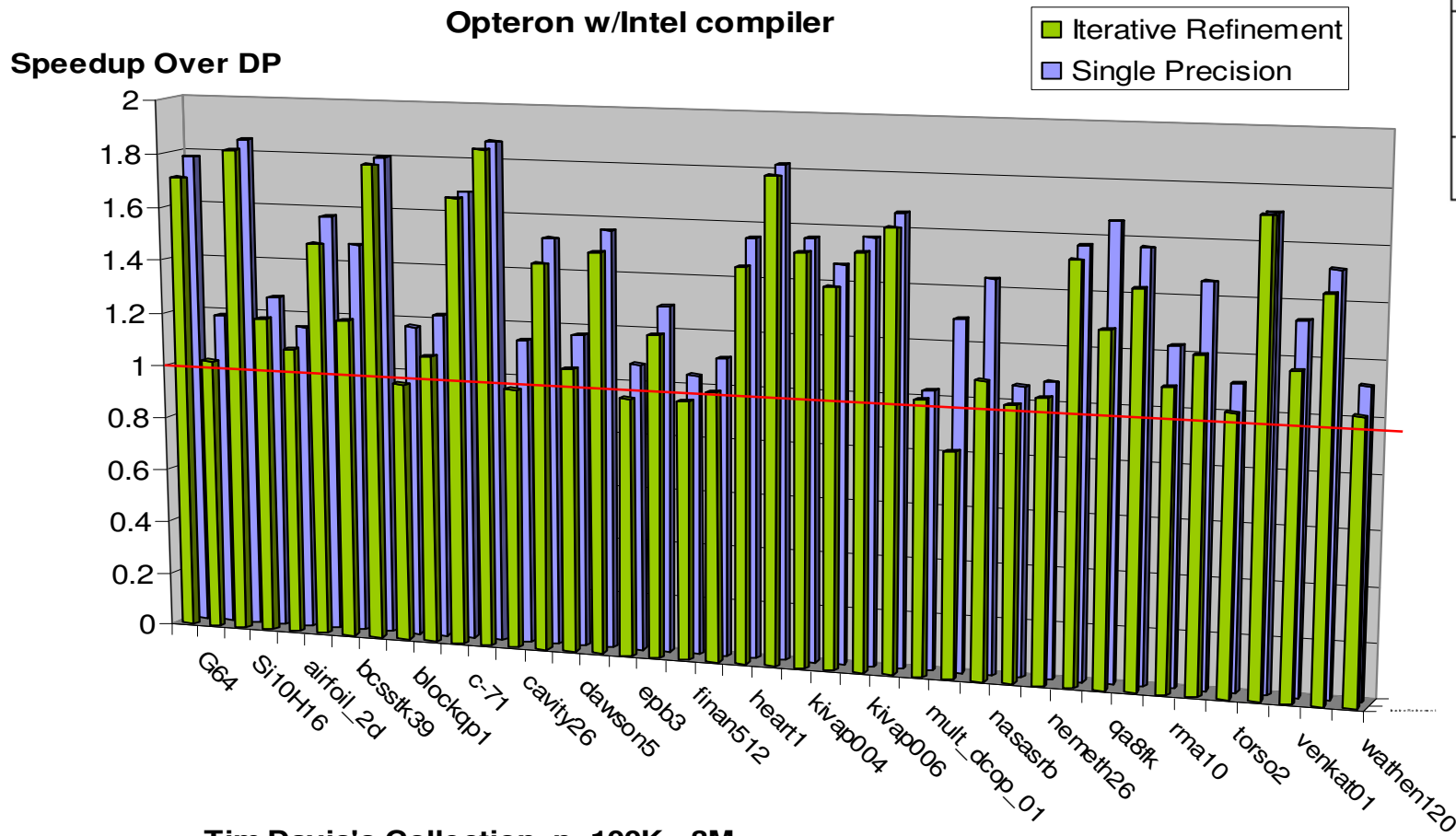


Tesla C2050, 448 CUDA cores (14 multiprocessors x 32) @ 1.15 GHz.,
3 GB memory, connected through PCIe to a quad-core Intel @2.5 GHz.



Sparse Direct Solver and Iterative Refinement

MUMPS package based on multifrontal approach which generates small dense matrix multiplies



Sparse Iterative Methods (PCG)

- Outer/Inner Iteration**

Outer iterations using 64 bit floating point

Inner iteration:

In 32 bit floating point

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

for $i = 1, 2, \dots$

 solve $Mz^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)T} z^{(i-1)}$

 if $i = 1$

$p^{(1)} = z^{(0)}$

 else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

 endif

$q^{(i)} = Ap^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)T} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

 check convergence; continue if necessary

end

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

for $i = 1, 2, \dots$

 solve $Mz^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)T} z^{(i-1)}$

 if $i = 1$

$p^{(1)} = z^{(0)}$

 else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

 endif

$q^{(i)} = Ap^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)T} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

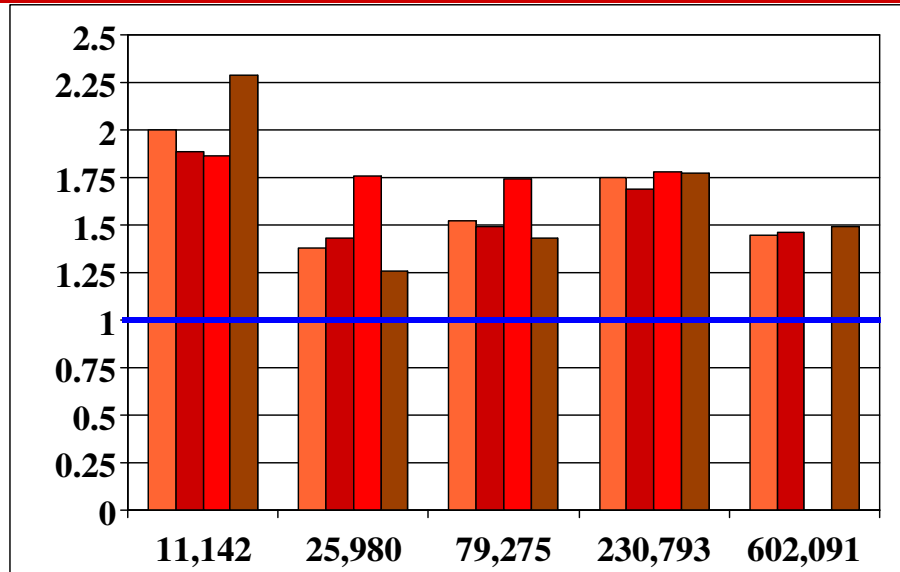
 check convergence; continue if necessary

end

- Outer iteration in 64 bit floating point and inner iteration in 32 bit floating point**



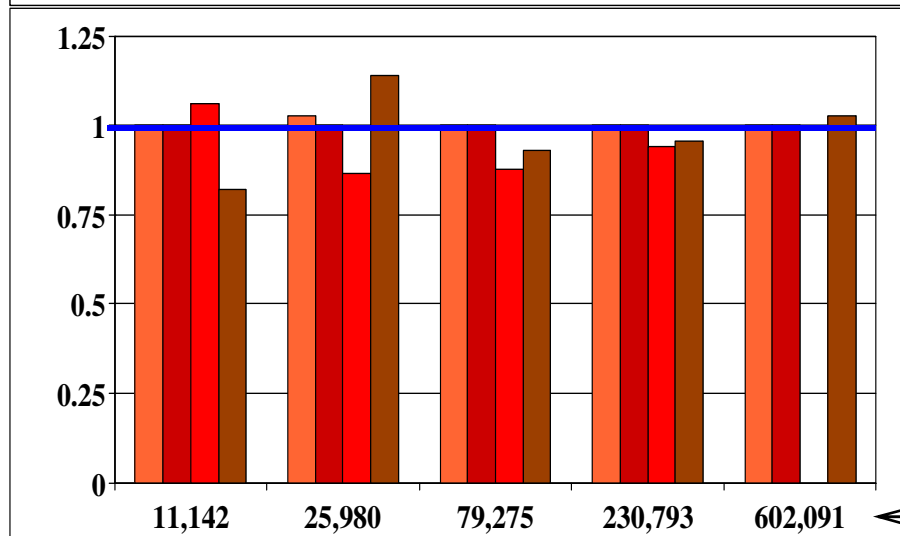
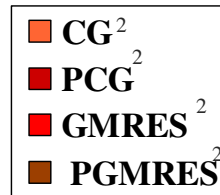
Mixed Precision Computations for Sparse Inner/Outer-type Iterative Solvers



Speedups for mixed precision

Inner SP/Outer DP (SP/DP) iter. methods vs DP/DP (CG², GMRES², PCG², and PGMRES² with diagonal prec.)

(Higher is better)



Iterations for mixed precision

SP/DP iterative methods vs DP/DP

(Lower is better)

Machine:

Intel Woodcrest (3GHz, 1333MHz bus)

Stopping criteria:

Relative to r_0 residual reduction (10^{-12})

6,021 18,000 39,000 120,000 240,000

← Matrix size
← Condition number



Intriguing Potential

- Exploit lower precision as much as possible
 - Payoff in performance
 - Faster floating point
 - Less data to move
- Automatically switch between SP and DP to match the desired accuracy
 - Compute solution in SP and then a correction to the solution in DP
- Potential for GPU, FPGA, special purpose processors
 - Use as little you can get away with and improve the accuracy
- Applies to sparse direct and iterative linear systems and Eigenvalue, optimization problems, where Newton's method is used.

$$x_{i+1} - x_i = -\frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Correction = - A\|(b - Ax)



A Call to Action



61

- Hardware has changed dramatically while software ecosystem has remained stagnant
- Need to exploit new hardware trends (e.g., manycore, heterogeneity) that cannot be handled by existing software stack, memory per socket trends
- Emerging software technologies exist, but have not been fully integrated with system software, e.g., UPC, Cilk, CUDA, HPCS
- Community codes unprepared for sea change in architectures
- No global evaluation of key missing components



International Exascale Software Program



Improve the world's simulation and modeling capability by improving the coordination and development of the HPC software environment

Workshops:

Build an international plan for coordinating research for the next generation open source software for scientific high-performance computing



International Community Effort



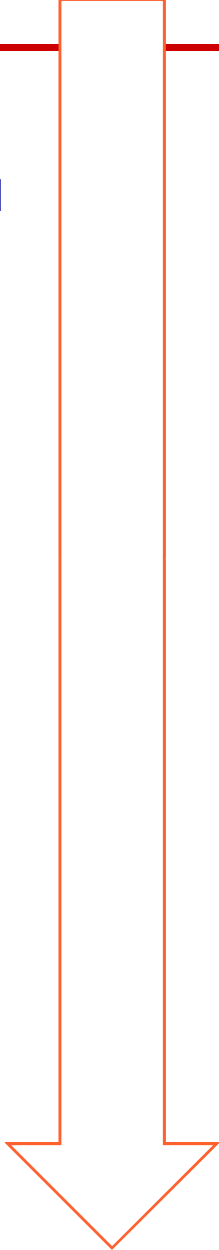
63

- **We believe this needs to be an international collaboration for various reasons including:**
 - **The scale of investment**
 - **The need for international input on requirements**
 - **US, Europeans, Asians, and others are working on their own software that should be part of a larger vision for HPC.**
 - **No global evaluation of key missing components**
 - **Hardware features are uncoordinated with software development**



Where We Are Today:



-
- SC08 (Austin TX) meeting to generate interest
 - Funding from DOE's Office of Science & NSF Office of Cyberinfrastructure and sponsorship by Europeans and Asians
 - US meeting (Santa Fe, NM) April 6-8, 2009
 - 65 people
 - European meeting (Paris, France) June 28-29, 2009
 - Outline Report
 - Asian meeting (Tsukuba Japan) October 18-20, 2009
 - Draft roadmap
 - Refine Report
 - SC09 (Portland OR) BOF to inform others
 - Public Comment; Draft Report presented
 - European meeting (Oxford, UK) April 13-14, 2010
 - Refine and prioritize roadmap
 - Explore governance structure and management models
 - Maui Meeting October 18-19, 2010
 - Kobe Meeting - Spring 2011
- 
- | |
|----------|
| Nov 2008 |
| Apr 2009 |
| Jun 2009 |
| Oct 2009 |
| Nov 2009 |
| Apr 2010 |
| Oct 2010 |

64



IESP Executive Committee

65

- Jack Dongarra, UTK & ORNL
- Pete Beckman, ANL
- Patrick Aerts, NWO Netherlands
- Franck Cappello, INRIA, France
- Thom Dunning, NCSA
- Thomas Lippert, Juelich, Germany
- Satoshi Matsuoka, TiTech, Japan
- Paul Messina, ANL
- Anne Trefethen, Oxford, UK
- Mateo Valero, BSC, Spain



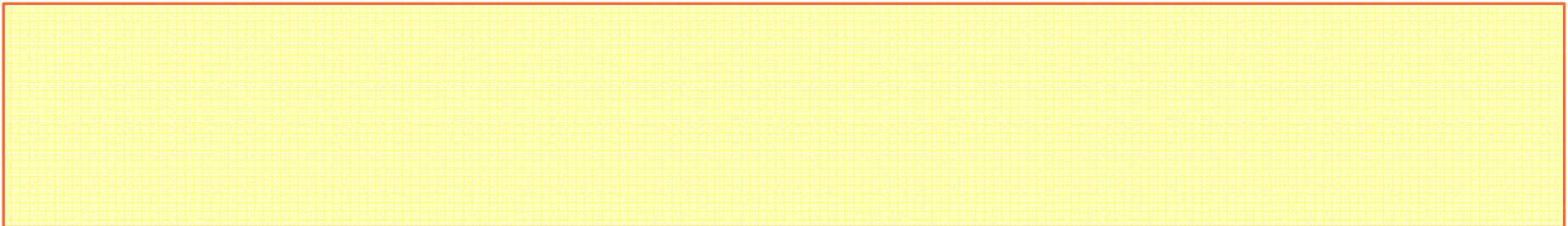
Roadmap Purpose

- The IESP software roadmap is a planning instrument designed to enable the international HPC community to improve, coordinate and leverage their collective investments and development efforts.
- After we determine what needs to be accomplished, our task will be to construct the organizational structures suitable to accomplish the work



Roadmap Components

www.exascale.org





European Exascale Software Initiative - EESI

- A detailed evaluation of how Europe is positioned, its strengths and weaknesses, in the overall international HPC landscape and competition
 - Are European stakeholders willing/able to build an exa-scale prototype/by when?
 - Actors/users/projects
- A European and international vision and roadmap
 - Why is exa-scale initiatives important? Who cares? Impact?
 - Scientific
 - Economic
 - Social benefits
- Dissemination actions
 - Visibility of EESI: who is the potential target public?
 - R&D stakeholders
 - EC and national policy-makers
 - Society as a whole
- Identification of opportunities of worldwide collaborations
 - European position inside IESP: who's doing/deciding what?
 - Contribution to the international dialogs: mutual benefits!

EC and G8 Related

- G8 has a call out for “Interdisciplinary Program on Application Software towards Exascale Computing for Global Scale Issues”
 - 10 million € over three years
 - An initiative between Research Councils from Canada, France, Germany, Japan, Russia, the UK, and the USA
 - 78 preproposals submitted, 25 selected, expect to fund 6-10
 - Full proposals due August 25th
- EC FP7: Exascale computing, software and simulation
 - Announcement due September 28, 2010
 - 25 million €
 - 2 or 3 integrated project to be funded



If you are wondering what's beyond ExaFlops

**Mega, Giga, Tera,
Peta, Exa, Zetta ...**

10^3 kilo

10^6 mega

10^9 giga

10^{12} tera

10^{15} peta

10^{18} exa

10^{21} zetta

10^{24} yotta

10^{27} xona

10^{30} weka

10^{33} vunda

10^{36} uda

10^{39} treda

10^{42} sorta

10^{45} rinta

10^{48} quexa

10^{51} pepta

10^{54} ocha

10^{57} nena

10^{60} minga

10^{63} luma

- www.exascale.org

INTERNATIONAL EXASCALE SOFTWARE PROJECT



ROADMAP

Jack Dongarra	Alok Choudhary	Yuzaka Ishikawa	Paul Messina	John Shalf	Aad van der Steen
Pete Deckman	Sudip Dossanjh	Fred Johnson	Dernd Mohr	David Skinner	Fred Stretz
Terry Moore	Al Geist	Sanjay Kala	Matthias Mueller	Thomas Sterling	Bob Sugar
Jean-Claude Andre	Bill Gropp	Richard Kenway	Wolfgang Nagel	Rick Stevens	Shinji Sumimoto
Jean-Yves Berthou	Robert Harrison	Bill Kramer	Hiroshi Nakashima	William Tang	Jeffrey Vetter
Taisuke Beku	Mark Horold	Jesus Labarta	Michael E. Papka	John Taylor	Robert Weniowski
Franck Cappello	Michael Heroux	Bob Lucas	Dan Reed	Rajeev Thakur	Kathy Yelick
Barbara Chapman	Aditya Holte	Barney Maccabe	Mitsuhsa Sato	Anne Trefethen	
Xuebin Chi	Koh Hotta	Satoshi Matsuoka	Ed Seidel	Marc Snir	

SPONSORS





Summary

- **Major Challenges are ahead for extreme computing**
 - **Power**
 - **Parallelism**
 - **Hybrid**
 - **Fault Tolerance**
 - **... and many others not discussed here**
- **We will need completely new approaches and technologies to reach the Exascale level**
- **This opens up many new opportunities for applied mathematicians**

- “We can only see a short distance ahead, but we can see plenty there that needs to be done.”
 - *Alan Turing (1912–1954)*

Shackleton's Quote on Exascale



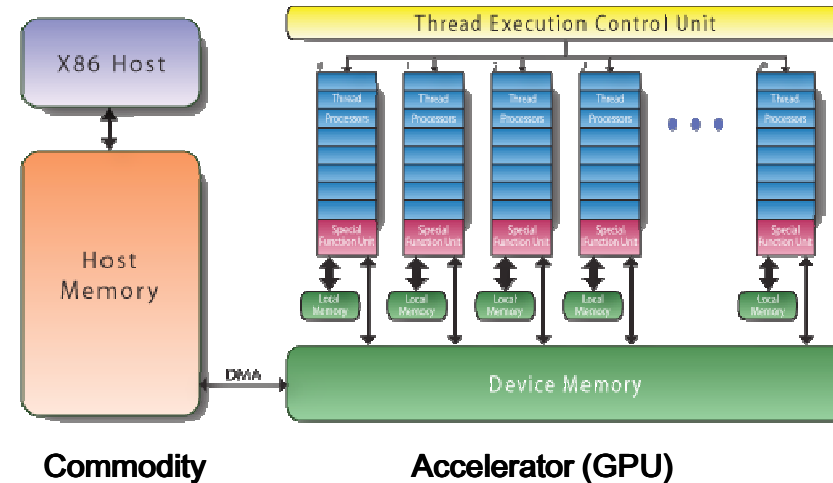
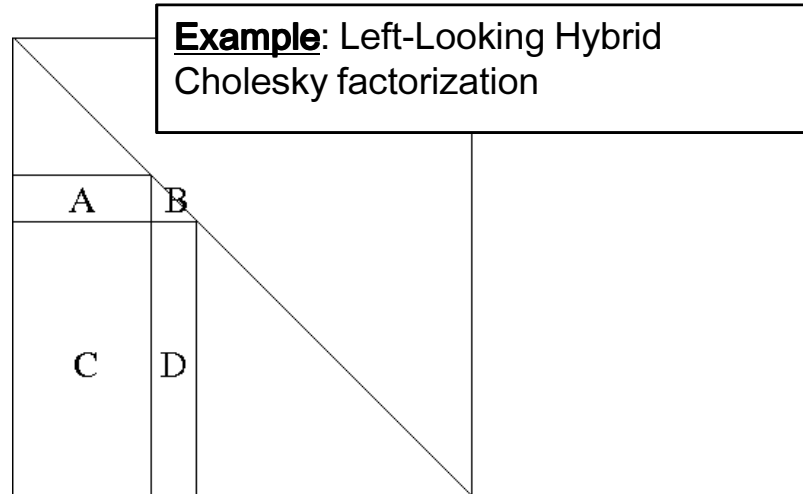
***Ernest Shackleton's 1907 ad in London's Times,
recruiting a crew to sail with him on his
exploration of the South Pole***

"Wanted. Men/women for hazardous architectures. Low wages. Bitter cold. Long hours of software development. Safe return doubtful. Honor and recognition in the event of success."





One-Sided Dense Matrix Factorizations (LU, QR, and Cholesky) from MAGMA



MATLAB code

LAPACK code

Hybrid code

(1) $B = B - A \cdot A'$

`ssyrk("L", "N", &nb, &j, &mone, hA(j,0), ...)` `cublasSsyrk('L', 'N', nb, j, mone, dA(j,0), ...)`

`cublasGetMatrix(nb, nb, 4, dA(j, j), *lda, hwork, nb)`

(2) $B = \text{chol}(B, \text{'lower'})$ `spotrf("L", &nb, hA(j, j), lda, info)` `cublasSgemm("N", 'T', j, ...)`

(3) $D = D - C \cdot A'$ `sgemm("N", "T", &j, ...)` `spotrf("L", &nb, hwork, &nb, info)`

`cublasSetMatrix(nb, nb, 4, hwork, nb, dA(j, j), *lda)`

(4) $D = B \cdot D$ `strsm("R", "L", "T", "N", &j, ...)`

`cublasStrsm('R', 'L', 'T', 'N', j, ...)`

CUDA implementation:

- `a_ref` points to the GPU memory
- GPU kernels are started asynchronously which results in overlapping the GPU `sgemm` with transferring `T` to the CPU, factoring it, and sending the result back to the GPU
- For full details see http://www.cs.utk.edu/~tomov/magma/spotrf_gpu.cpp



Communication Reducing Iterative Methods

- **Take k -steps of Krylov subspace method**
 - **GMRES, CG, Lanczos, Arnoldi**
 - **Assume matrix “well-partitioned,” with modest surface-to-volume ratio**
 - **Parallel implementation**
 - Conventional: $O(k \log p)$ messages
 - New: $O(\log p)$ messages - optimal
 - **Serial implementation**
 - Conventional: $O(k)$ moves of data from slow to fast memory
 - New: $O(1)$ moves of data - optimal
 - **Can incorporate some preconditioners**
 - Need to be able to “compress” interactions between distant i, j
 - Hierarchical, semiseparable matrices ...
 - **Lots of speed up possible (modeled and measured)**
 - Price: some redundant computation



Minimizing Communication of to solve $Ax=b$

GMRES

- **GMRES: find x in $\text{span}\{b, Ab, \dots, A^k b\}$ minimizing $\|Ax-b\|_2$**
- **Cost of k steps of standard GMRES vs new GMRES**

Standard GMRES

for $i=1$ to k

$w = A \cdot v(i-1)$

MGS($w, v(0), \dots, v(i-1)$)

update $v(i), H$

endfor

solve LSQ problem with H

Sequential: #words_moved =

$O(k \cdot \text{nnz})$ from SpMV

+ $O(k^2 \cdot n)$ from MGS

Parallel: #messages =

$O(k)$ from SpMV

+ $O(k^2 \cdot \log p)$ from MGS

Communication-avoiding GMRES

$W = [v, Av, A^2v, \dots, A^k v]$

$[Q, R] = \text{TSQR}(W)$... "Tall Skinny QR"

Build H from R , solve LSQ problem

Sequential: #words_moved =

$O(\text{nnz})$ from SpMV

+ $O(k \cdot n)$ from TSQR

Parallel: #messages =

$O(1)$ from computing W

+ $O(\log p)$ from TSQR

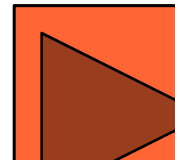
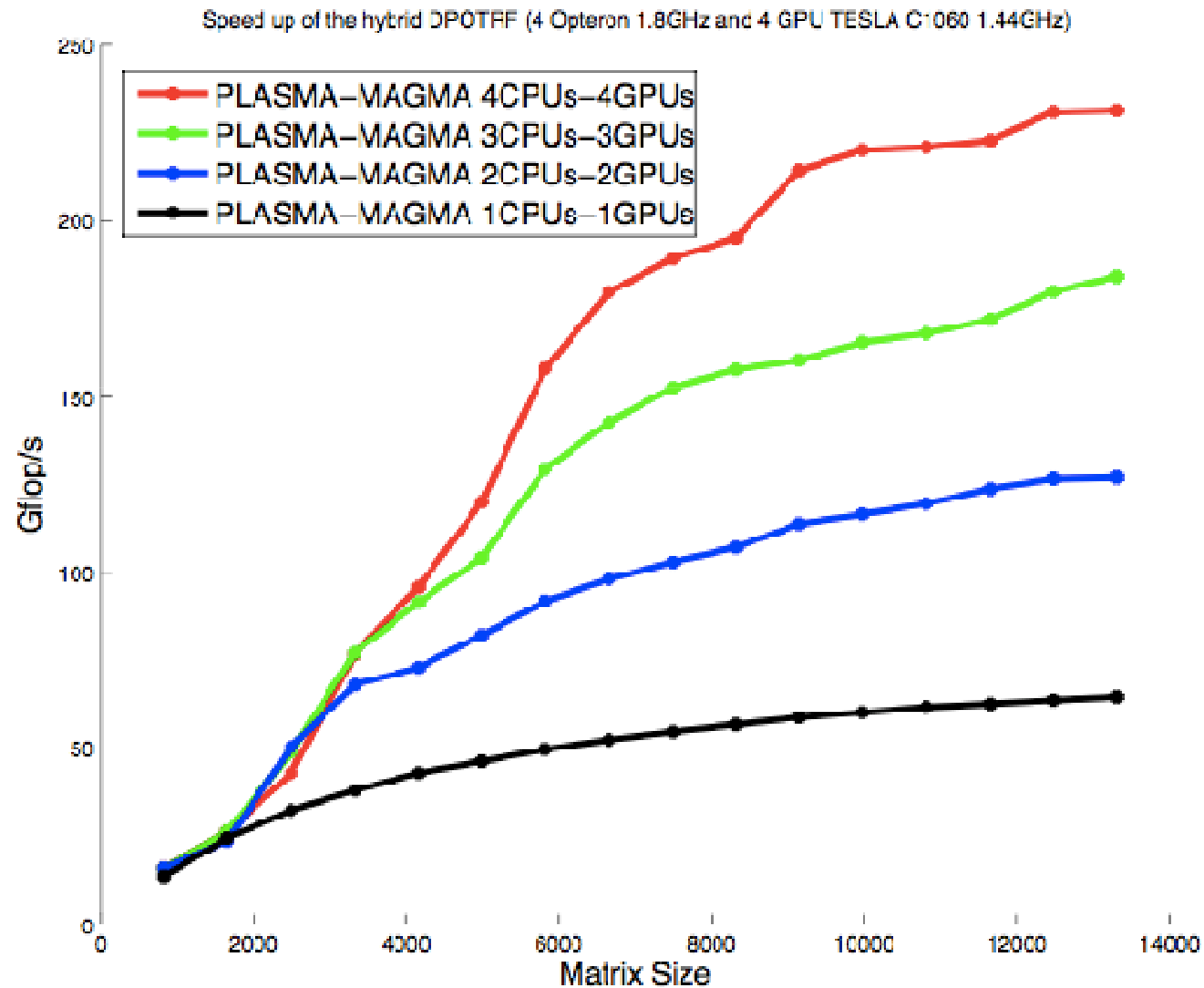
- **Numerical issue with potential loss of precision from computing W from power method.**



Systems in Italy

Rank	Site	Manufact.	Computer	Cores	Rmax Gflop/s
70	CINECA	IBM	Power 575, p6 4.7 GHz, Infiniband	5376	78680
129	Telecom	IBM	BladeCenter HS22 Cluster, Xeon QC GT 2.53 GHz, GigEthernet	8048	45528
211	CILEA	HP	Cluster Platform 3000 BL2x220, X56xx 3.0 Ghz, Infiniband QDR	4032	35665
295	Energy Company (A)	IBM	BladeCenter HS22 Cluster, Xeon QC X56xx 2.66 GHz, Infiniband	3408	31310
296	Energy Company (A)	IBM	BladeCenter HS22 Cluster, Xeon QC X56xx 2.66 GHz, Infiniband	3408	31310
297	Energy Company (A)	IBM	BladeCenter HS22 Cluster, Xeon QC X56xx 2.66 GHz, Infiniband	3408	31310
404	Sardegna Ricerche	HP	Cluster Platform 3000 BL460c G1, Xeon E5440 2.83 GHz, Infiniband	3088	27708

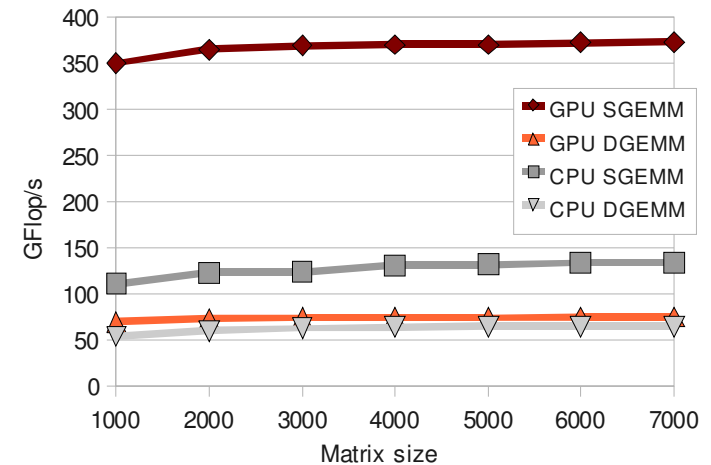
DP Cholesky with Multiple GPUs



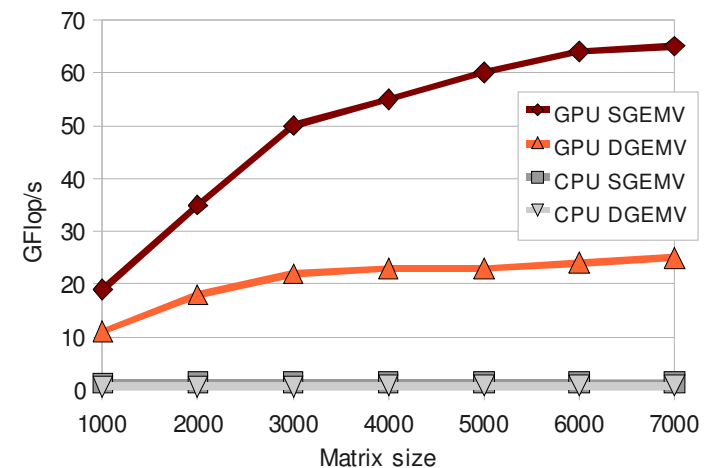
How to Code for GPUs?

- **Complex question**
 - Language, programming model, user productivity, etc
- **Recommendations**
 - **Use CUDA / OpenCL**
 [already demonstrated benefits in many areas; data-based parallelism; move to support task-based]
 - **Use GPU BLAS**
 [high level; available after introduction of shared memory - can do data reuse; leverage existing developments]
 - **Use Hybrid Algorithms**
 [currently GPUs - massive parallelism but serial kernel execution;
 hybrid approach - small non-parallelizable tasks on the CPU, large parallelizable tasks on the GPU]

GPU vs CPU GEMM

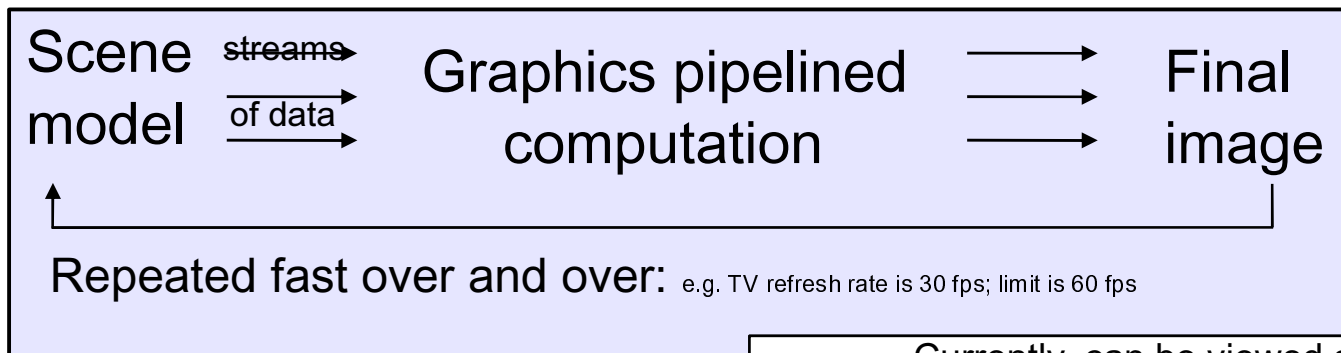


GPU vs CPU GEMV



Evolution of GPUs

GPUs: excelling in graphics rendering



- Currently, can be viewed as **multithreaded multicore vector units**

This type of computation:

- Requires **enormous computational power**
- Allows for **high parallelism**
- Needs **high bandwidth vs low latency**
(as low latencies can be compensated with deep graphics pipeline)

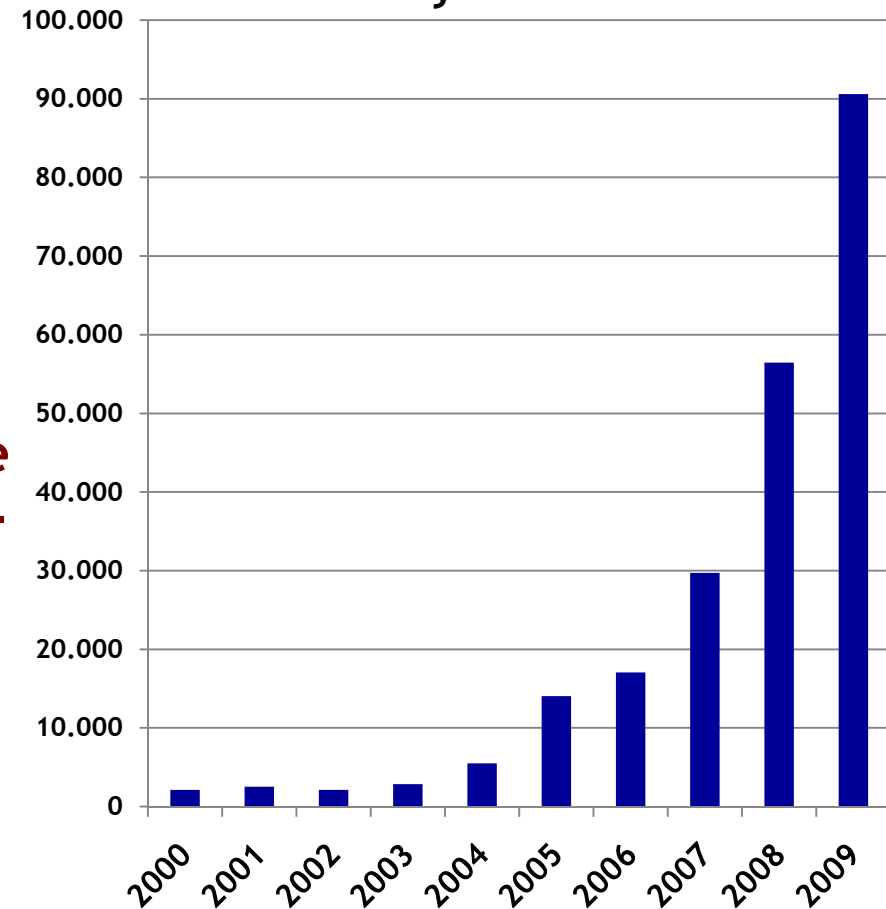
Obviously, this pattern of computation is common with many other applications



Moore's Law Reinterpreted

- **Number of cores per chip doubles every 2 year, while clock speed decreases (not increases).**
 - **Need to deal with systems with millions of concurrent threads**
 - Future generation will have billions of threads!
 - **Need to be able to easily replace inter-chip parallelism with intra-chip parallelism**
- **Number of threads of execution doubles every 2 year**

Average Number of Cores Per Supercomputer for Top20 Systems





Potential System Architectures

Systems	2009
System peak	2 Pflop/s
System memory	0.3 PB
Node performance	125 Gflop/s
Node memory BW	25 GB/s
Node concurrency	12
Interconnect BW	1.5 GB/s
System size (nodes)	18,700
Total concurrency	225,000
Storage	15 PB
IO	0.2 TB/s
MTTI	days
Power	7 MW



Conclusions

- For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware.
- This strategy needs to be rebalanced - barriers to progress are increasingly on the software side.
- Moreover, the return on investment is more favorable to software.
 - Hardware has a half-life measured in years, while software has a half-life measured in decades.
- High Performance Ecosystem out of balance
 - Hardware, OS, Compilers, Software, Algorithms, Applications
 - No Moore's Law for software, algorithms and applications



Collaborators / Support

Employment opportunities for post-docs in the **PLASMA/MAGMA** projects

PLASMA Parallel Linear Algebra Software for Multicore Architectures

<http://icl.cs.utk.edu/plasma/>

MAGMA Matrix Algebra on GPU and Multicore Architectures

<http://icl.cs.utk.edu/magma/>

Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julie & Julien Langou, Hatem Ltaief, Piotr Luszczek, Stan Tomov



The MathWorks

Microsoft



Google

Web [Images](#) [Video](#) [News](#) [Maps](#) [Desktop](#) [more »](#)

dongarra

Google Search

I'm Feeling Lucky

[Advanced Search](#)
[Preferences](#)
[Language Tools](#)

New! Try [Docs & Spreadsheets](#) and share your projects instantly.

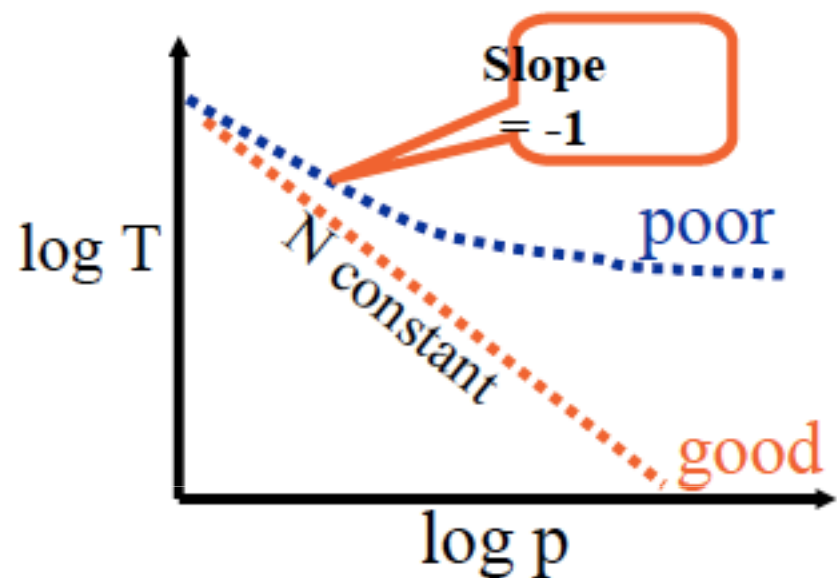
[Advertising Programs](#) - [Business Solutions](#) - [About Google](#)

- **Strong scaling: fixed problem size.**
 - Data on each node decreases as the number of nodes increases
- **Weak scaling: fixed the data size on each node.**
 - Problem size increases as the number of node increases.

Review: two definitions of scalability

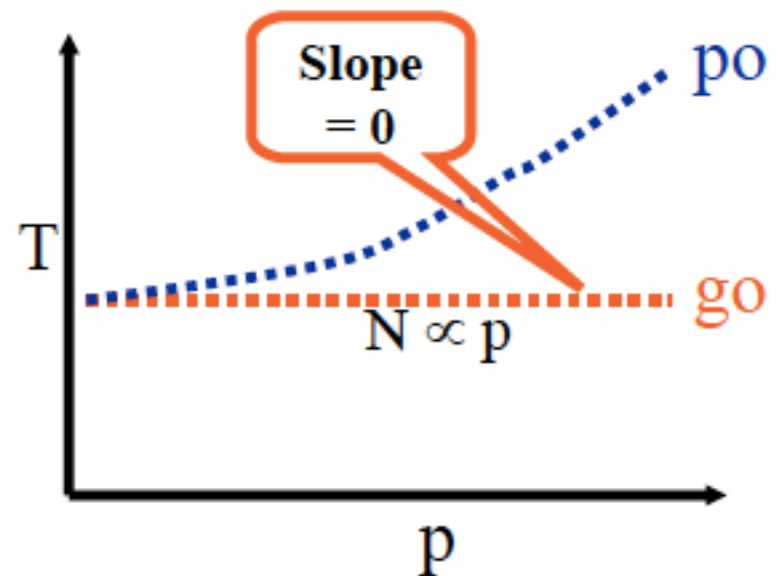
“Strong scaling”

- ◆ execution time decreases in inverse proportion to the number of processors
- ◆ *fixed size problem overall*
- ◆ often instead graphed as reciprocal, “speedup”



“Weak scaling”

- ◆ execution time remains constant, as problem size and processor number are increased in proportion
- ◆ *fixed size problem per processor*
- ◆ Various sub-types of weak-scaling “memory bound”, etc. (see Kumar et



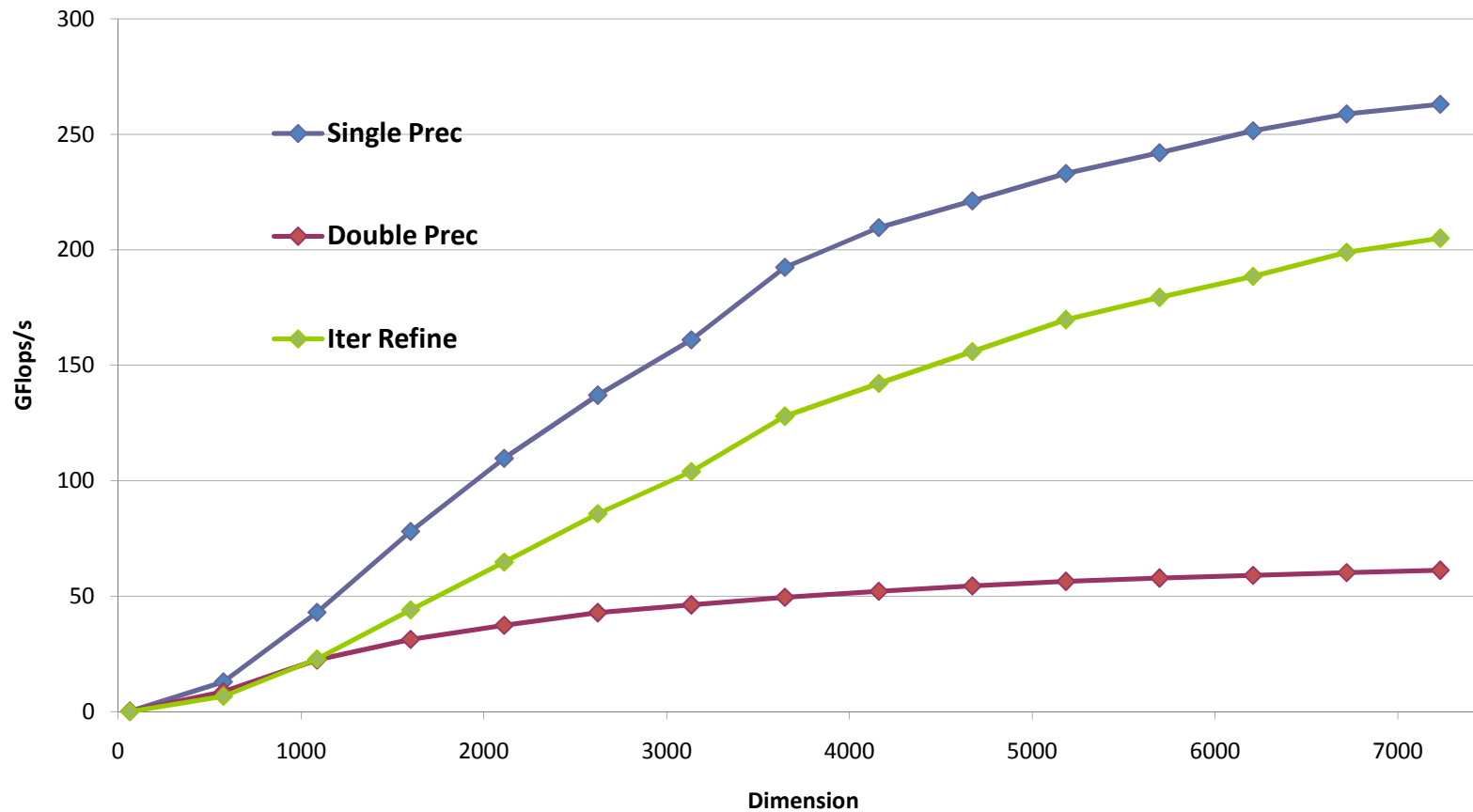


Symmetric Positive Definite

Cholesky-GPU

Intel (R) Xeon (R) E5410 2.33 GHz (8 Core)

GForce GTX 280 1.3 GHz (240 Core)



GPU : NVIDIA GeForce GTX 280

GPU BLAS : CUBLAS 2.2, s/dgemm peak: 375 / 75 GFlop/s

CPU : Intel Xeon dual socket quad-core @2.33 GHz

CPU BLAS : MKL 10.0 , s/dgemm peak: 17.5 / 8.6 GFlop/s

- **Projections**
 - Performance
 - Memory
- **Async**
 - Break fork-join
 - DAGs
 - New algorithms - numerical issues
 - Communication avoiding
 - Chaotic iteration
- **Mixed precision**
 - Iter refine
 - precondition
- **Hybrid**
 - balance
 - autotune
- **FT**
 - Number of approaches



MAGMA Software

- Available through MAGMA's homepage <http://icl.cs.utk.edu/magma/>
- Included are the 3 one-sided matrix factorizations
- Iterative Refinement Algorithm (Mixed Precision)
- Standard (LAPACK) data layout and accuracy
- Two LAPACK-style interfaces
 - CPU interface: both input and output are on the CPU
 - GPU interface: both input and output are on the GPU
- This release is intended for single GPU



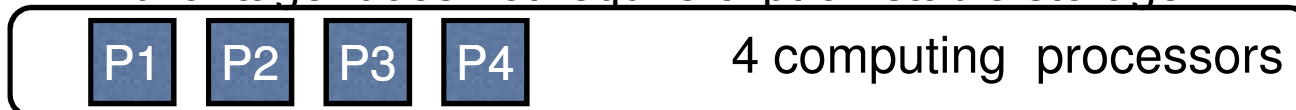
Purpose

- 92 • **The IESP software roadmap is a planning instrument designed to enable the international HPC community to improve, coordinate and leverage their collective investments and development efforts.**
- **After we determine what needs to be accomplished, our task will be to construct the organizational structures suitable to accomplish the work**

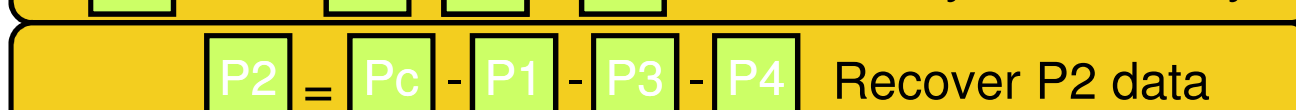
Diskless Checkpointing 1/2

Principle: Compute a checksum of the processes' memory and store it on spare processors

Advantage: does not require ckpt on stable storage.



....



A) Every process saves a copy of its local state of in memory or local disc

B) Perform a global bitstream or floating point operation on all saved local states

All processes restore its local state from the one saved in memory or local disc

Diskless Checkpointing 2/2

• Could be done at application and system levels

• Process data could be considered (and encoded) either as bit-streams or as floating point numbers.

Computing the checksum from bit-streams uses operations such as parity. Computing checksum from floating point numbers uses operations such as addition

• Can survive multiple failures of arbitrary patterns

Reed Solomon for bit-streams and weighted checksum for floating point numbers (sensitive to round-off errors).

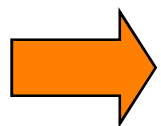
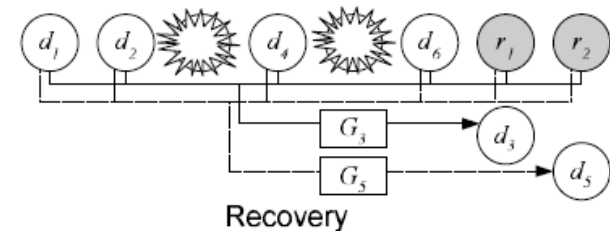
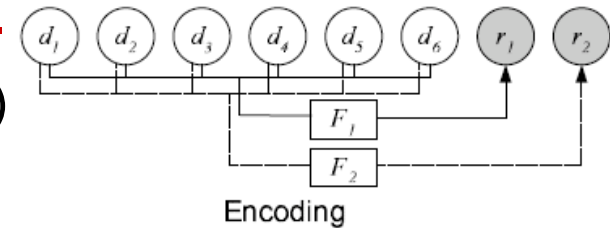
• Work with with incremental ckpt.

• Need spare nodes and double the memory occupation (to survive failures during ckpt.) --> increases the overall cost and #failures

• **Need coordinated checkpointing** or message logging protocol

• Need very fast encoding & reduction operations

• **Need automatic Ckpt protocol or program modifications**



Challenge: experiment more Diskless CKPT and in very large machines (current result are for ~1000 CPUs)



Algorithmic Based Fault Tolerance”

In 1984, **Huang and Abraham**, proposed the **ABFT** to detect and correct errors in some matrix operations on systolic arrays.

ABFT encodes data & redesign algo. to operate on encoded data. Failure are detected and corrected off-line (after execution).

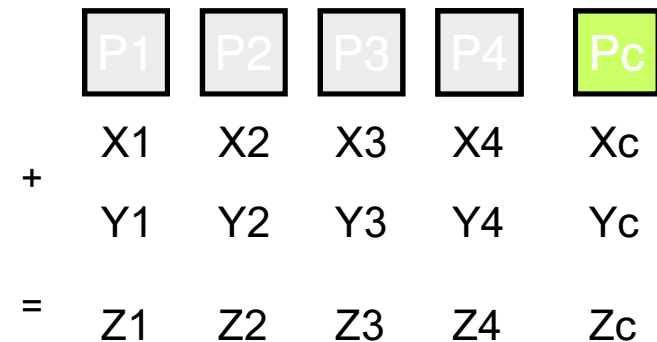
ABFT variation for on-line recovery (runtime detects failures + robust to failures):

• **Similar to Diskless ckpt.**, an extra processor is added, P_{i+1} , store the checksum of data: (vector X and Y in this case)

$$X_c = X_1 + \dots + X_p, Y_c = Y_1 + \dots + Y_p.$$

$$X_f = [X_1, \dots, X_p, X_c], Y_f = [Y_1, \dots, Y_p, Y_c],$$

• Operations are performed on X_f and Y_f instead of X and Y : $Z_f = Y_f + Z_f$



• Compared to diskless checkpointing, the memory AND CPU of P_c take part of the computation):

- **No global operation for Checksum!**
- **No local checkpoint!**

Works for many Linear Algebra operations:

Matrix Multiplication: $A * B = C \rightarrow A_c * B_r = C_f$

LU Decomposition: $C = L * U \rightarrow C_f = L_c * U_r$

Addition: $A + B = C \rightarrow A_f + B_f = C_f$

Scalar Multiplication: $c * A_f = (c * A)_f$

Transpose: $A_f^T = (A^T)_f$

Cholesky factorization & QR factorization



“Naturally fault tolerant algorithm”

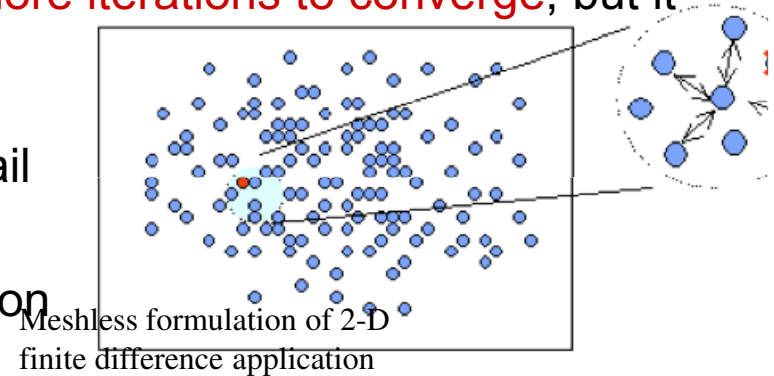
Natural fault tolerance is the ability to **tolerate failures through the mathematical properties of the algorithm itself**, without requiring notification or recovery.

The algorithm includes **natural compensation** for the lost information.

For example, **an iterative algorithm may require more iterations to converge**, but it still converges despite lost information

Assumes that a maximum of 0.1% of tasks may fail

Ex1 : Meshless iterative methods+chaotic relaxation
(asynchronous iterative methods)



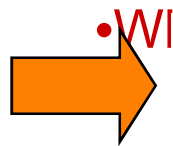
This algorithm share some features with **SelfStabilization** algorithms:
detection of termination is very hard!
→it provides the max « eventually »...
BUT, it does not tolerate Byzantine faults (SelfStabilization does for transient failures + acyclic topology)





Proactive Migration

- Principle: ~~predict failures and migrate processes~~ before failures
- Prediction models are based on the analysis of correlations between non fatal and fatal errors, and temporal and spatial correlations between failure events.
- Results on the ~~100 first days of BlueGene/L demonstrate good failure~~ proactive migration may help to significantly increase the checkpoint interval.
- Results are lacking concerning real time predictions and actual benefits of migration in real conditions
- Migration has a cost (need to checkpoint and log or delay messages)
- What to migrate?
 - Virtual Machine, Process checkpoint?
 - Only application state (user checkpoint)?



• WI Challenge: Analyze more traces, Identify more correlations, Improve predictive algorithms

Fault Recovery Options

- **Saved State**

- Restart – from checkpoint file
- Restart from **local** checkpoint
- Recalculate lost data from in-memory checkpoint (RAID like)

- **No Checkpoint**

- Lossy recalculation of lost data
- Recalculate lost data from initial and remaining data
- Replicate computation across system
- Reassign lost work to another resource
- Use natural fault tolerant algorithms





Fault Tolerance

Hard errors – permanent component failure either HW or SW (hung or crash)

Soft errors – transient errors, a blip or short term failure of either HW or SW

Silent errors – undetected errors either hard or soft, due to lack of detectors for a component or inability to detect (transient effect too short). **Real danger is that answer may be incorrect but the user wouldn't know.**

HW (node and interconnect) resilience needed to reduce Silent errors – Either turn them into Hard or Soft errors or fix them



Potential System Architecture

Systems	2009	2018
System peak	2 Pflop/s	1 Eflop/s
Power	6 MW	~20 MW
System memory	0.3 PB	32 - 64 PB
Node performance	125 GF	1,2 or 15TF
Node memory BW	25 GB/s	2-4TB/s
Node concurrency	12	O(1k) or 10k
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)
System size (nodes)	18,700	O(100,000) or O(1M)
Total concurrency	225,000	O(billion) [O(10) to O(100) for latency hiding]
Storage	15 PB	500-1000 PB (>10x system memory is min)
IO	0.2 TB	60 TB/s (how long to drain the machine)
MTTI	days	O(0.1 day)



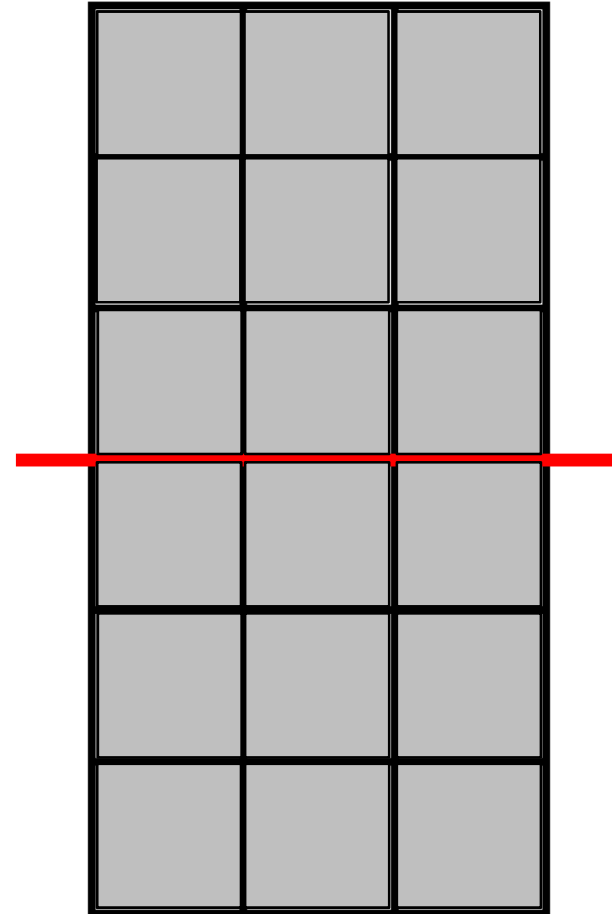
Factorization

TS matrix

- › $MT=6$ and $NT=3$
- › split into 2 domains

3 overlapped steps

- › panel factorization
- › updating the trailing submatrix
- › merge the domains





Factorization

TS matrix

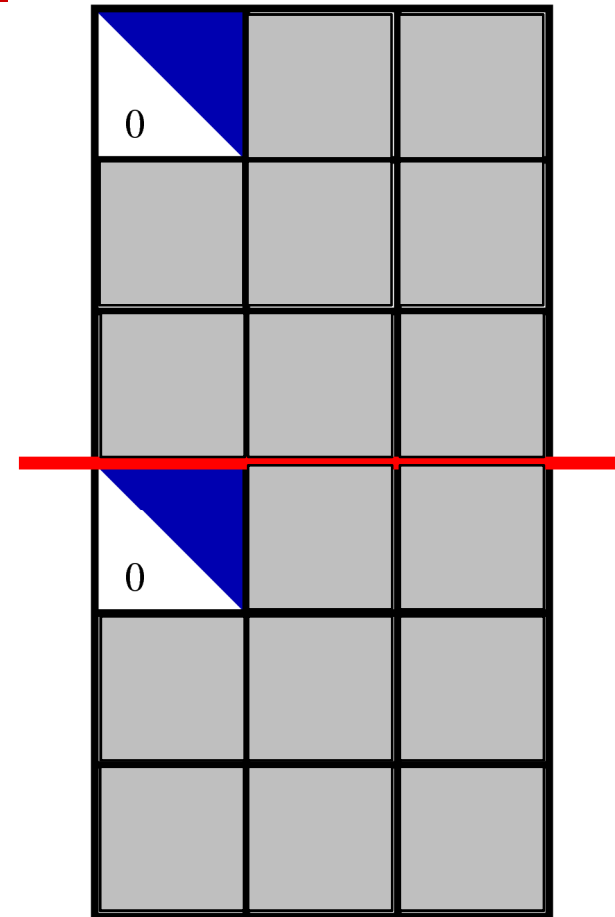
- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**

- › updating the trailing submatrix

- › merge the domains





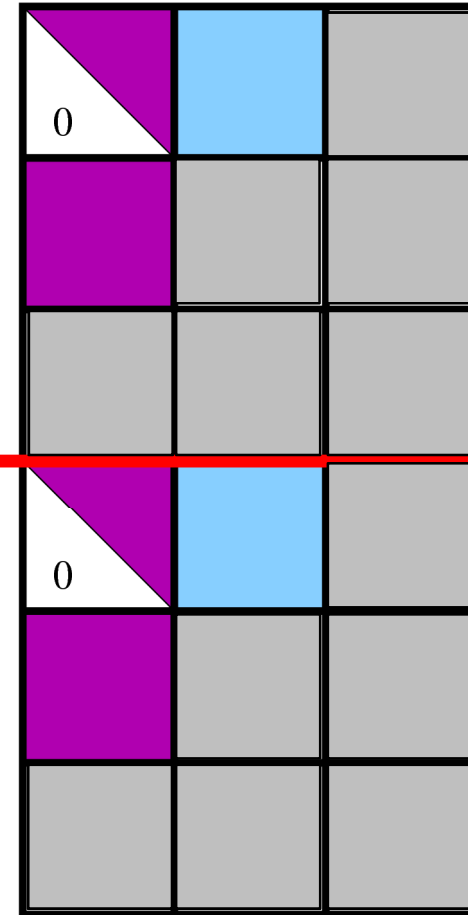
Factorization

TS matrix

- › $MT=6$ and $NT=3$
- › split into 2 domains

3 overlapped steps

- › **panel factorization**
- › **updating the trailing submatrix**
- › merge the domains





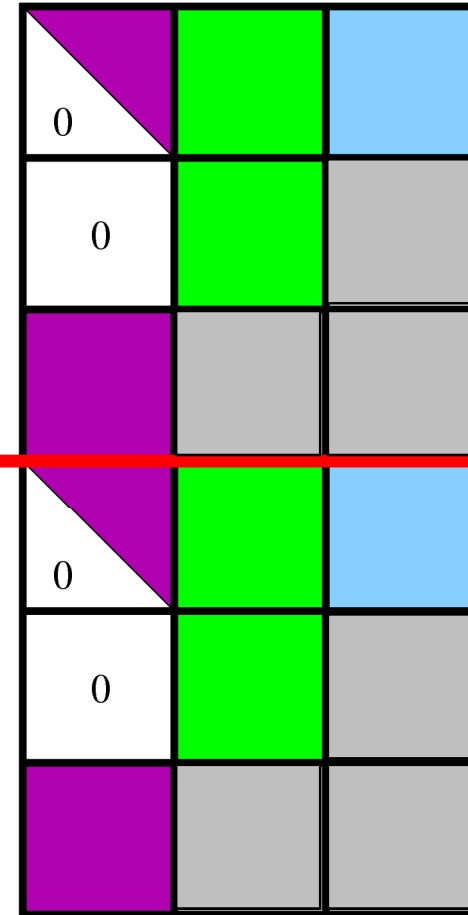
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**
- › **updating the trailing submatrix**
- › merge the domains





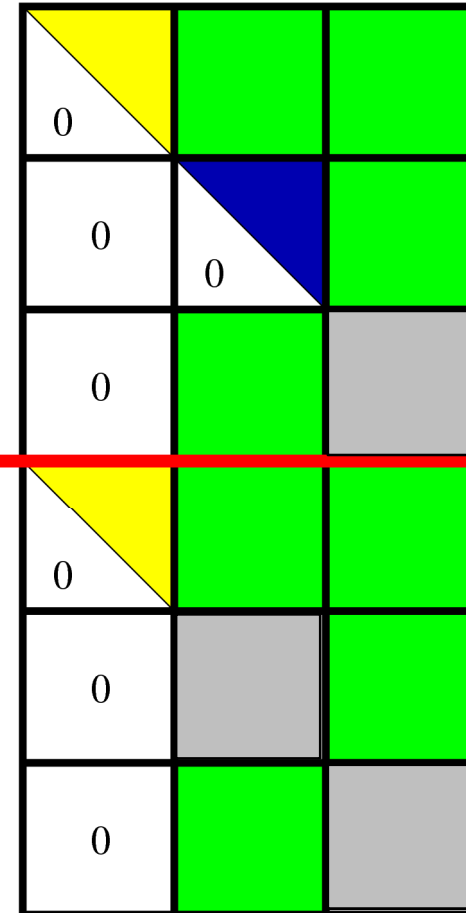
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**
- › **updating the trailing submatrix**
- › **merge the domains**





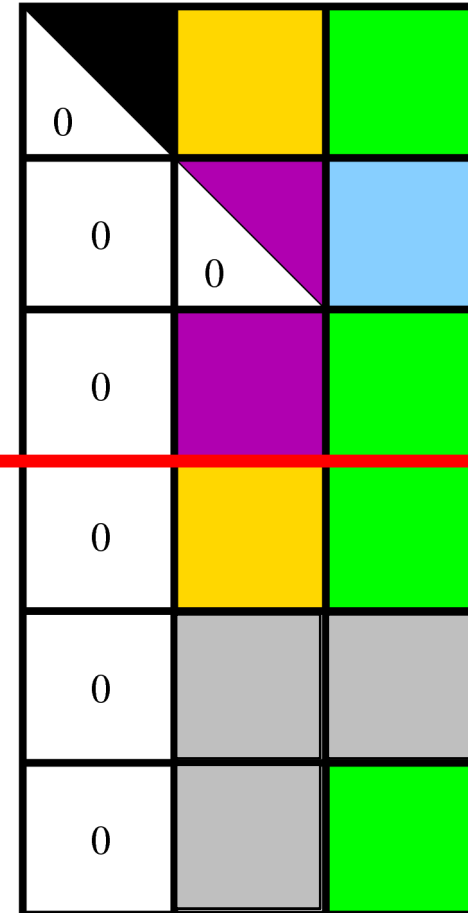
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**
- › **updating the trailing submatrix**
- › **merge the domains**





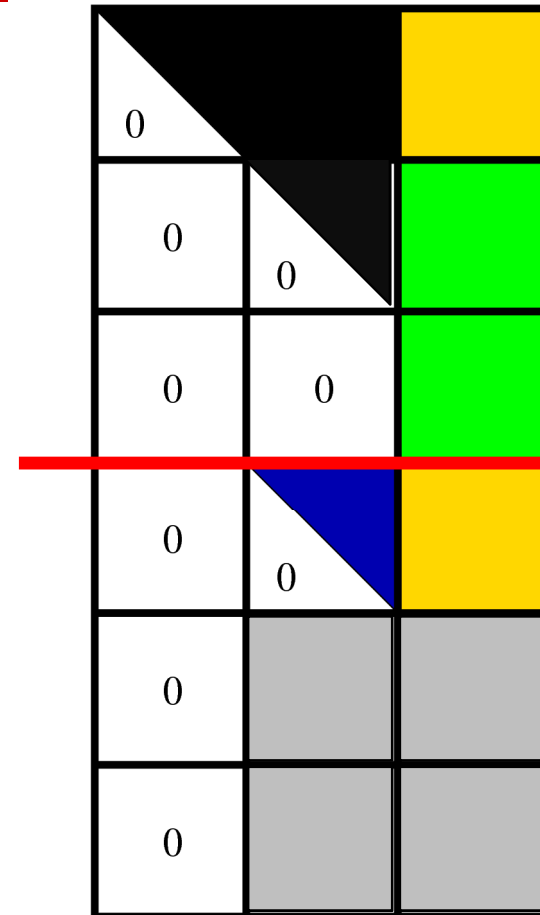
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**
- › **updating the trailing submatrix**
- › **merge the domains**





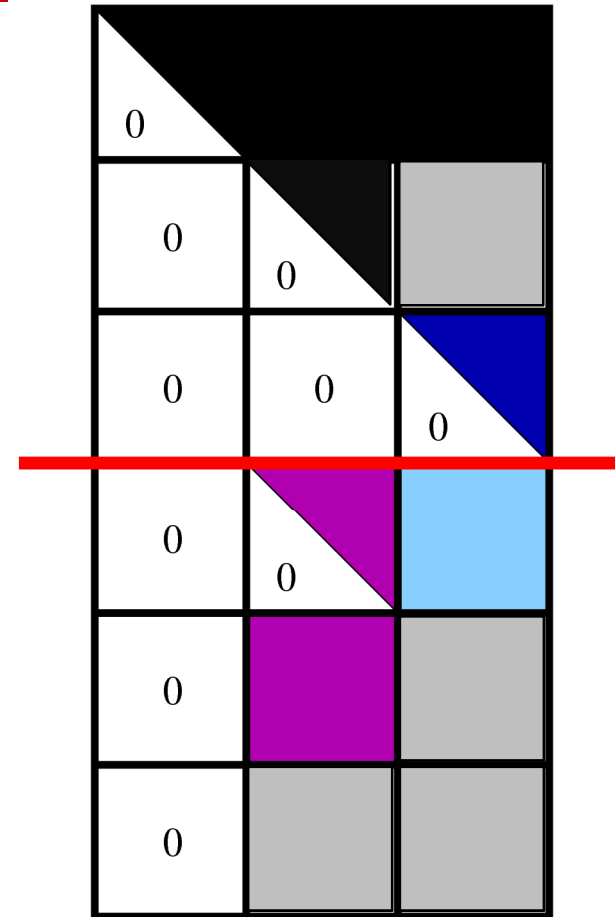
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**
- › **updating the trailing submatrix**
- › merge the domains





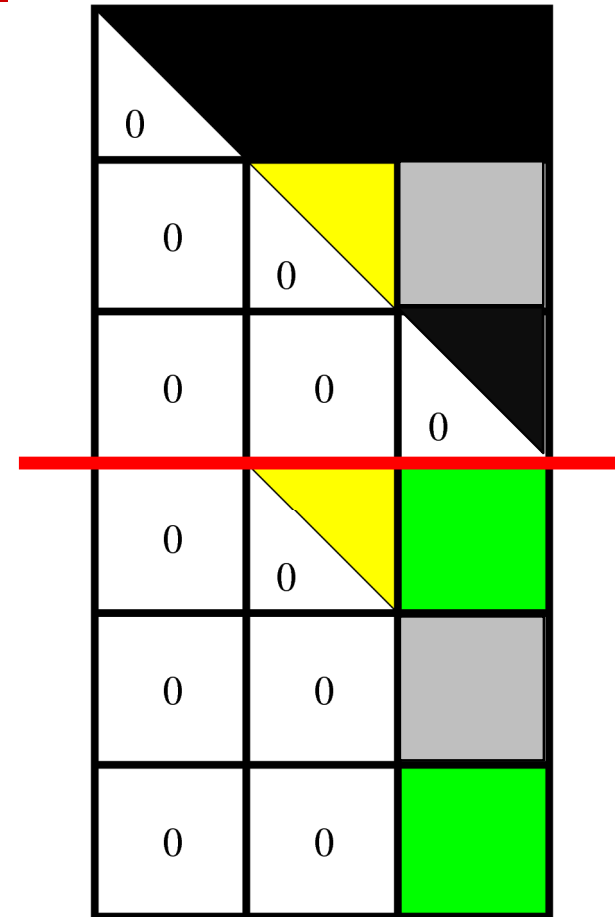
Factorization

TS matrix

- MT=6 and NT=3
- split into 2 domains

3 overlapped steps

- panel factorization
- **updating the trailing submatrix**
- **merge the domains**





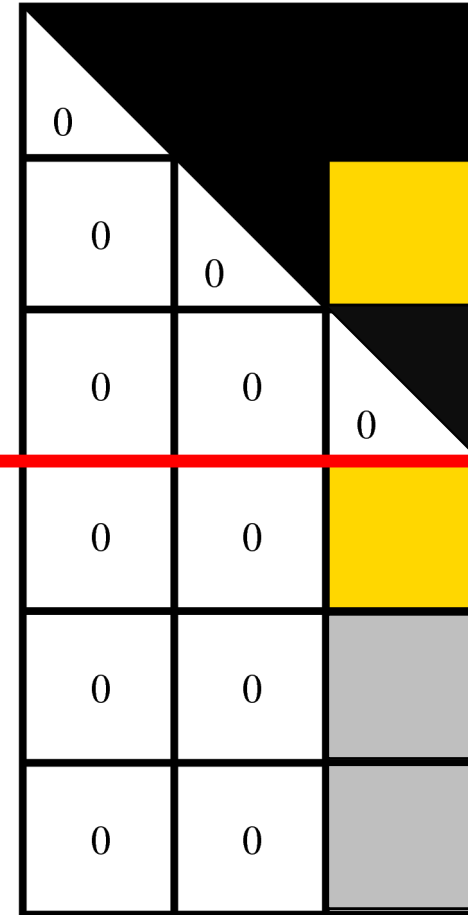
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › panel factorization
- › updating the trailing submatrix
- › **merge the domains**





Factorization

TS matrix

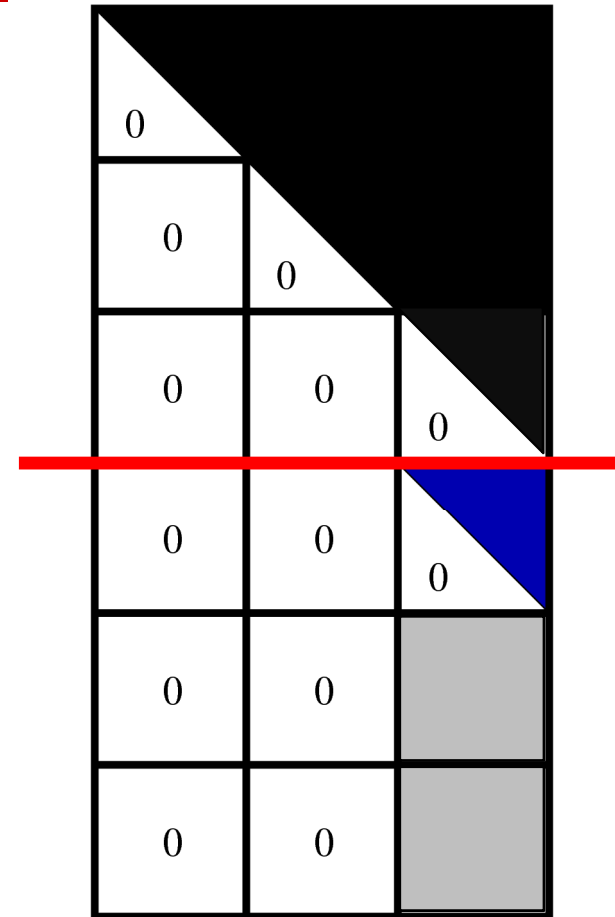
- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**

- › updating the trailing submatrix

- › merge the domains





Factorization

TS matrix

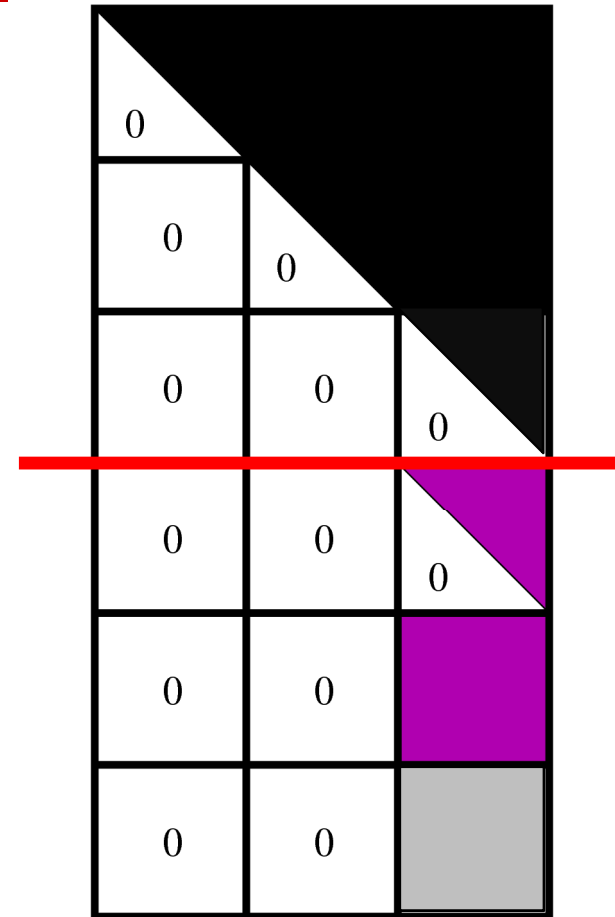
- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**

- › updating the trailing submatrix

- › merge the domains





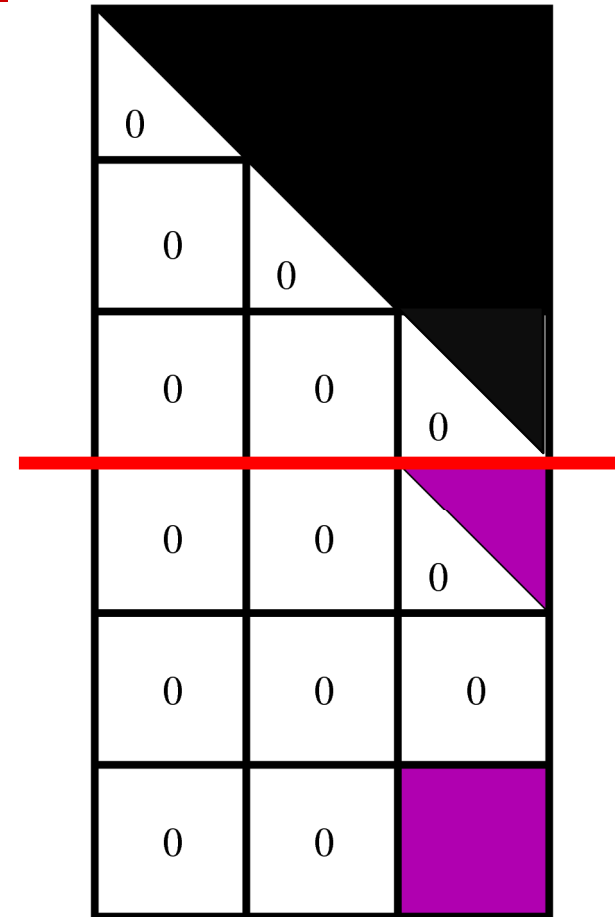
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › **panel factorization**
- › updating the trailing submatrix
- › merge the domains





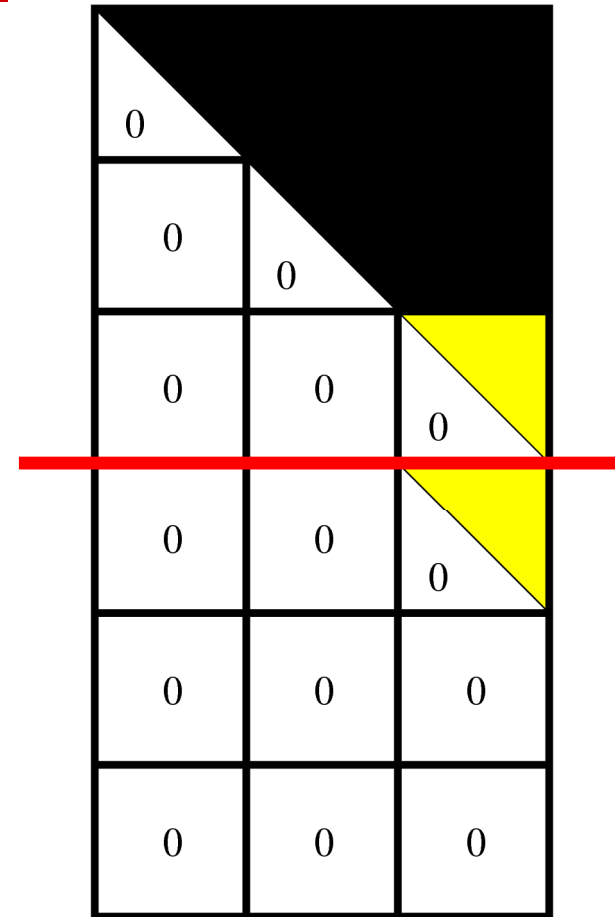
Factorization

TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › panel factorization
- › updating the trailing submatrix
- › **merge the domains**





Factorization

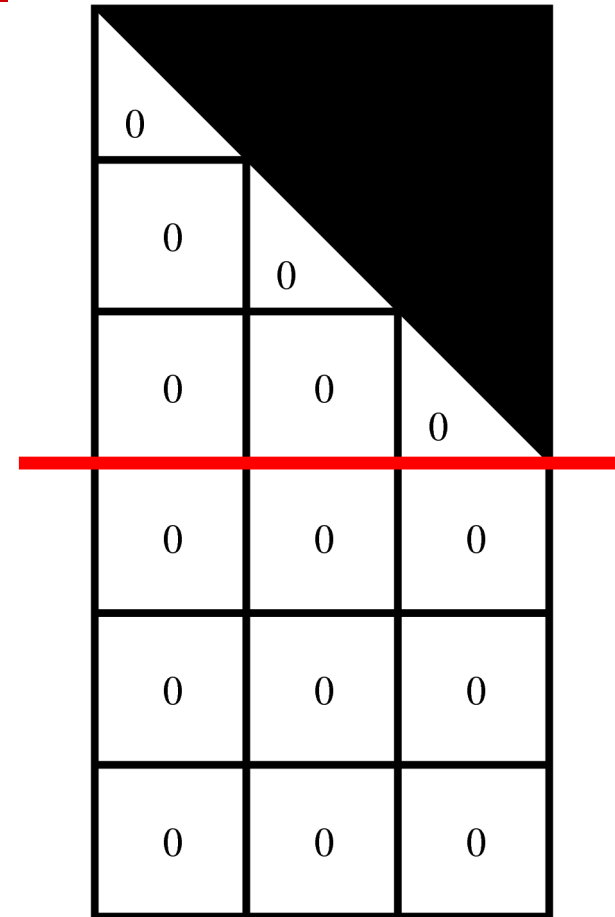
TS matrix

- › MT=6 and NT=3
- › split into 2 domains

3 overlapped steps

- › panel factorization
- › updating the trailing submatrix
- › merge the domains

- › **Final R computed**





35rd List: The TOP10

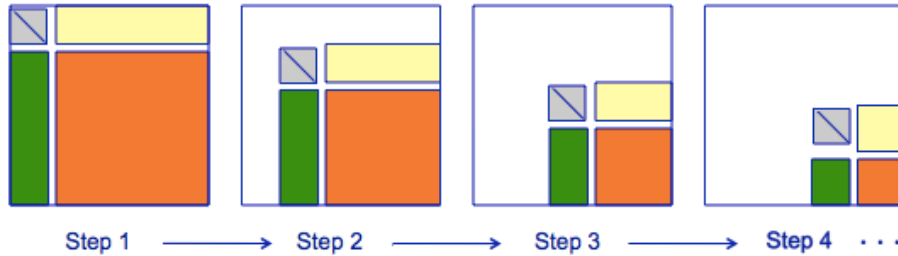
Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak
1	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5sixCore 2.6 GHz	USA	224,162	1.76	75
2	Nat. Supercomputer Center in Shenzhen	Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU	China	120,640	1.27	43
3	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenterQS22/LS21	USA	122,400	1.04	76
4	NSF / NICS / U of Tennessee	Kraken/ Cray Cray XT5sixCore 2.6 GHz	USA	98,928	.831	81
5	Forschungszentrum Julich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	.825	82
6	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	56,320	.544	82
7	National SC Center in Tianjin / NUDT	Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870	China	71,680	.563	46
8	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServerBlue Gene Solution	USA	212,992	.478	80
9	DOE / OS Argonne Nat Lab	Intrepid / IBM Blue Gene/P Solution	USA	163,840	.458	82
10	DOE / NNSA Sandia Nat Lab	Red Sky / Sun / SunBlade 6275	USA	42,440	.433	87



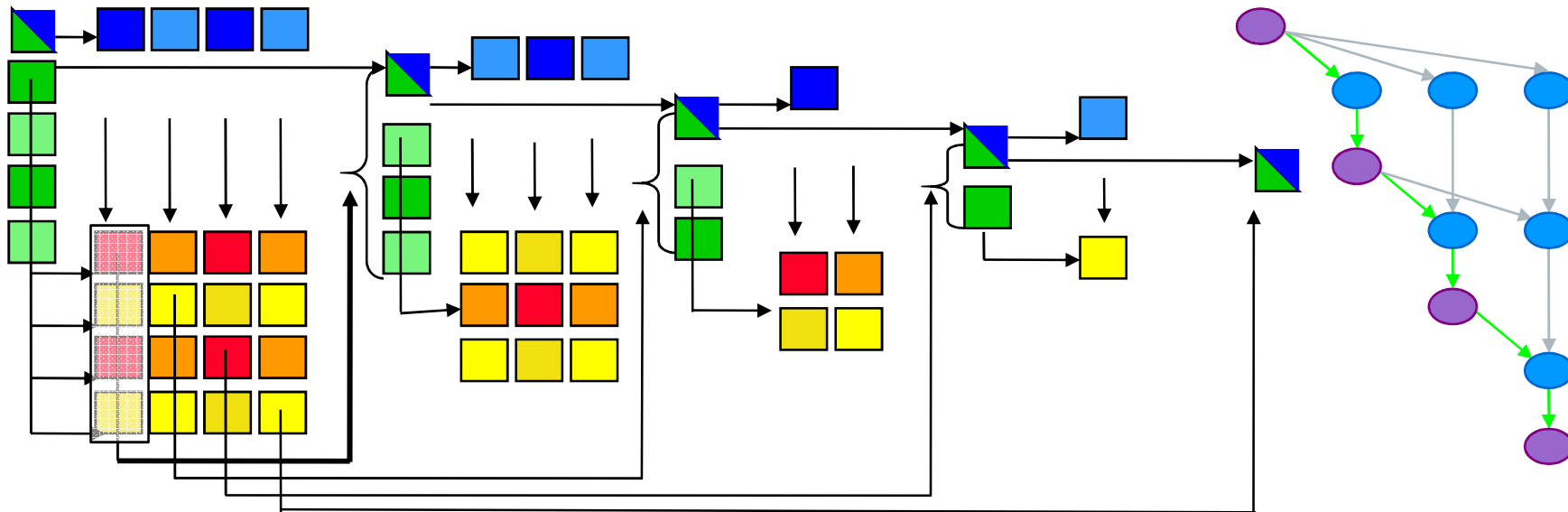
35rd List: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	MFlops /Watt
1	DOE / OS Oak Ridge Nat Lab	Jaguar / Cray Cray XT5sixCore 2.6 GHz	USA	224,162	1.76	75	7.0	251
2	Nat. Supercomputer Center in Shenzhen	Nebulea / Dawning / TC3600 Blade, Intel X5650, Nvidia C2050 GPU	China	120,640	1.27	43	2.58	493
3	DOE / NNSA Los Alamos Nat Lab	Roadrunner / IBM BladeCenterQS22/LS21	USA	122,400	1.04	76	2.48	446
4	NSF / NICS / U of Tennessee	Kraken/ Cray Cray XT5sixCore 2.6 GHz	USA	98,928	.831	81	3.09	269
5	ForschungszentrumJuelich (FZJ)	Jugene / IBM Blue Gene/P Solution	Germany	294,912	.825	82	2.26	365
6	NASA / Ames Research Center/NAS	Pleiades / SGI SGI Altix ICE 8200EX	USA	56,320	.544	82	3.1	175
7	National SC Center in Tianjin / NUDT	Tianhe-1 / NUDT TH-1 / IntelQC + AMD ATI Radeon 4870	China	71,680	.563	46	1.48	380
8	DOE / NNSA Lawrence Livermore NL	BlueGene/L IBM eServerBlue Gene Solution	USA	212,992	.478	80	2.32	206
9	DOE / OS Argonne Nat Lab	Intrepid / IBM Blue Gene/P Solution	USA	163,840	.458	82	1.26	363
10	DOE / NNSA Sandia Nat Lab	Red Sky / Sun / SunBlade 6275	USA	42,440	.433	87	2.4	180

Parallel Tasks in LU

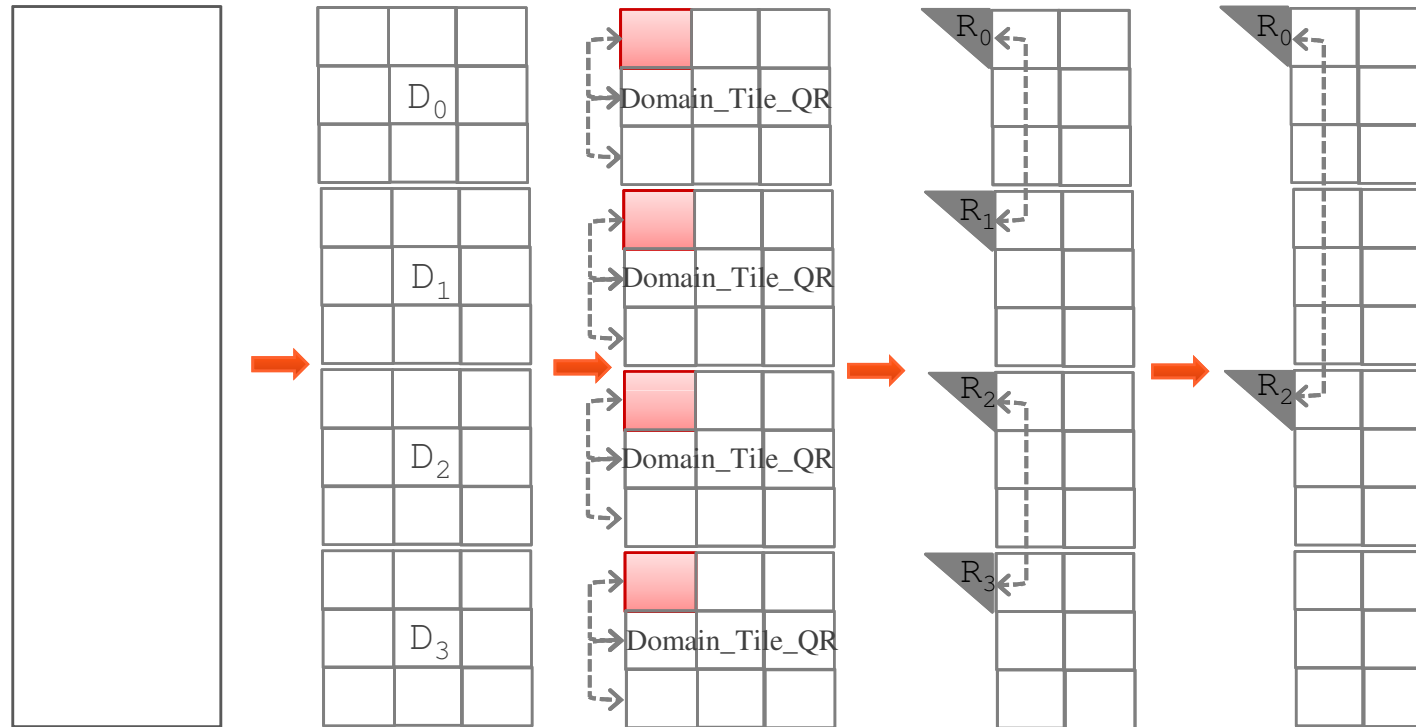


- Break into smaller tasks and remove dependencies





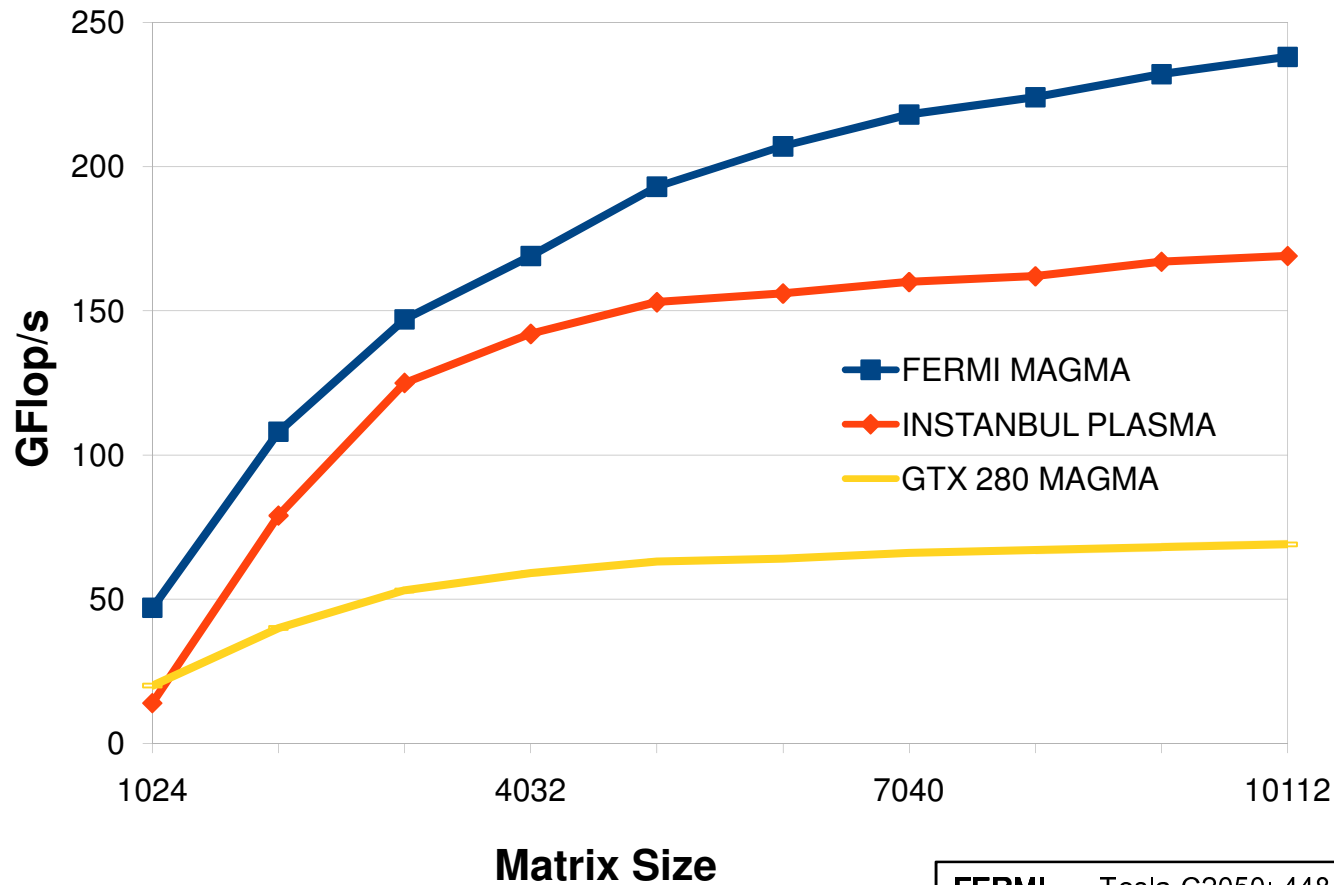
Communication Avoiding QR Example



A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications*, pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

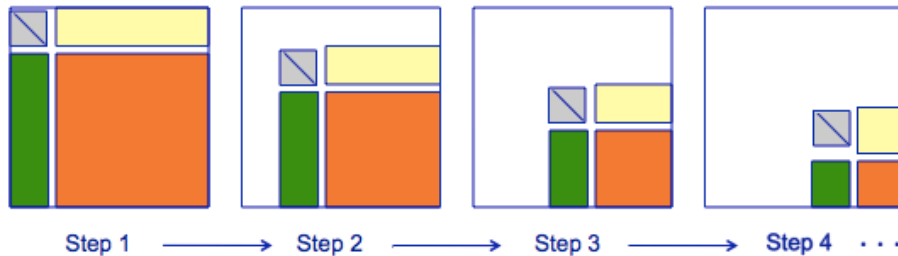


LU Factorization in Double Precision

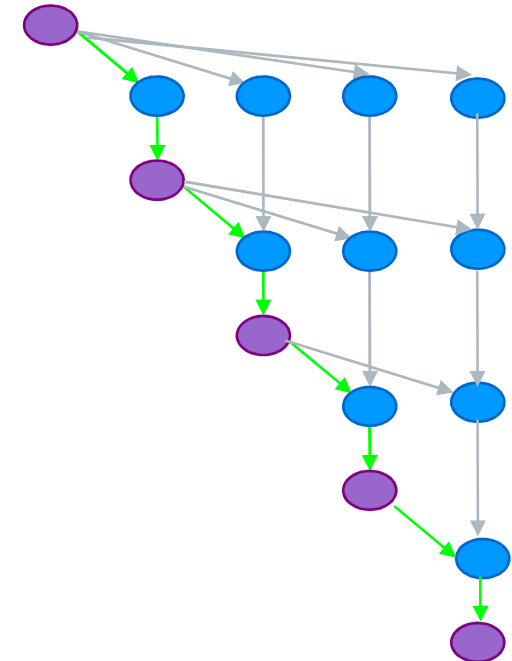


FERMI	Tesla C2050: 448 CUDA cores @ 1.15GHz SP/DP peak is 1030 / 515 GFlop/s
INSTANBUL	AMD 8 socket 6 core (48 cores) @2.8GHz SP/DP peak is 1075 / 538 GFlop/s

Parallel Tasks in LU/LL^T/QR



- Break into smaller tasks and remove dependencies
- Tile LU factorization on a square matrix with 5 x 5 tiles. Each tile is of size $b \times b$ and corresponds to a fine grain task. The arcs show the data dependencies between the tasks.



* LU does block pair wise pivoting



#3 LANL Roadrunner A Petascale System in 2008

“Connected Unit” cluster
192 Oteron nodes
(180 w/ 2 dual-Cell blades
connected w/ 4 PCIe x8

≈ 13,000 Cell HPC chips
≈ **1.33 PetaFlop/s** (from Cell)
≈ 7,000 dual-core Oterons
≈ **122,000 cores**

