On the Future of High Performance Computing: How to Think for Peta and Exascale Computing

Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

This picture was taken at Argonne around 1981

- Since then there have been tremendous changes in our scientific computing environment.
- Many changes in Mathematic Software and Numerical Libraries

This picture was taken at Argonne around 1981

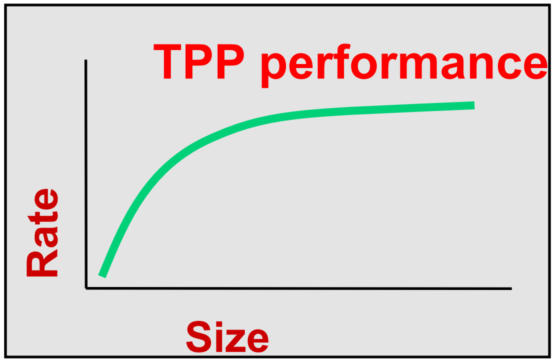
- Since then there have been tremendous changes in our scientific computing environment.
- Many changes in Mathematic Software and Numerical Libraries

Top500 List of Supercomputers

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

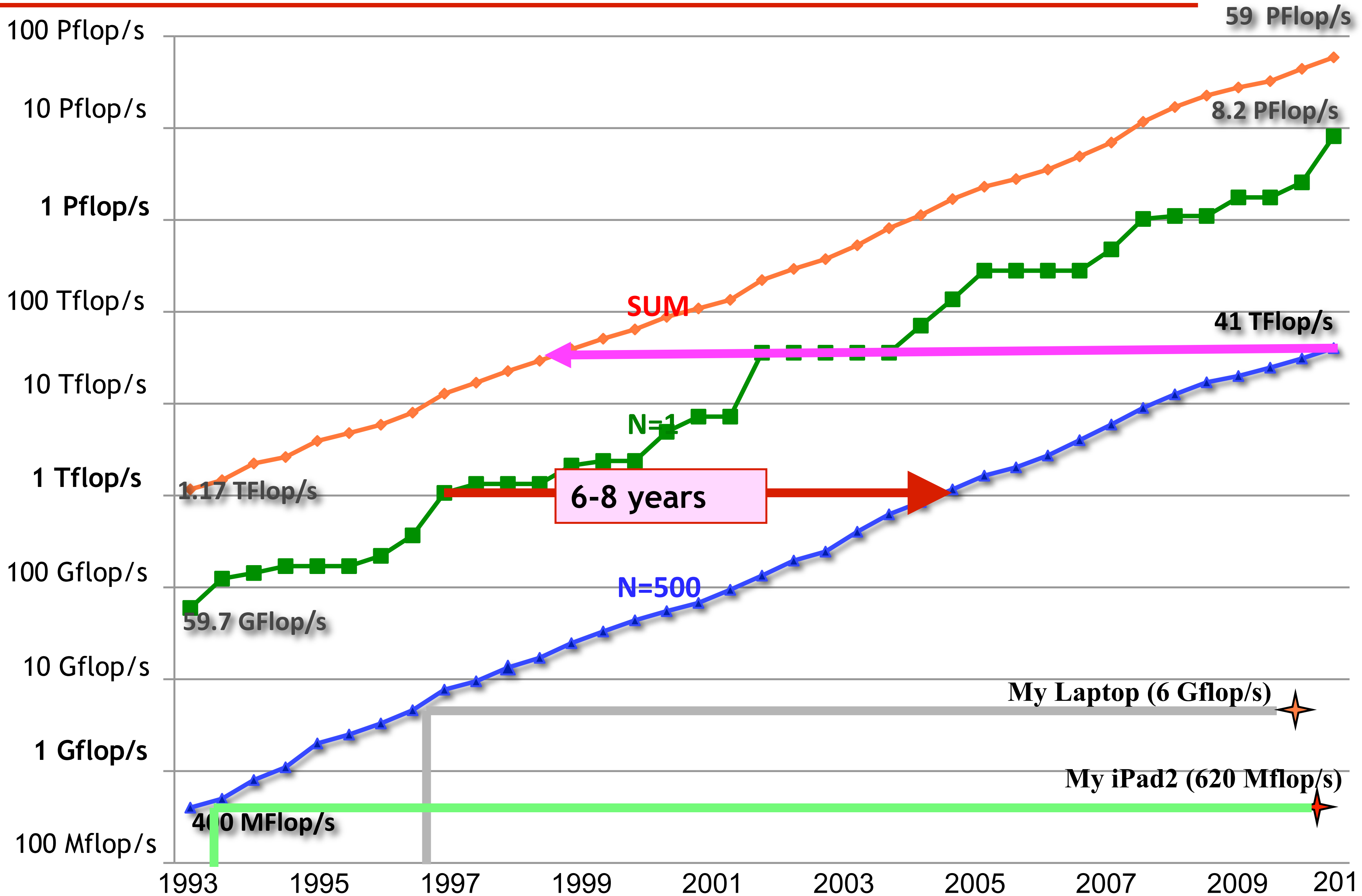
- Listing of the 500 most powerful Computers in the World
- Yardstick: Rmax from LINPACK MPP

$Ax = b$, *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



Emerging Computer Architectures

- Are needed by applications

- Applications are given (as function of time)
- Architectures are given (as function of time)

- Algorithms and software must be adapted or created to bridge to computer architectures for the sake of the complex applications

Three Design Points Today

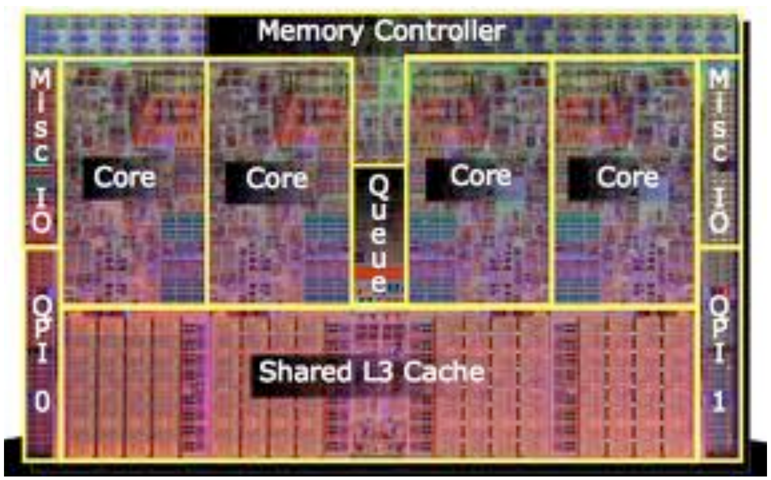
- Gigascale Laptop: Uninode-Multicore

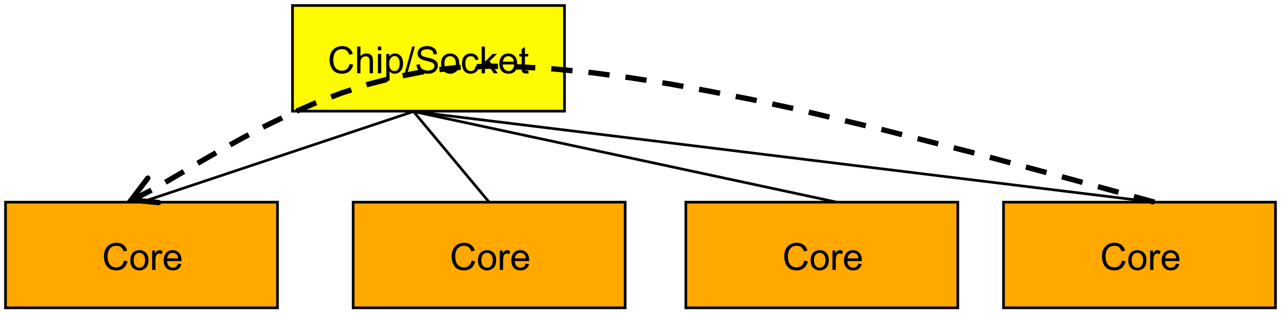
  (Your iPhone and iPad are Mflop/s devices)

- Terascale Deskside: Multinode-Multicore

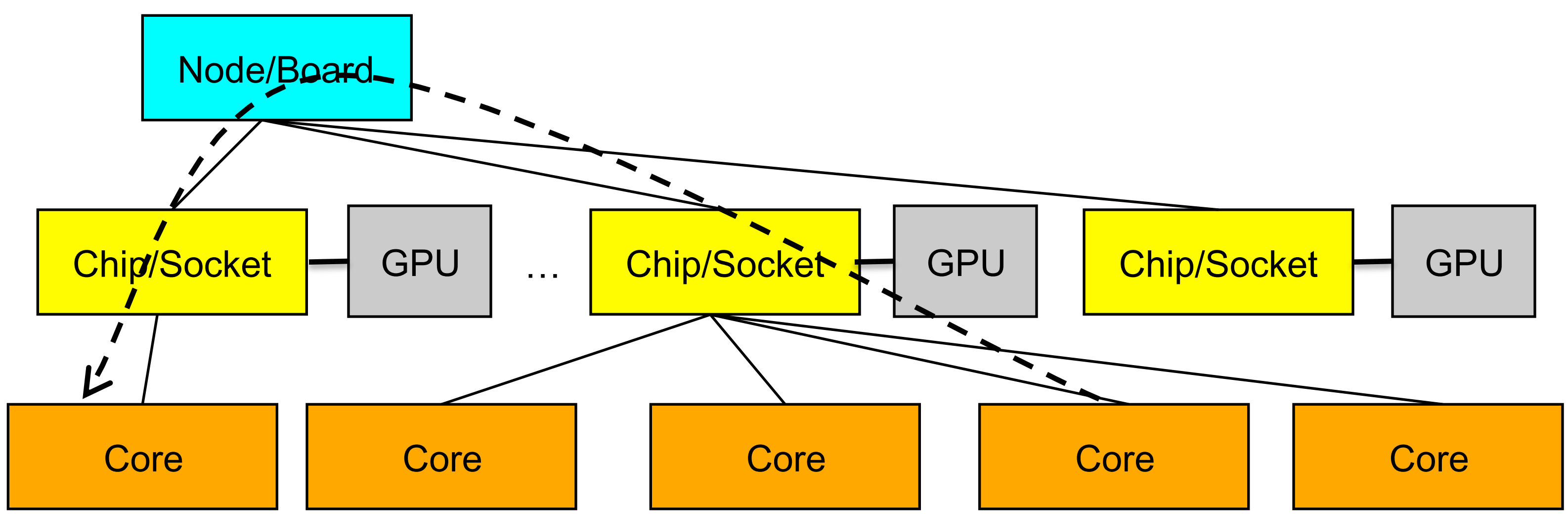
- Petacale Center: Multinode-Multicore

Example of typical parallel machine



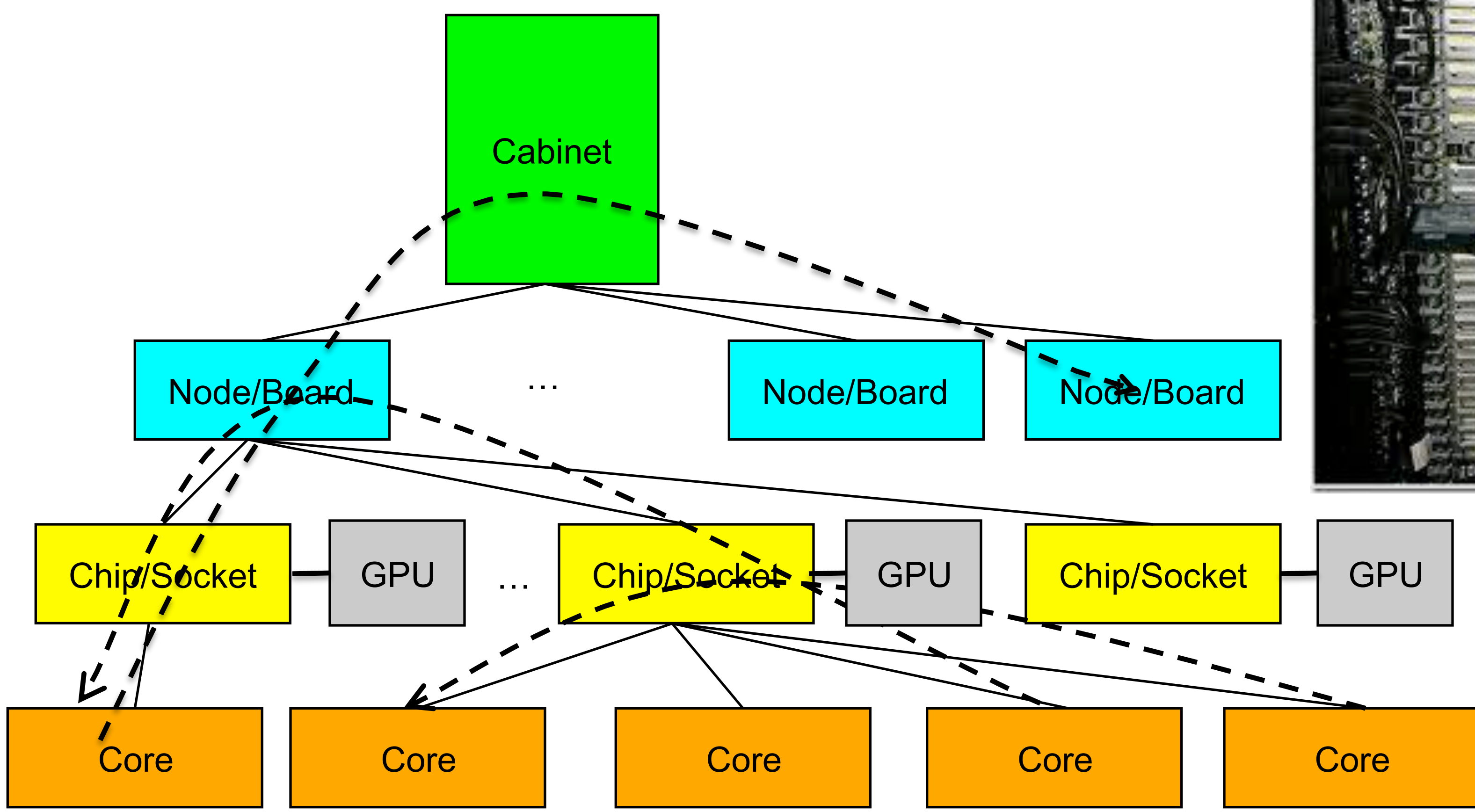


Example of typical parallel machine

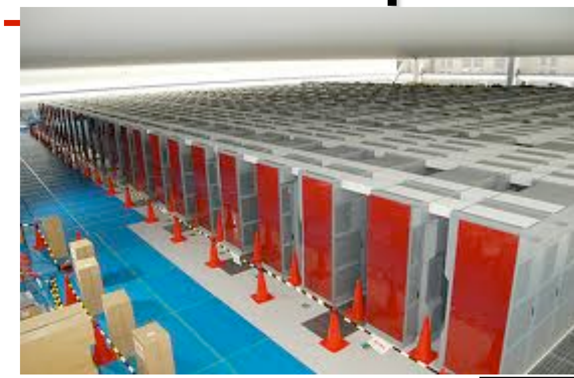


Example of typical parallel machine

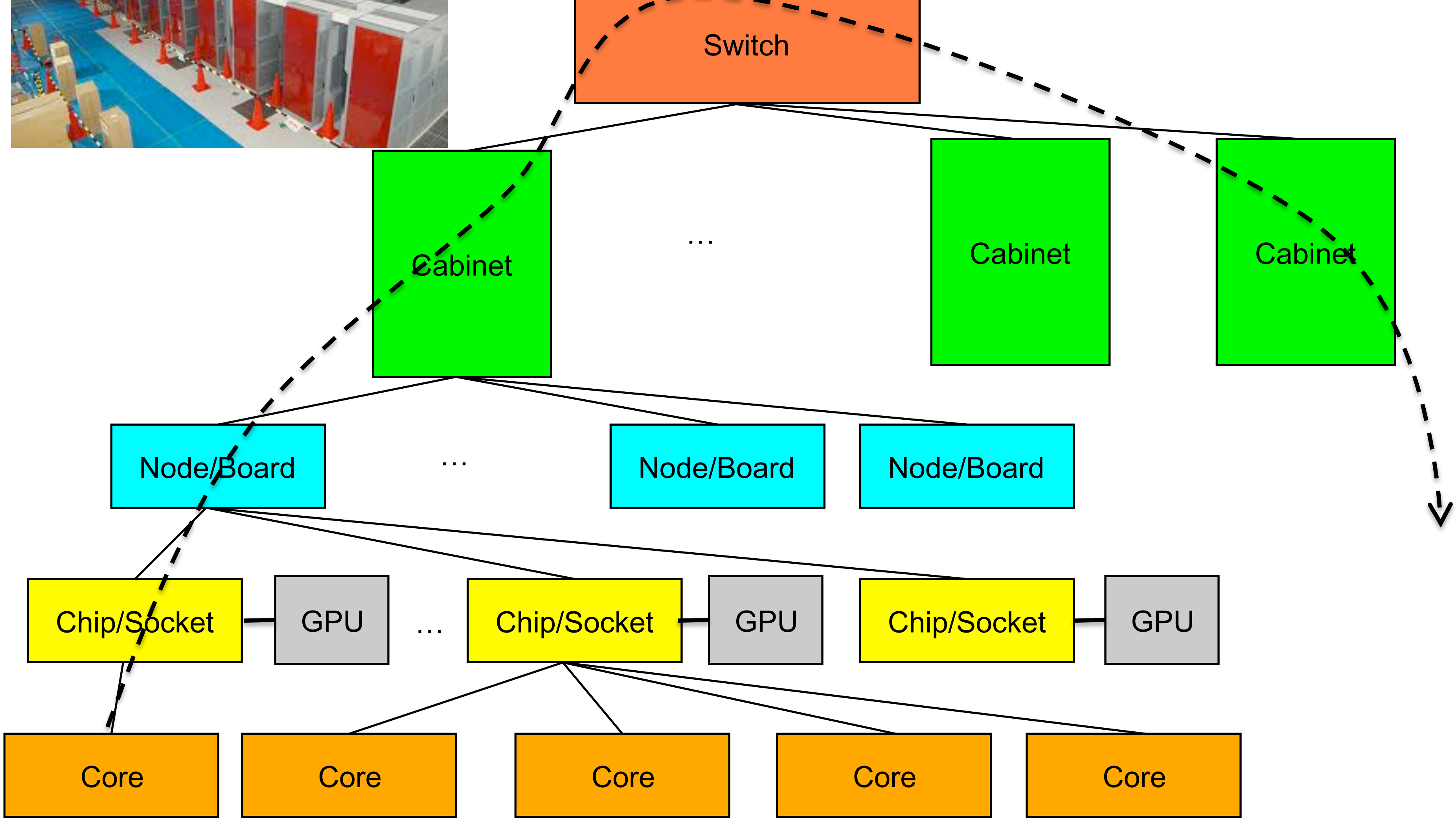
Shared memory programming between processes on a board and
a combination of shared memory and distributed memory programming
between nodes and cabinets



Example of typical parallel machine



Combination of shared memory and distributed memory programming



June 2011: The TOP10

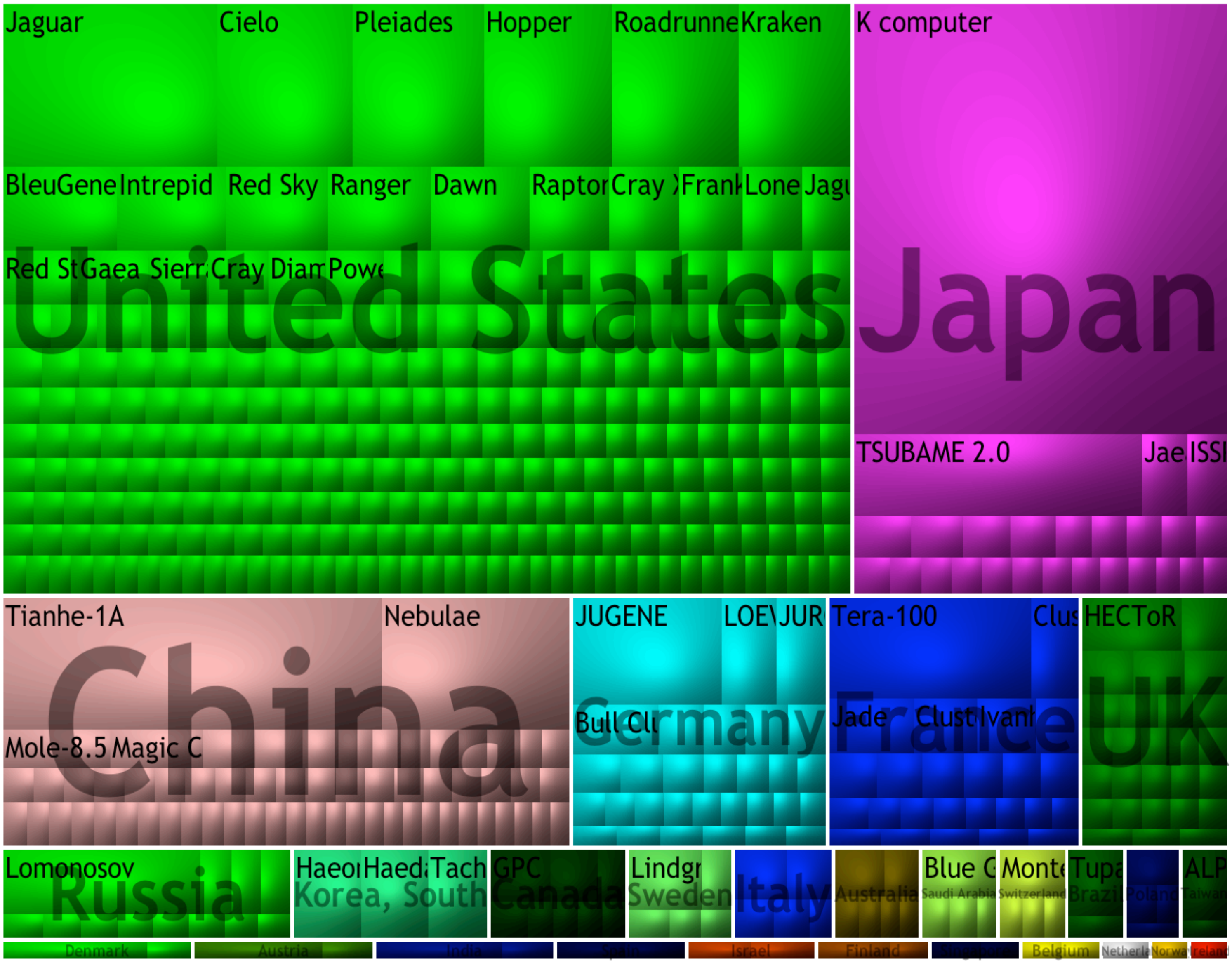
Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak
1	RIKEN Advanced Inst for Comp Sci	K Computer Fujitsu SPARC64 VIIIfx + custom	Japan	548,352	8.16	93
2	Nat. SuperComputer Center in Tianjin	Tianhe-1A, NUDT Intel + Nvidia GPU + custom	China	186,368	2.57	55
3	DOE / OS Oak Ridge Nat Lab	Jaguar, Cray AMD + custom	USA	224,162	1.76	75
4	Nat. Supercomputer Center in Shenzhen	Nebulea, Dawning Intel + Nvidia GPU + IB	China	120,640	1.27	43
5	GSIC Center, Tokyo Institute of Technology	Tusbame 2.0, HP Intel + Nvidia GPU + IB	Japan	73,278	1.19	52
6	DOE / NNSA LANL & SNL	Cielo, Cray AMD + custom	USA	142,272	1.11	81
7	NASA Ames Research Center/NAS	Plelades SGI Altix ICE 8200EX/8400EX + IB	USA	111,104	1.09	83
8	DOE / OS Lawrence Berkeley Nat Lab	Hopper, Cray AMD + custom	USA	153,408	1.054	82
9	Commissariat a l'Energie Atomique (CEA)	Tera-10, Bull Intel + IB	France	138,368	1.050	84
10	DOE / NNSA Los Alamos Nat Lab	Roadrunner, IBM AMD + Cell GPU + IB	USA	122,400	1.04	76

June 2011: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	GFlops/ Watt
1	RIKEN Advanced Inst for Comp Sci	K Computer Fujitsu SPARC64 VIIIfx + custom	Japan	548,352	8.16	93	9.9	824
2	Nat. SuperComputer Center in Tianjin	Tianhe-1A, NUDT Intel + Nvidia GPU + custom	China	186,368	2.57	55	4.04	636
3	DOE / OS Oak Ridge Nat Lab	Jaguar, Cray AMD + custom	USA	224,162	1.76	75	7.0	251
4	Nat. Supercomputer Center in Shenzhen	Nebulea, Dawning Intel + Nvidia GPU + IB	China	[illegible]	[illegible].27	[illegible]	2.58	493
[illegible]	GSIC Center, Tokyo Institute of Technology	Tusbame 2.0, HP Intel + Nvidia GPU + IB	Japan	[illegible]	[illegible].19	[illegible]	[illegible]	[illegible]
6	DOE / NNSA LANL & SNL	Cielo, Cray AMD + custom	USA	142,272	1.11	81	3.98	279
7	NASA Ames Research Center/NAS	Plelades SGI Altix ICE 8200EX/8400EX + IB	USA	111,104	1.09	83	4.10	265
8	DOE / OS Lawrence Berkeley Nat Lab	Hopper, Cray AMD + custom	USA	153,408	1.054	82	2.91	362
9	Commissariat a l'Energie Atomique (CEA)	Tera-10, Bull Intel + IB	France	138,368	1.050	84	4.59	229
10	DOE / NNSA Los Alamos Nat Lab	Roadrunner, IBM AMD + Cell GPU + IB	USA	122,400	1.04	76	2.35	446
500	Energy Comp	IBM Cluster, Intel + GigE	China	7,104	.041	53		

Quiz: How Many of the Top500 systems use GPUs?

Countries Share



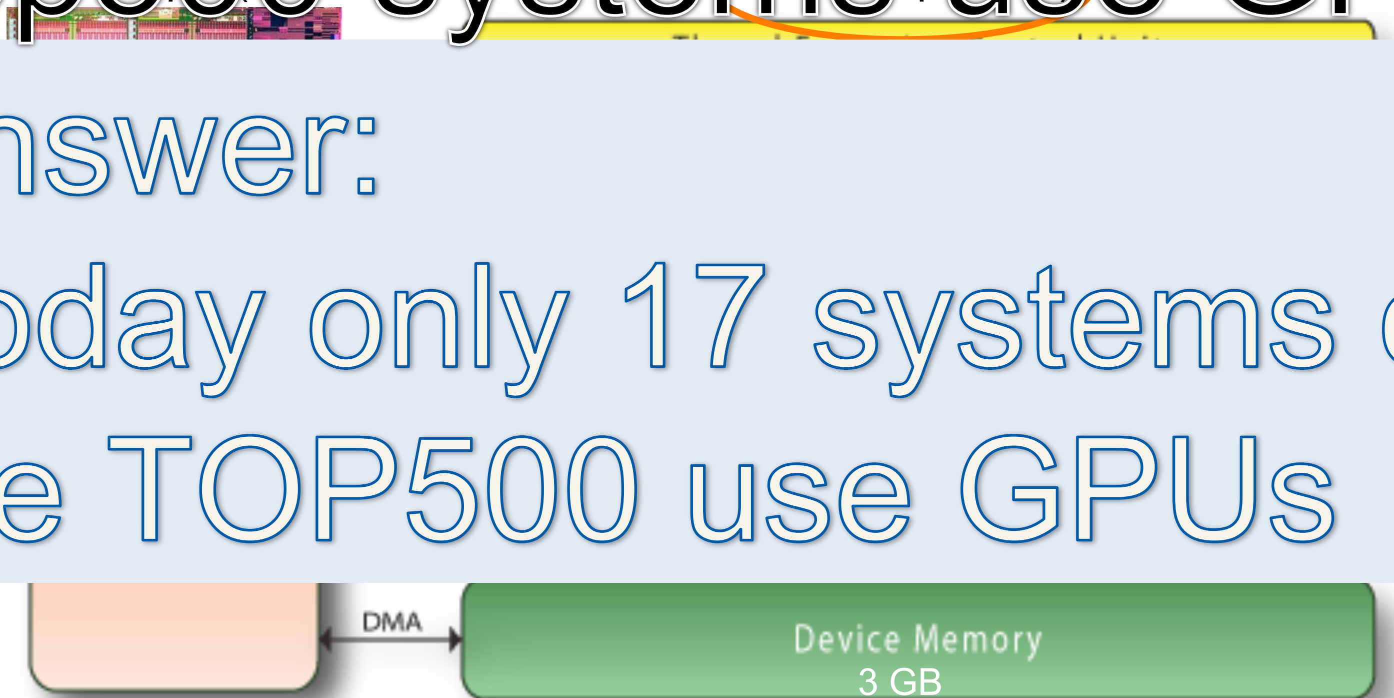
Absolute Counts

US:	251
China:	64
Germany:	31
UK:	28
Japan:	26
France:	25

Commodity plus Accelerator

Commodity	Accelerator (GPU)
Intel Xeon	Nvidia C2050 "Fermi"
8 cores	448 "Cuda cores"
3 GHz	1.15 GHz
8*4 ops/cycle	448 ops/cycle
96 Gflop/s (DP)	515 Gflop/s (DP)

Quiz: How Many of the Top500 systems use GPUs?



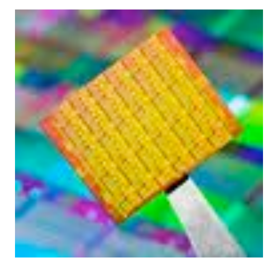
Answer:
Today only 17 systems on the TOP500 use GPUs

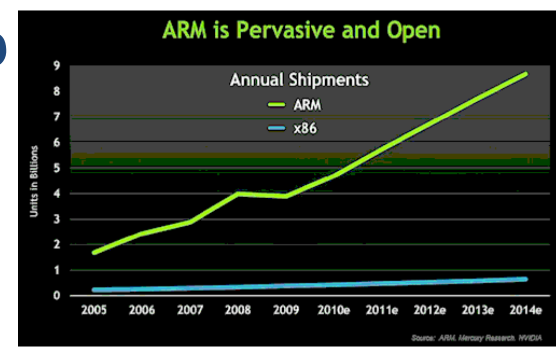
Interconnect
PCI-X 16 lane
64 Gb/s
1 GW/s

Future Computer Systems

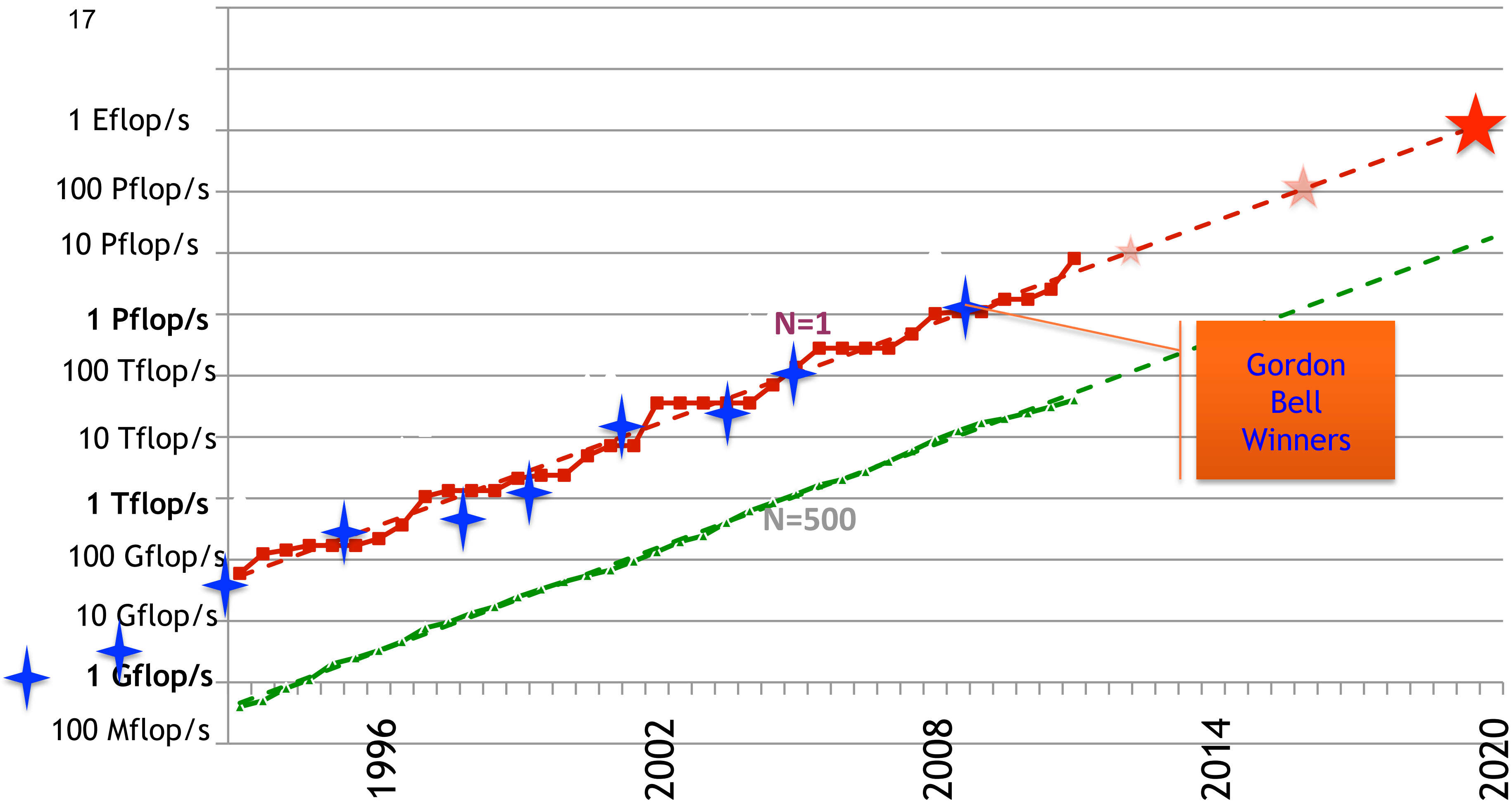
- Most likely be a hybrid design
  - Think standard multicore chips and accelerator (GPUs)
- Today accelerators are attached
- Next generation more integrated
- Intel's MIC architecture "Knights Ferry" and "Knights Corner" to come.
  - 48 x86 cores
- AMD's Fusion in 2012 - 2013
  - Multicore with embedded graphics ATI
- Nvidia's Project Denver plans to develop an integrated chip using ARM architecture in 2013.

The future is fusion





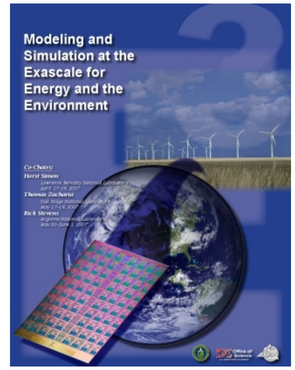
Performance Development in Top500

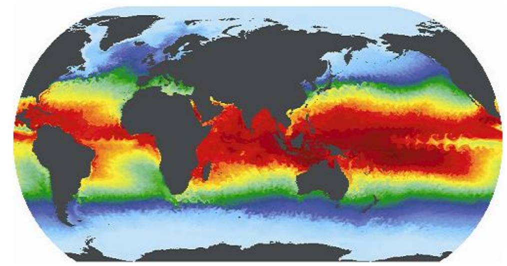


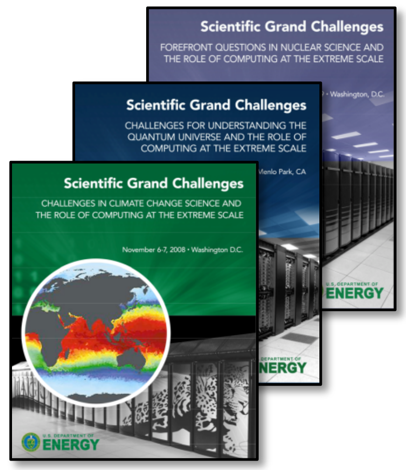
Broad Community Support and Development of the Exascale Initiative Since 2007

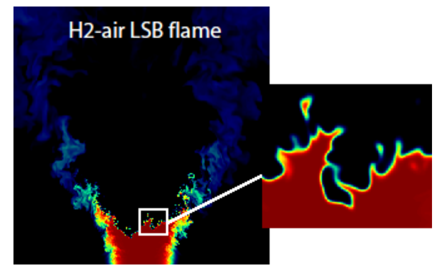
http://science.energy.gov/ascr/news-and-resources/program-documents/

- **Town Hall Meetings April-June 2007**
- **Scientific Grand Challenges Workshops Nov, 2008 – Oct, 2009**
  - **Climate Science (11/08)**
  - **High Energy Physics (12/08)**
  - **Nuclear Physics (1/09)**
  - **Fusion Energy (3/09)**
  - **Nuclear Energy (5/09)**
  - **Biology (8/09)**
  - **Material Science and Chemistry (8/09)**
  - **National Security (10/09)**
  - **Cross-cutting technologies (2/10)**
- **Exascale Steering Committee**
  - **“Denver” vendor NDA visits (8/09)**
  - **SC09 vendor feedback meetings**
  - **Extreme Architecture and Technology Workshop (12/09)**
- **International Exascale Software Project**
  - **Santa Fe, NM (4/09); Paris, France (6/09); Tsukuba, Japan (10/09); Oxford (4/10); Maui (10/10); San Francisco (4/11)**

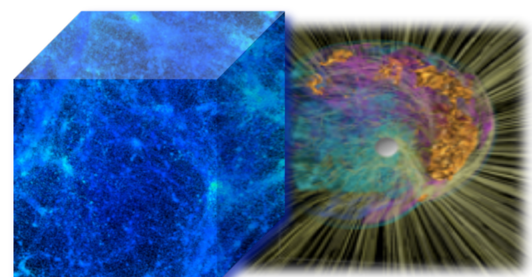








Mission Imperatives



Fundamental Science

Potential System Architecture

Systems	2011 K Computer
System peak	**8.7 Pflop/s**
Power	**10 MW**
System memory	1.6 PB
Node performance	128 GF
Node memory BW	64 GB/s
Node concurrency	8
Total Node Interconnect BW	20 GB/s
System size (nodes)	68,544
Total concurrency	548,352
MTTI	days

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

Systems	2011 K Computer	2019	Difference Today & 2019
System peak	**8.7 Pflop/s**	**1 Eflop/s**	**O(100)**
Power	**10 MW**	**~20 MW**	
System memory	1.6 PB	32 - 64 PB	O(10)
Node performance	128 GF	1,2 or 15TF	O(10) – O(100)
Node memory BW	64 GB/s	2 - 4TB/s	O(100)
Node concurrency	8	O(1k) or 10k	O(100) – O(1000)
Total Node Interconnect BW	20 GB/s	200-400GB/s	O(10)
System size (nodes)	68,544	O(100,000) or O(1M)	O(10) – O(100)
Total concurrency	548,352	O(billion)	O(1,000)
MTTI	days	O(1 day)	- O(10)

Three Design Points for Tomorrow

- Terascale Laptop:
  - ➢ Manycore

- Petascale Deskside:
  - ➢ Manynode-Manycore

- Exacale Center:
  - ➢ Manynode-Manycore

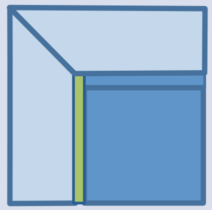
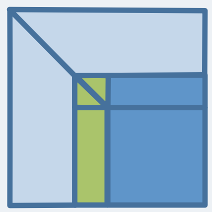
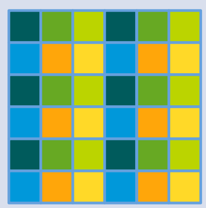
Major Changes to Software & Algorithms

- Must rethink the design of our algorithms and software
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software
  - Data movement is expense
  - Flop/s are cheap, so are provisioned in excess

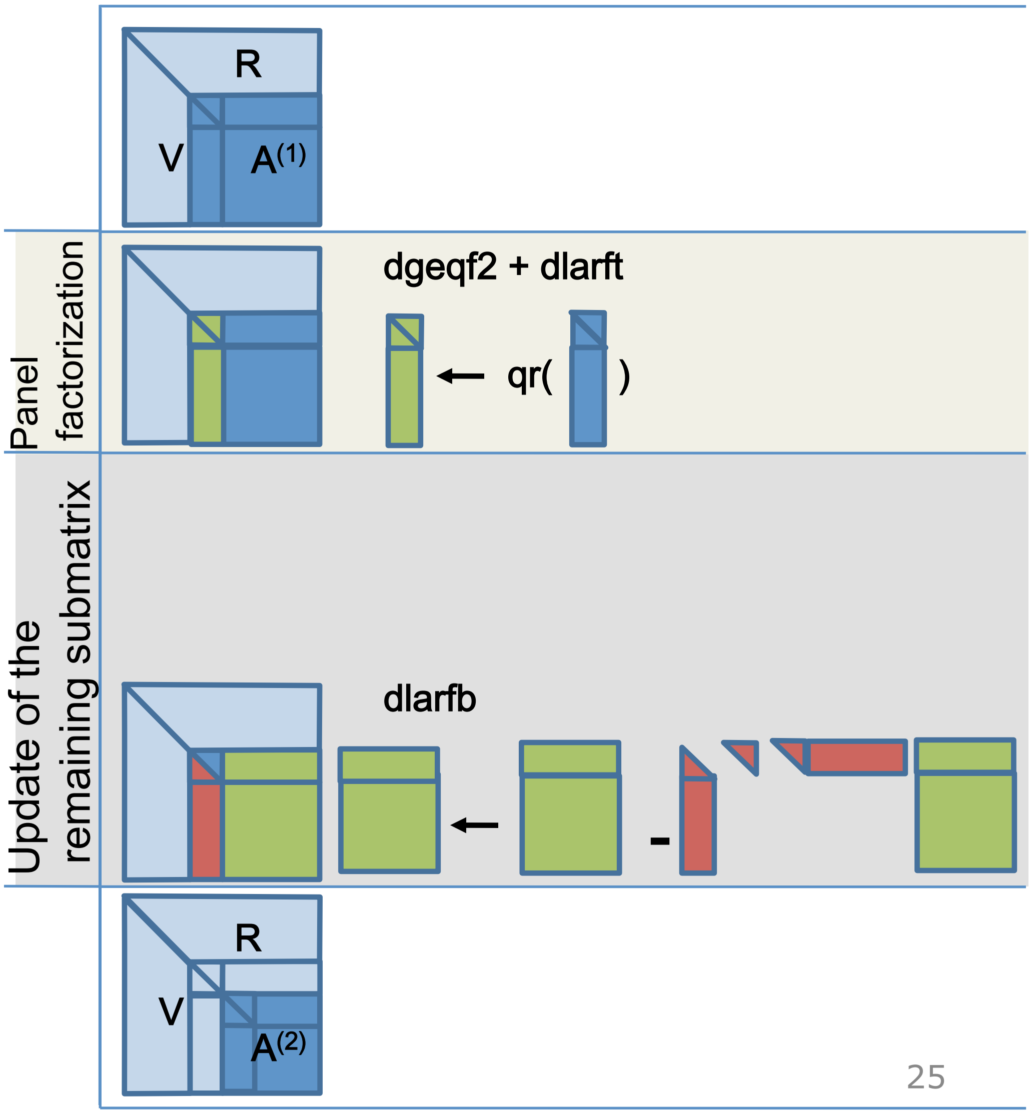
Critical Issues at Peta & Exascale for Algorithm and Software Design

- Synchronization-reducing algorithms
  - Break Fork-Join model
- Communication-reducing algorithms
  - Use methods which have lower bound on communication
- Mixed precision methods
  - 2x speed of ops and 2x speed for data movement
- Autotuning
  - Today's machines are too complicated, build "smarts" into software have experiment to optimize.
- Fault resilient algorithms
  - Implement algorithms that can recover from failures/bit flips
- Reproducibility of results
  - Today we can't guarantee this. We understand the issues, but some of our "colleagues" have a hard time with this.

Do you remember the 80’s and 90’s?

Algorithms follow hardware evolution along time.		
LINPACK (80’s) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (90’s) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (00’s) (Distributed memory, Message passing)		Rely on -Level-3 BLAS operations - MPI for message passing

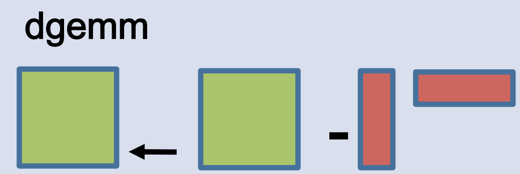
Blocked QR Factorization (LAPACK)

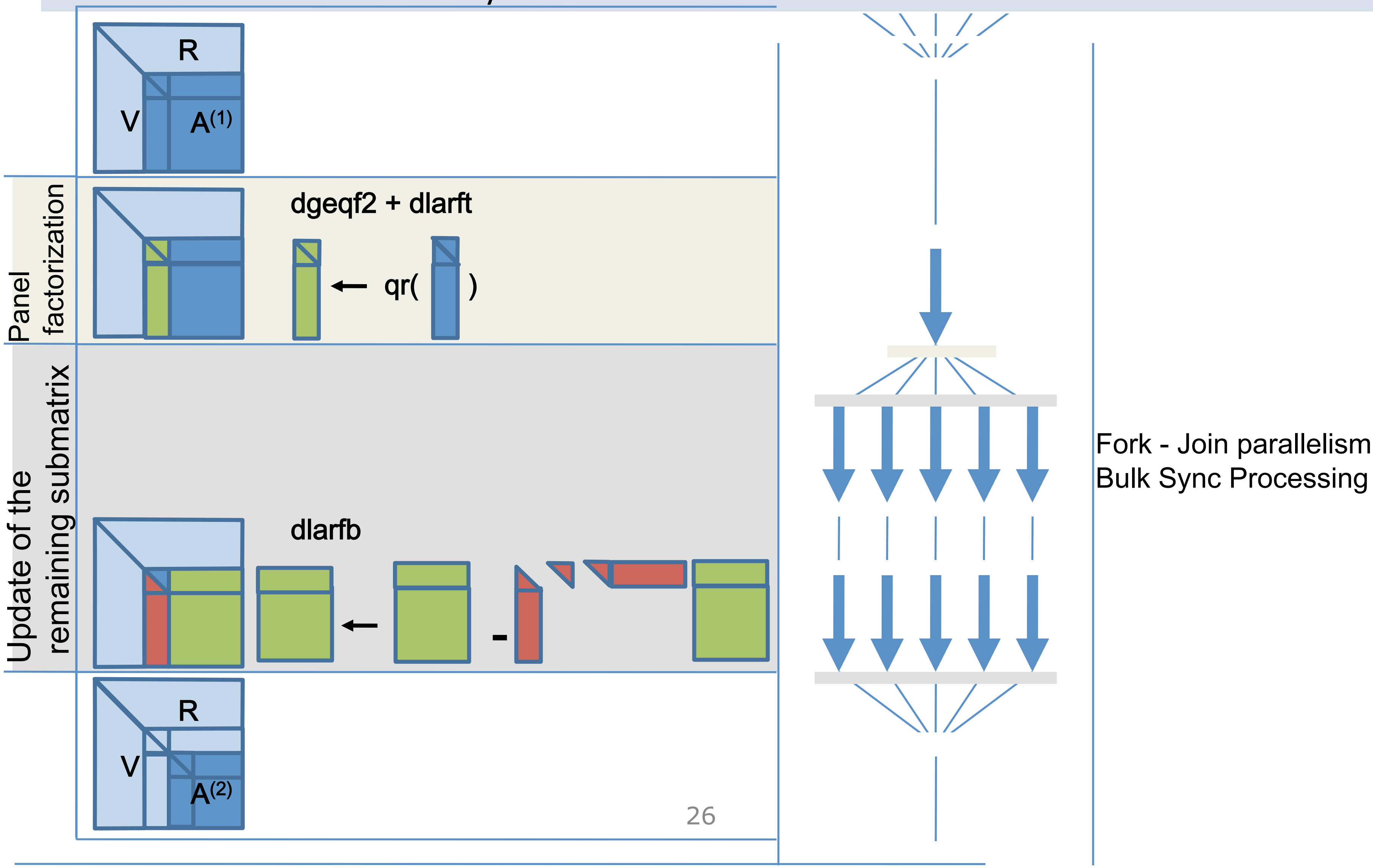


Parallelization of QR Factorization

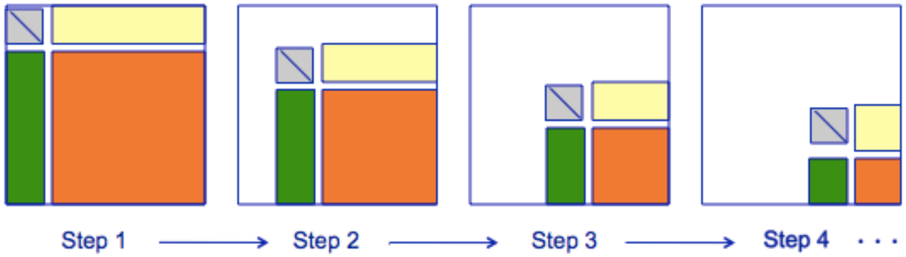
Parallelize the update:

- Easy and done in any reasonable software.
- This is the $2/3n^3$ term in the FLOPs count.
- Can be done "efficiently" with LAPACK+multithreaded BLAS

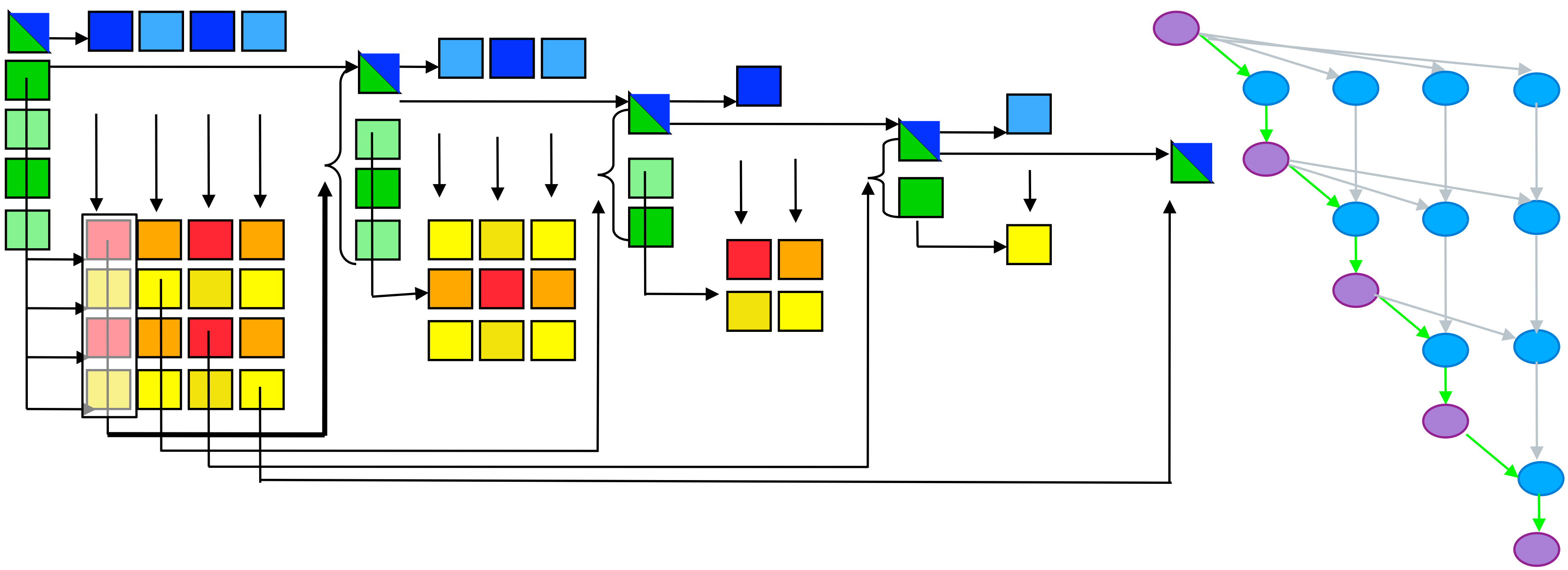




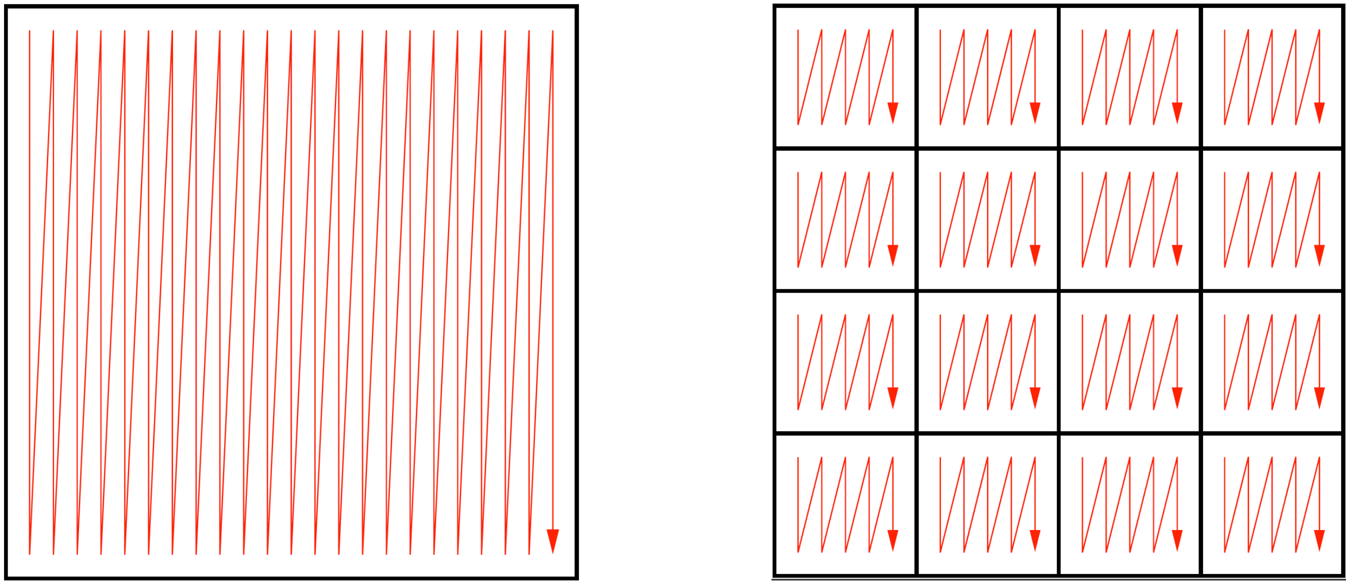
Parallel Tasks in LU/LL^T/QR



- Break into smaller tasks and remove dependencies



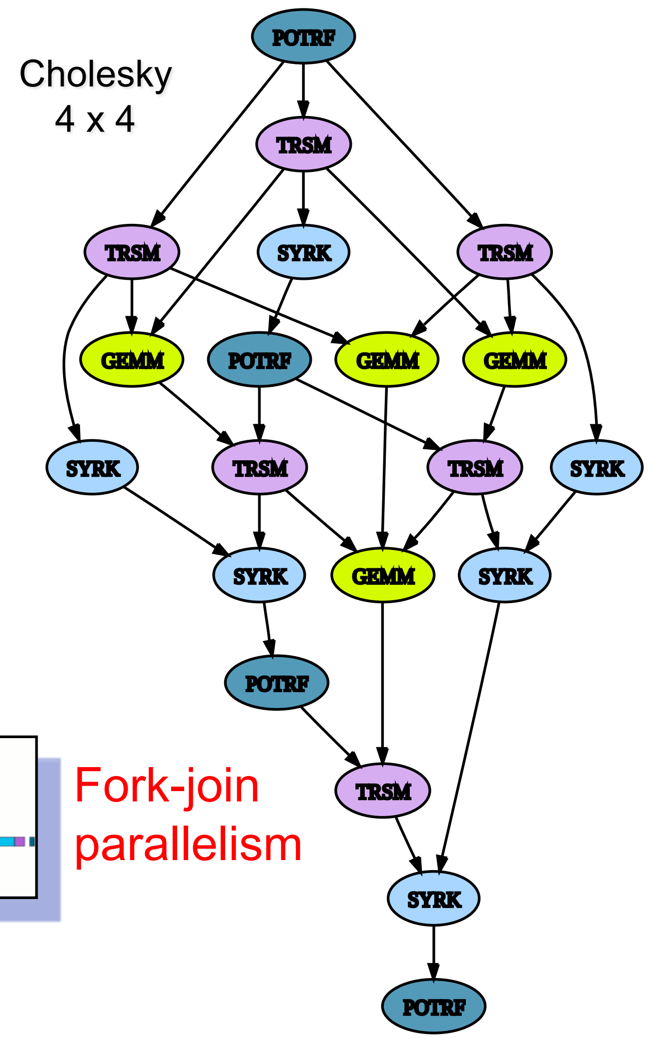
Data Layout is Critical

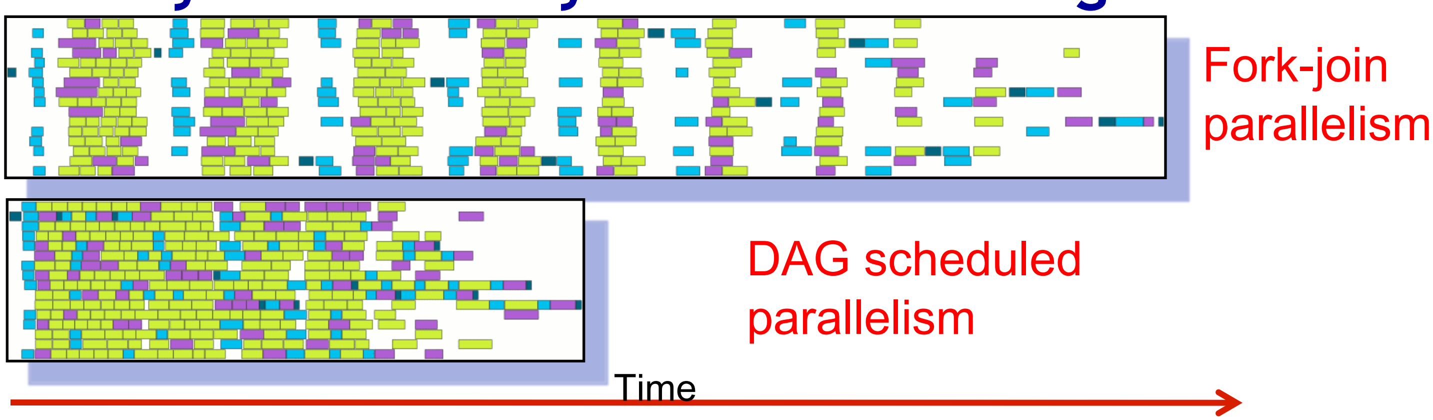


- **Tile data layout where each data tile is contiguous in memory**
- **Decomposed into several fine-grained tasks, which better fit the memory of the small core caches**

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 (QUARK)
  - Explicit parallelism
  - Implicit communication
  - Fine granularity / block data layout
- **Arbitrary DAG with dynamic scheduling**

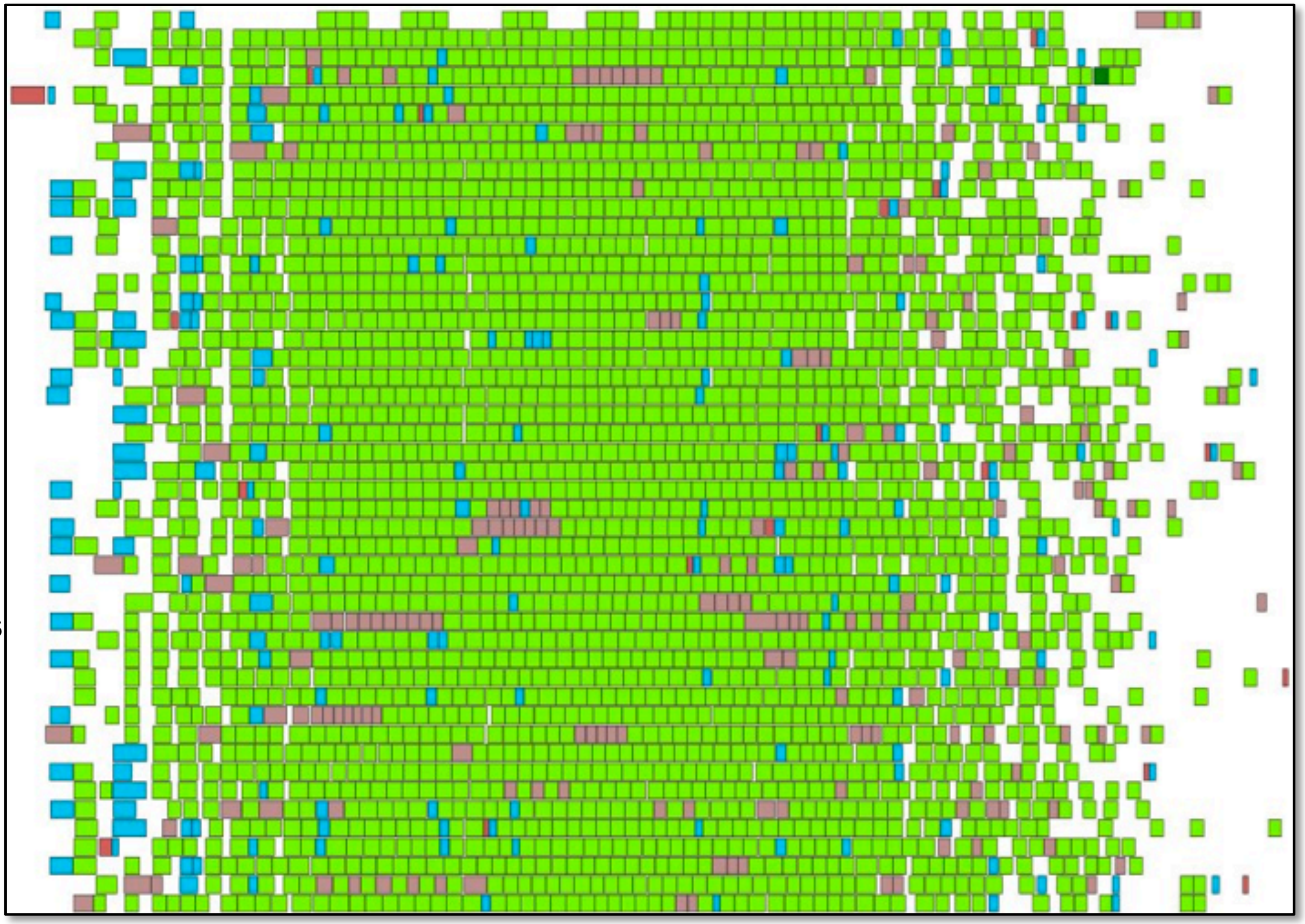




Synchronization Reducing Algorithms

- Regular trace
- Factorization steps pipelined
- Stalling only due to natural load imbalance
- Dynamic
- Out of order execution
- Fine grain tasks
- Independent block operations

The colored area over the rectangle is the efficiency



Tile QR factorization; Matrix size 4000x4000, Tile size 200
8-socket, 6-core (48 cores total) AMD Istanbul 2.8 GHz

Reduction to Condensed Form for Symmetric Eigenvalue Problem and Singular Value Decomposition

- For the symmetric eigenvalue problem the reduction is the expensive part
  - If just eigenvalues are required then 90% of the time to perform the reduction.
  - If both values and vectors needed then 50% of the time spent in the reduction.
- The existing LAPACK and ScaLAPACK uses two sided block Householder transformations
- Results in a BLAS 2.5 based implementation.

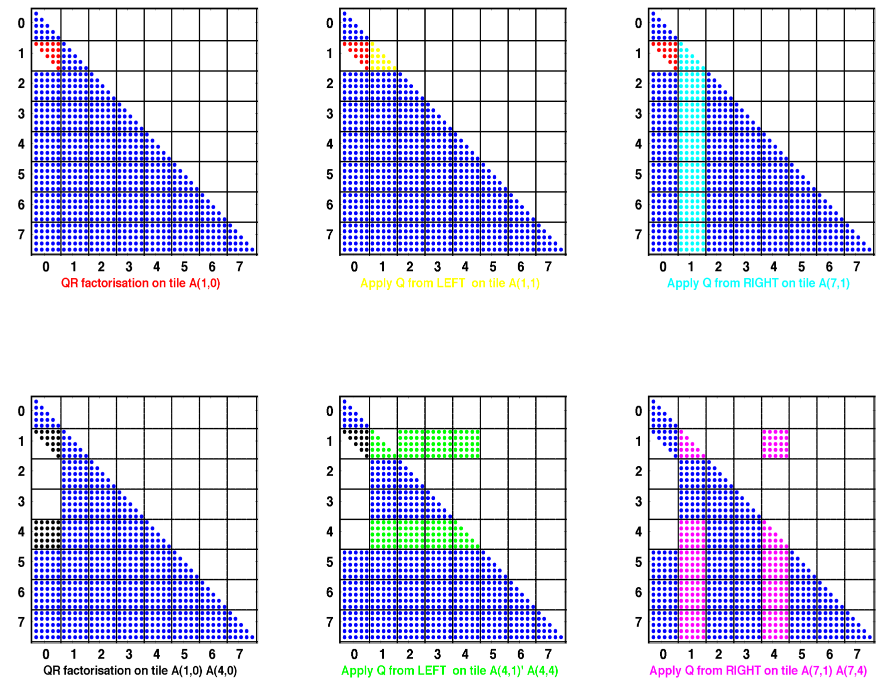
Reduction to Condensed Form for Symmetric Eigenvalue Problem and Singular Value Decomposition

Idea:

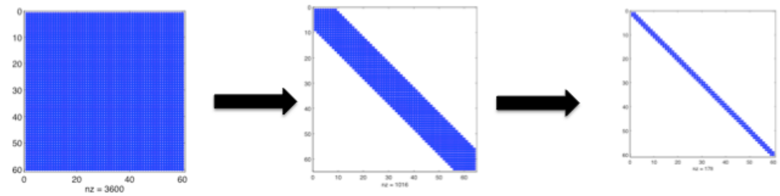
- Remove the fork join bottleneck by breaking the first stage algorithm into small granularity tasks in order to expose and to bring to the fore the parallelism residing within the BLAS library.

- The tile algorithm generates a directed acyclic graph (DAG), where nodes represent tasks and edges describe the data dependencies between them.

- Those tasks are scheduled asynchronously in an out-of -order fashion.

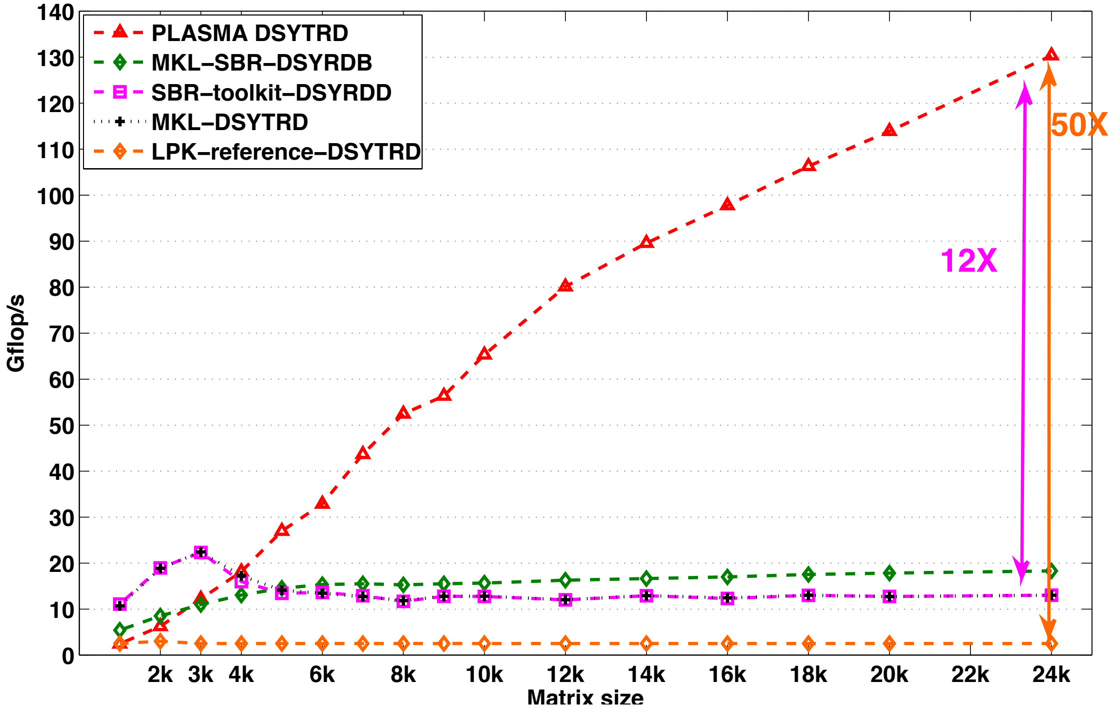


Performance



Eigenvalues

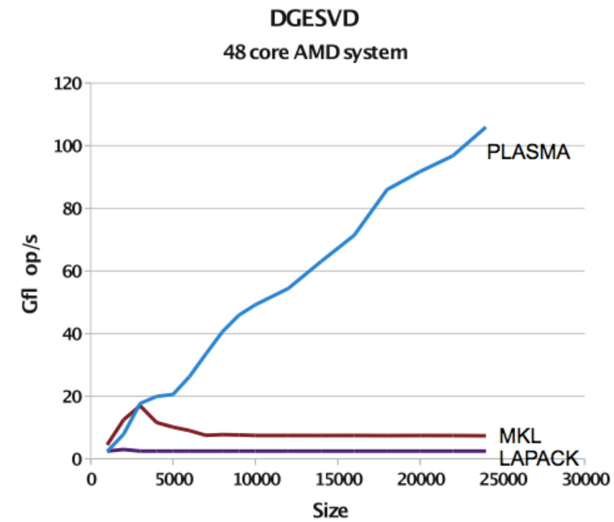
eigenvalues only



Experiments on eight-socket six-core AMD Opteron 2.4 GHz processors with MKL V10.3.

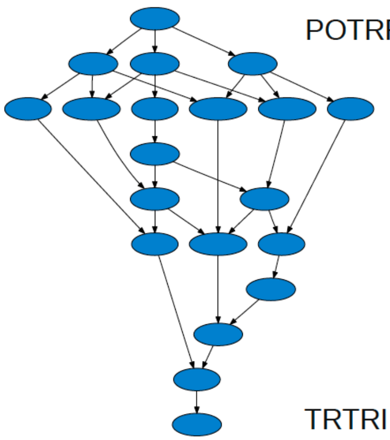
Singular Values

singular values only

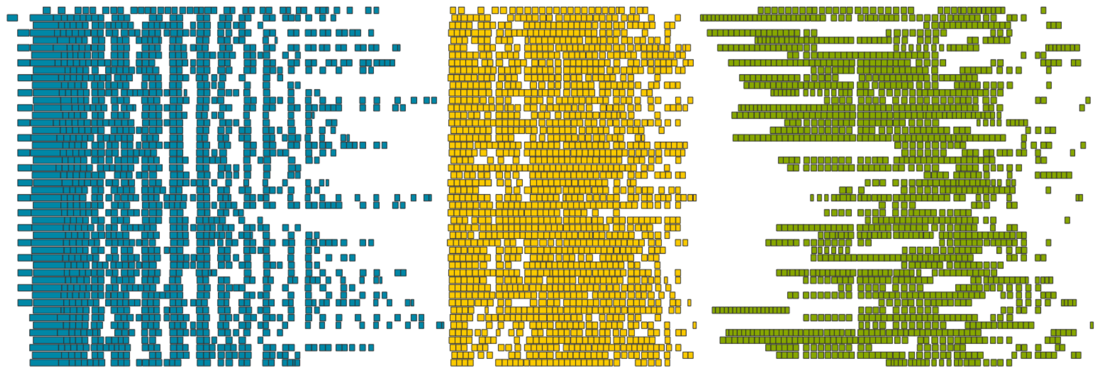


- Block DAG based to banded form, then pipelined group chasing to tridiagaonal form.
- The reduction to condensed form accounts for the factor of 50 improvement over LAPACK
- Execution rates based on $4/3n^3$ ops

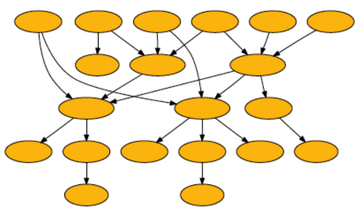
Pipelining: Cholesky Inversion
3 Steps: Factor, Invert L, Multiply L's



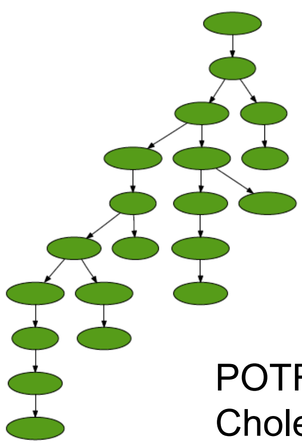
POTRF

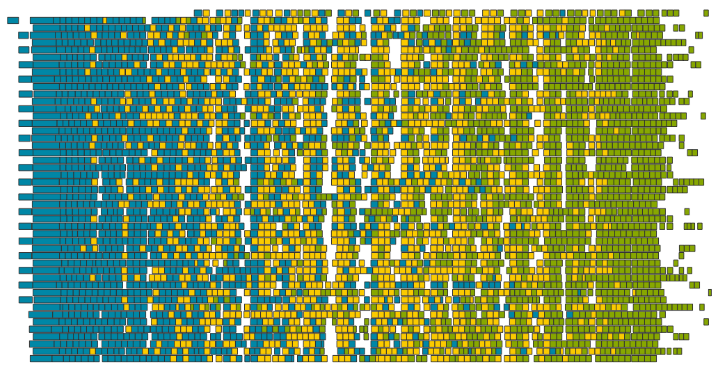


TRTRI

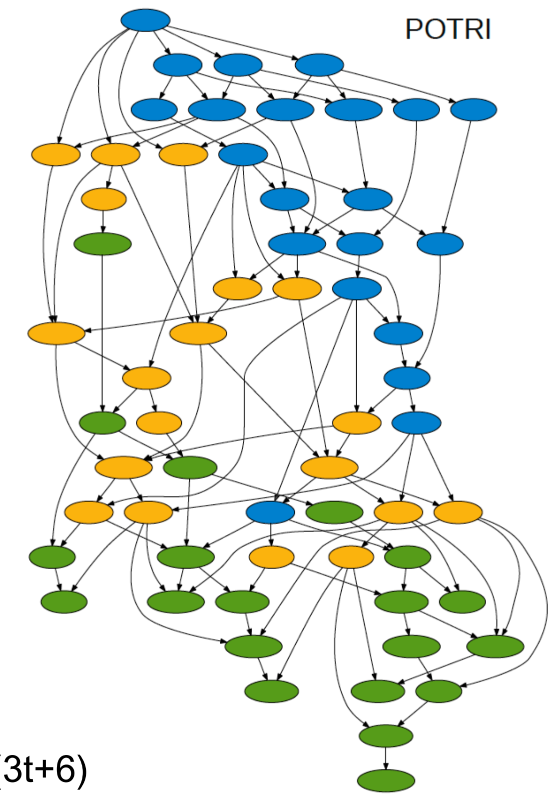


LAUUM





POTRI



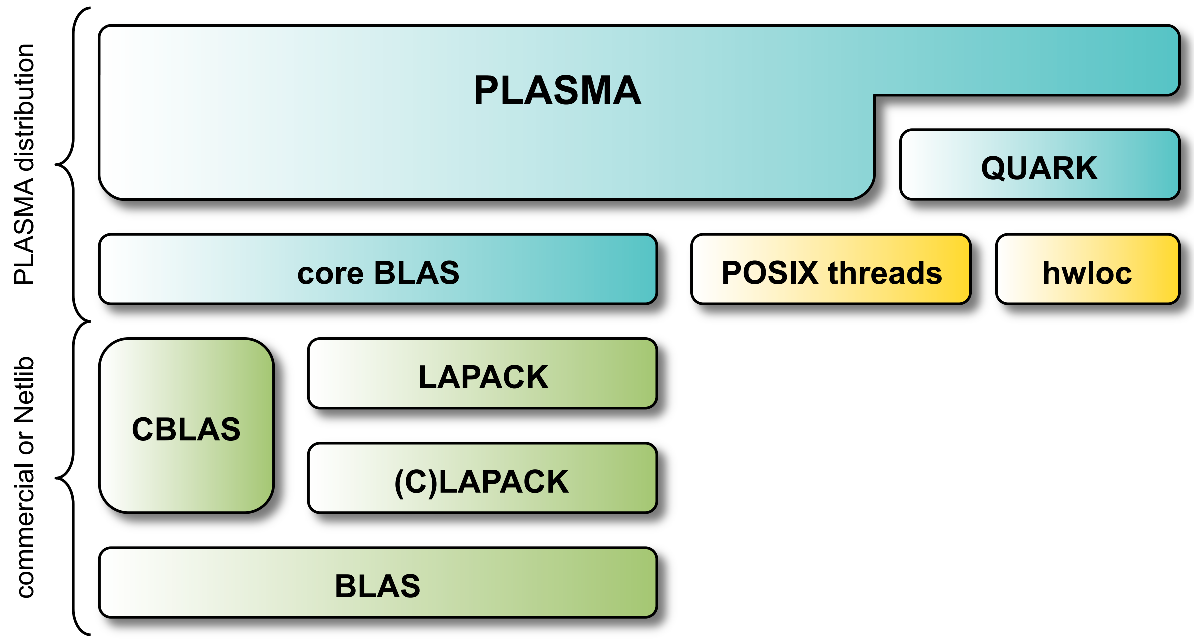
48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200,

POTRF+TRTRI+LAUUM: 25 (7t-3)
Cholesky Factorization alone: 3t-2

Pipelined: 18 (3t+6)

Software Stack

PLASMA



QUARK - **QU**euing **A**nd **R**untime for **K**ernels

LAPACK - **L**inear **A**lgebra **PACK**age

BLAS - **B**asic **L**inear **A**lgebra **S**ubroutines

hwloc - **h**ard**w**are **loc**ality

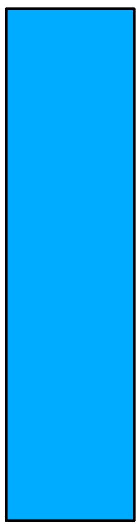
Communication Avoiding Algorithms

- Goal: Algorithms that communicate as little as possible
- Jim Demmel and company have been working on algorithms that obtain a provable minimum communication. (M. Anderson yesterday)
- 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)
- Iterative methods – Krylov subspace methods for Ax=b, Ax=λx
  - Communication lower bounds, and algorithms that attain them (depending on sparsity structure)
- For QR Factorization they can show:

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

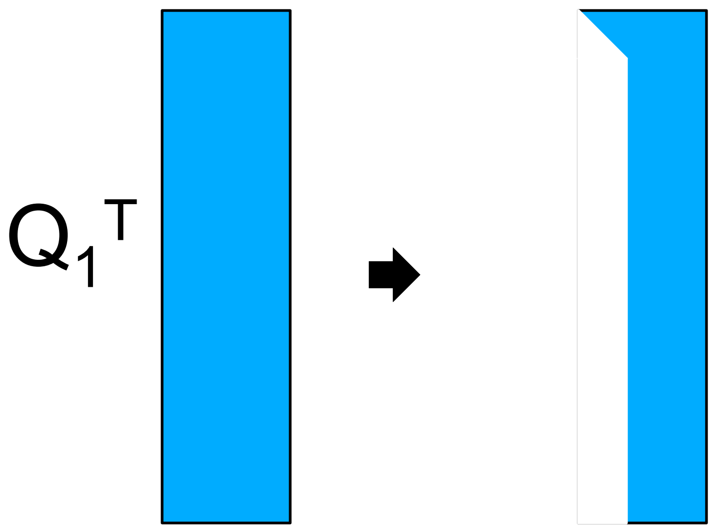
Standard QR Block Reduction

- **We have a *m x n* matrix *A* we want to reduce to upper triangular form.**



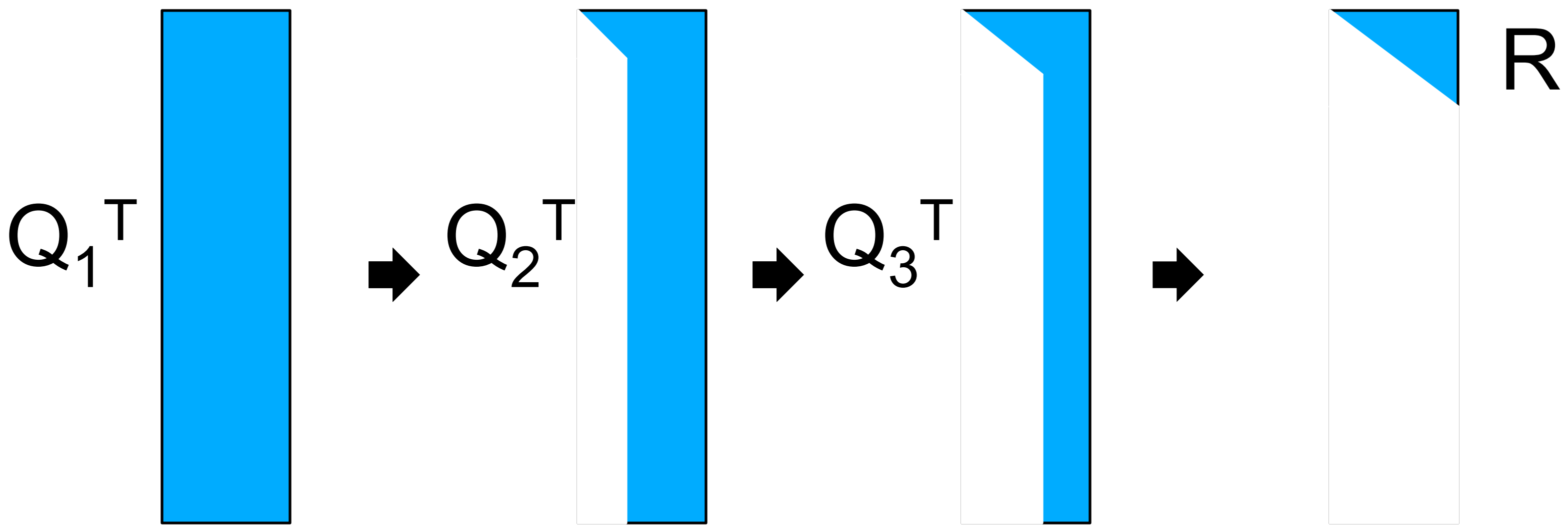
Standard QR Block Reduction

- **We have a *m x n* matrix *A* we want to reduce to upper triangular form.**



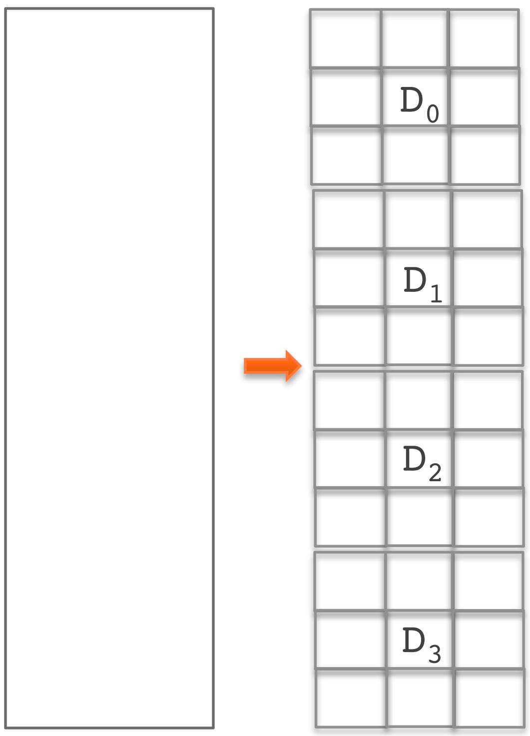
Standard QR Block Reduction

- **We have a *m x n* matrix *A* we want to reduce to upper triangular form.**



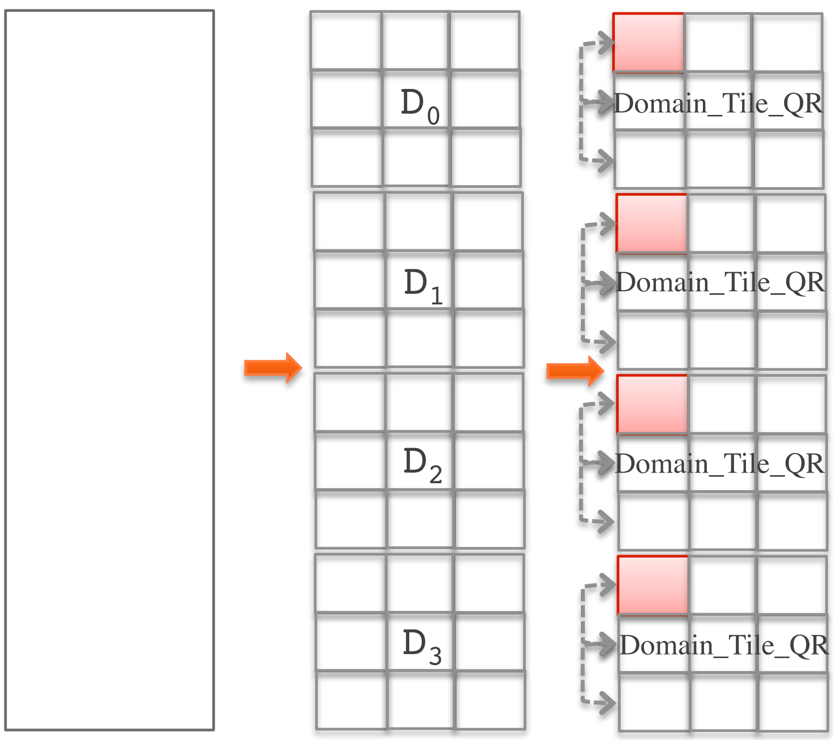
$$A = Q_1 Q_2 Q_3 R = QR$$

Communication Avoiding QR Example



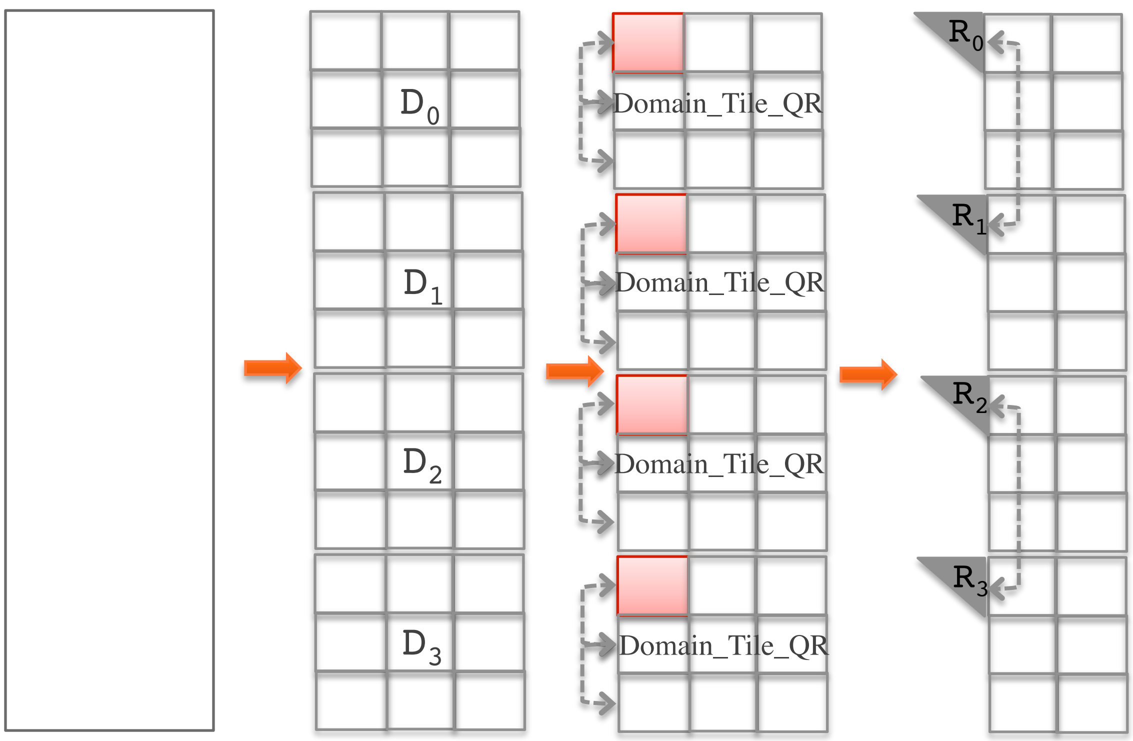
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Avoiding QR Example



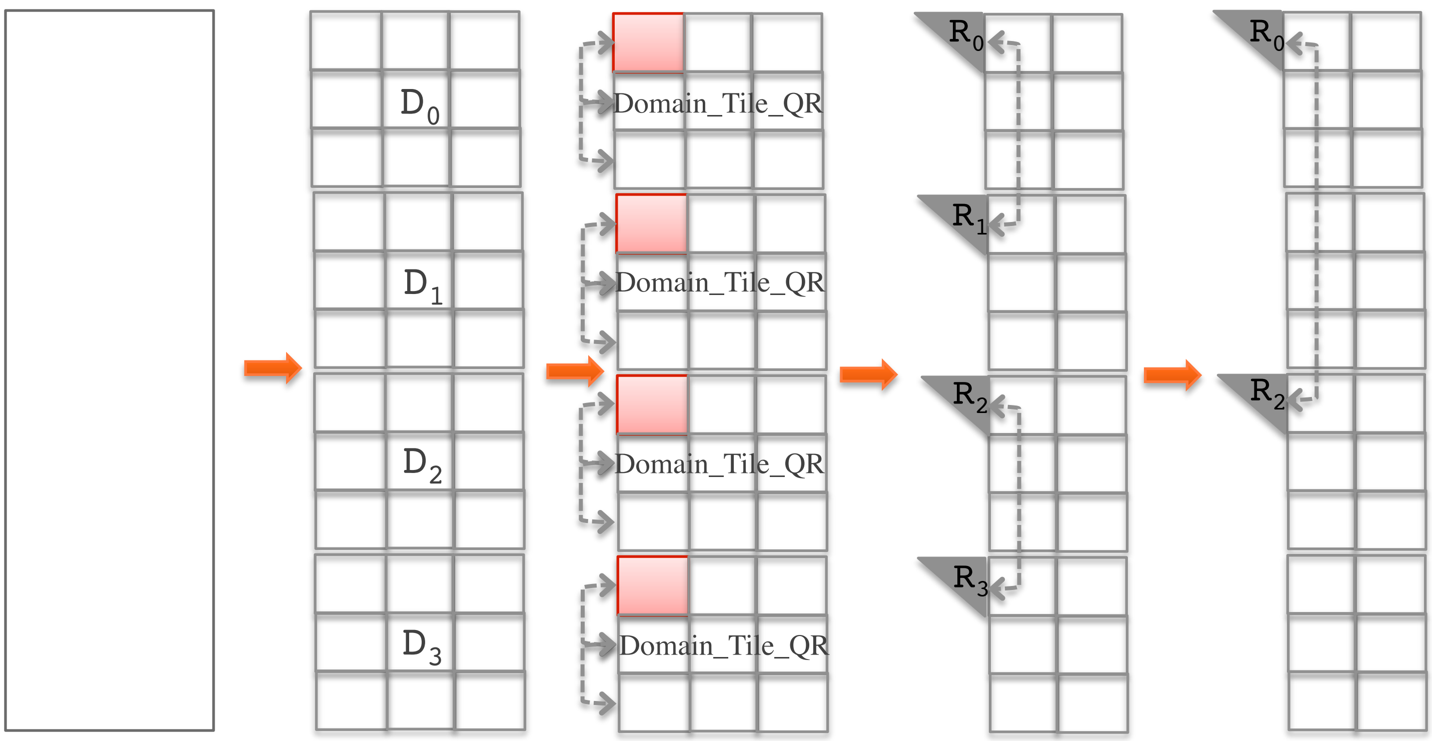
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Avoiding QR Example



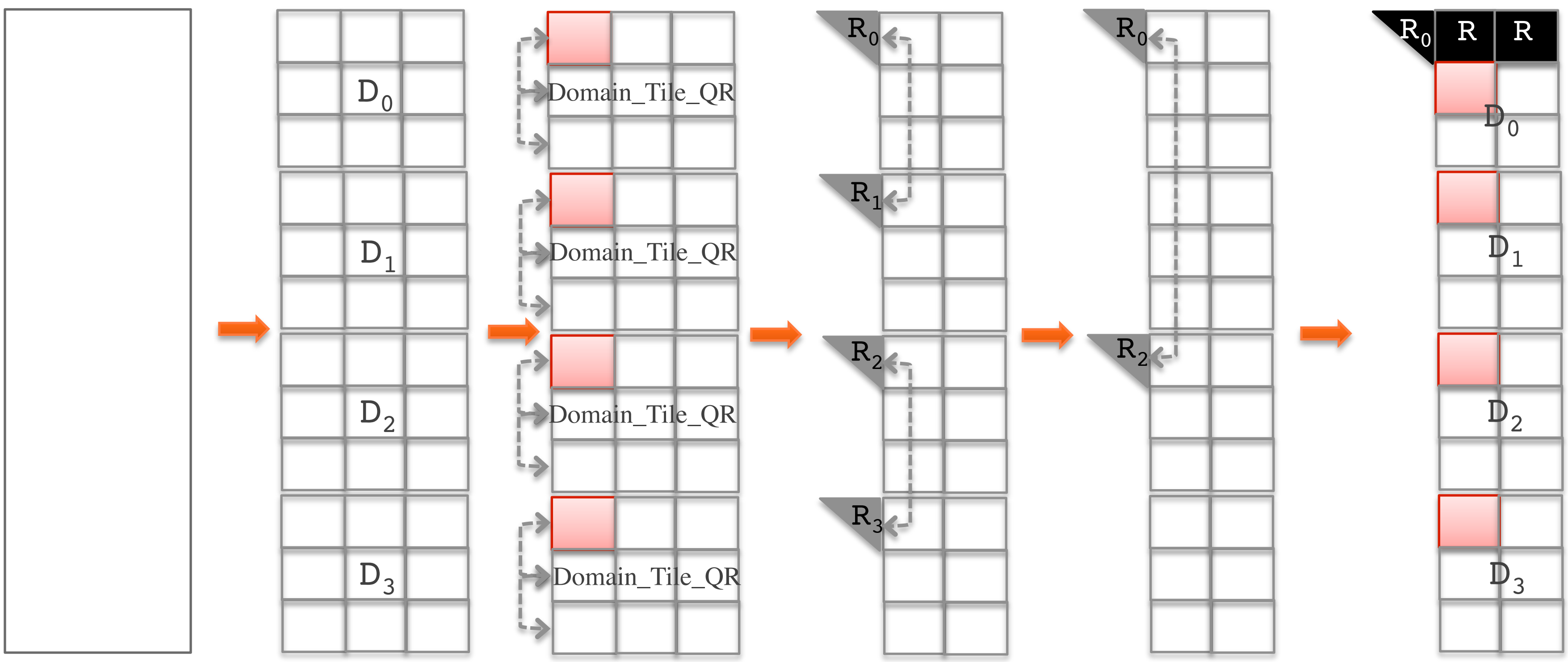
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Avoiding QR Example



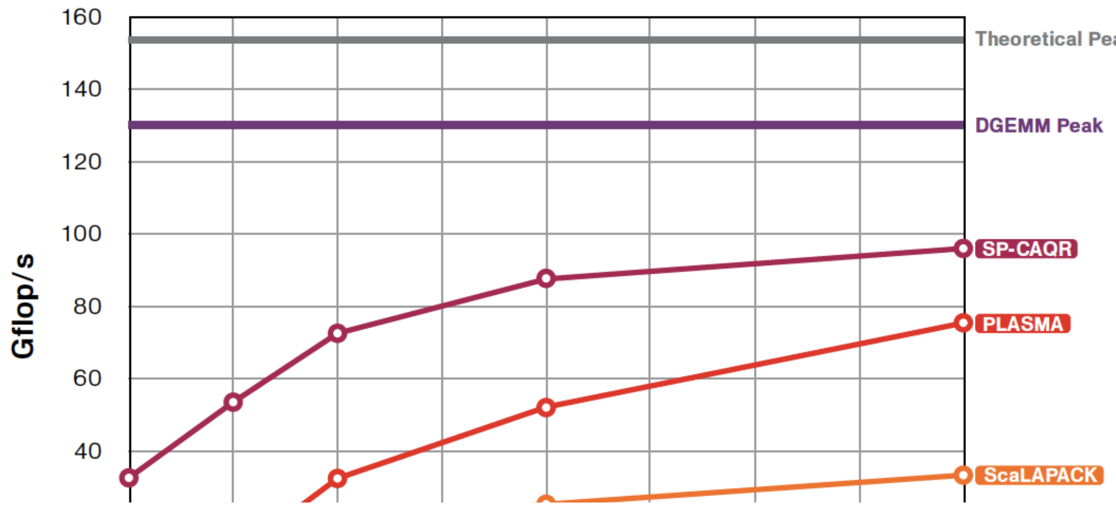
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

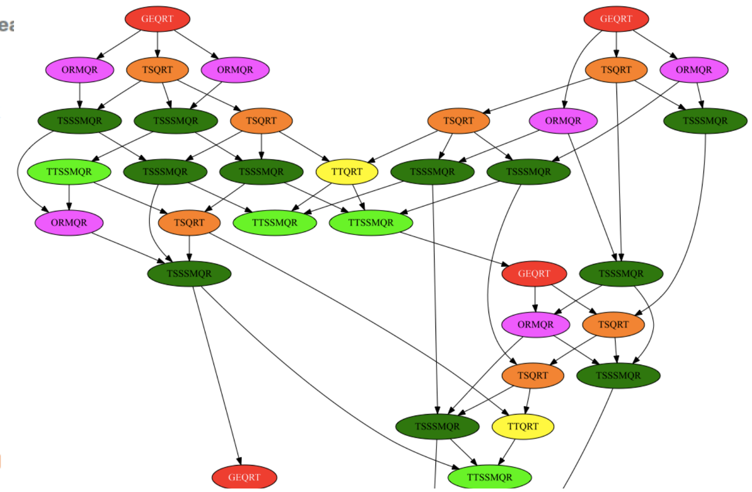
Communication Avoiding QR Example

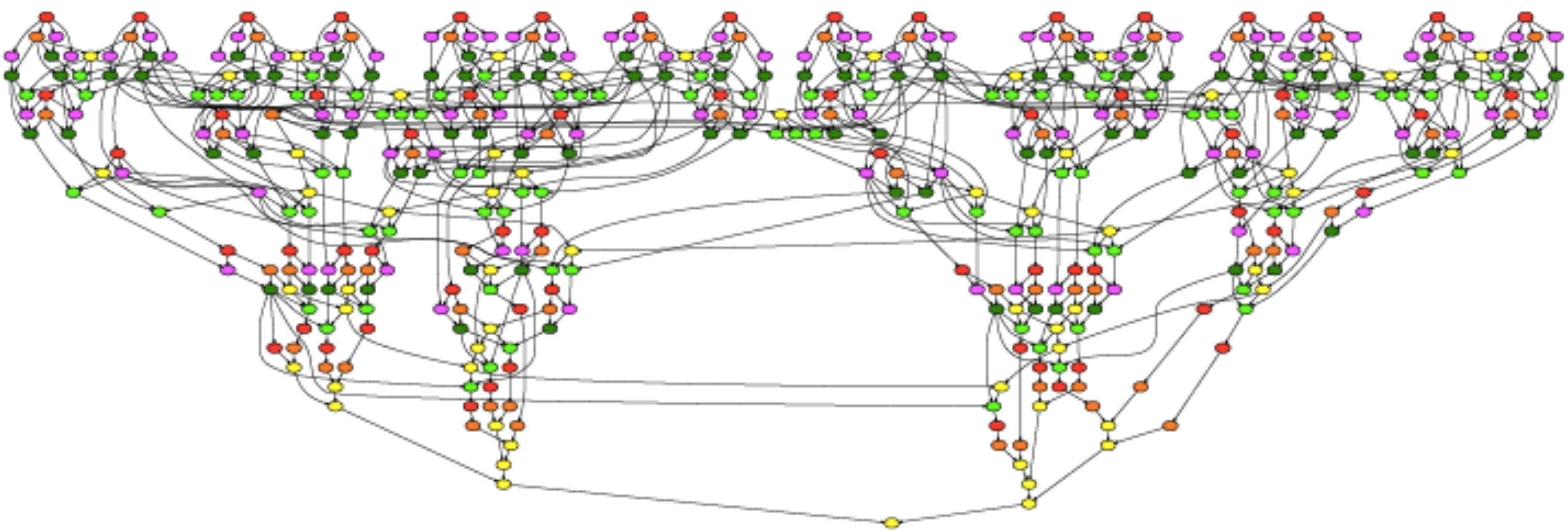


A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Reducing QR Factorization







Mixed Precision Methods

- **Mixed precision, use the lowest precision required to achieve a given accuracy outcome**
  - Improves runtime, reduce power consumption, lower data movement
  - Reformulate to find correction to solution, rather than solution; Δx rather than x.

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

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

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)	$O(n^3)$
x = L\(U\b)	$O(n^2)$
r = b – Ax	$O(n^2)$
WHILE || r || not small enough	
z = L\(U\r)	$O(n^2)$
x = x + z	$O(n^1)$
r = b – Ax	$O(n^2)$
END	

- 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.

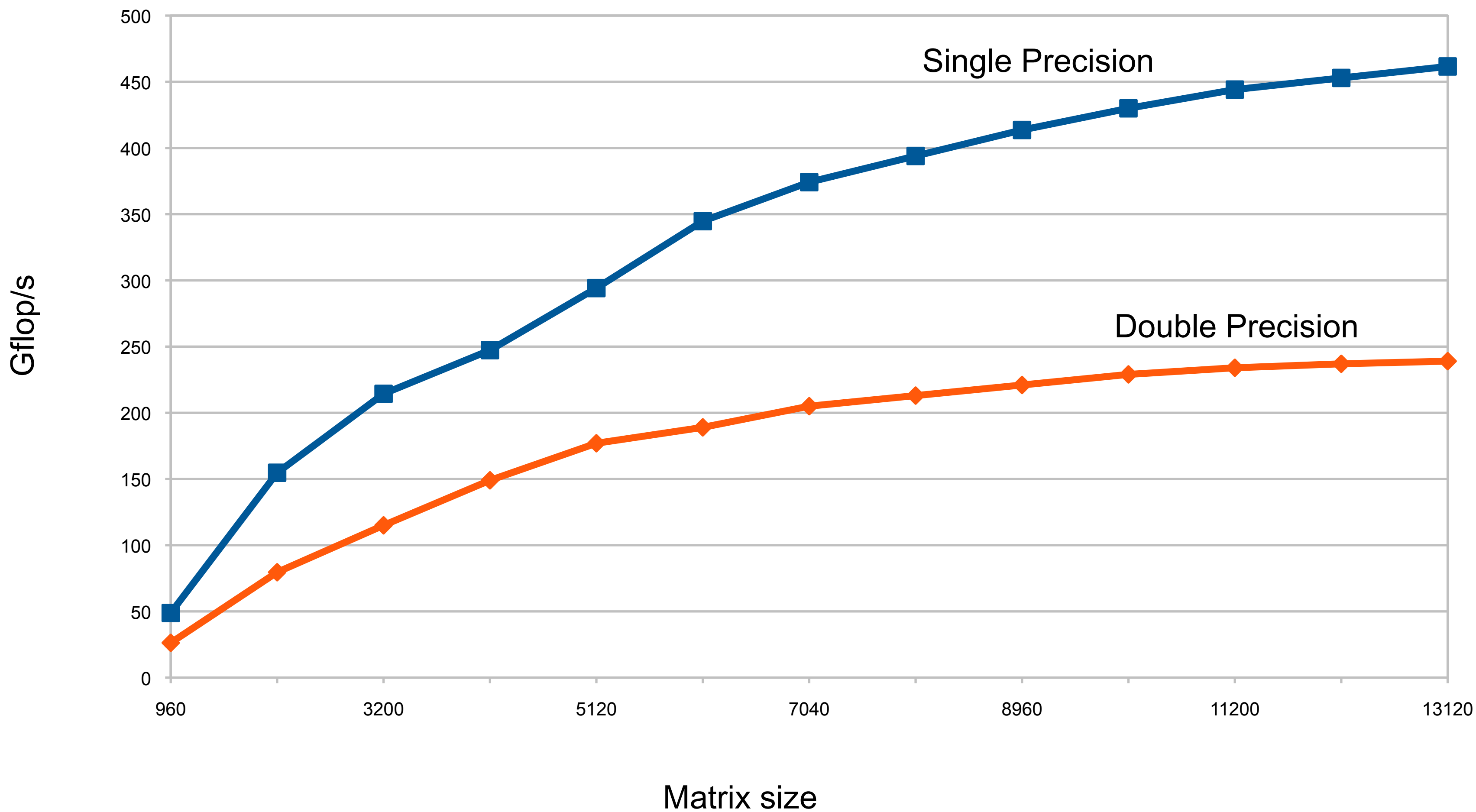
L U = lu(A)	SINGLE	$O(n^3)$
x = L\(U\b)	SINGLE	$O(n^2)$
r = b – Ax	DOUBLE	$O(n^2)$
WHILE || r || not small enough		
z = L\(U\r)	SINGLE	$O(n^2)$
x = x + z	DOUBLE	$O(n^1)$
r = b – Ax	DOUBLE	$O(n^2)$
END		

- 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; $O(10^8)$

Ax = b

FERMI Tesla C2050: 448 CUDA cores @ 1.15GHz
SP/DP peak is 1030 / 515 GFlop/s

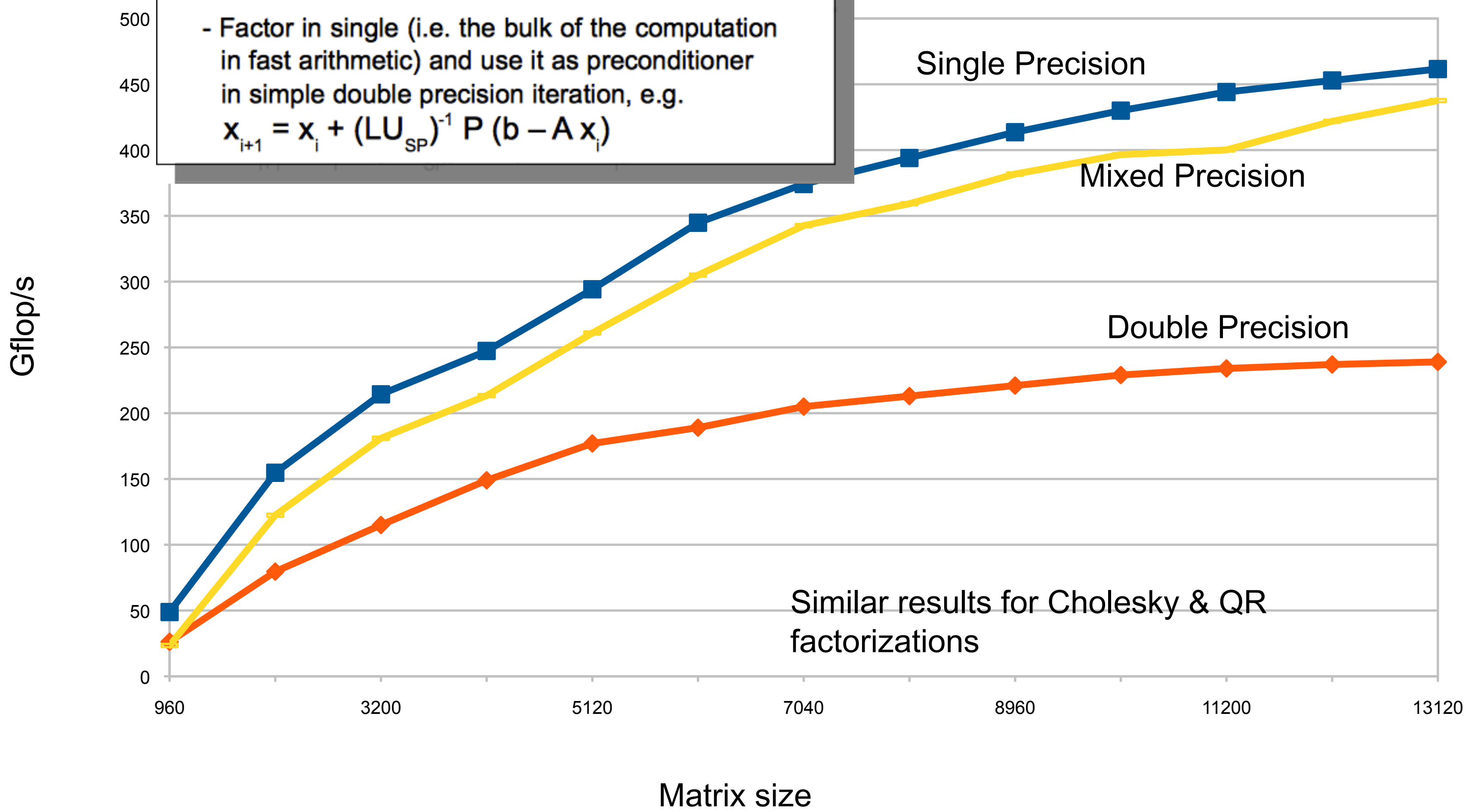


Ax = b

FERMI Tesla C2050: 448 CUDA cores @ 1.15GHz
SP/DP peak is 1030 / 515 GFlop/s

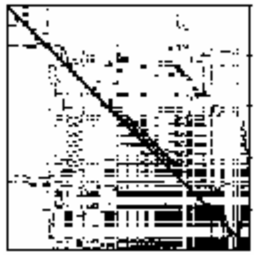
- **Direct solvers**
  - Factor and solve in working precision
- **Mixed Precision Iterative Refinement**
  - Factor in single (i.e. the bulk of the computation in fast arithmetic) and use it as preconditioner in simple double precision iteration, e.g.

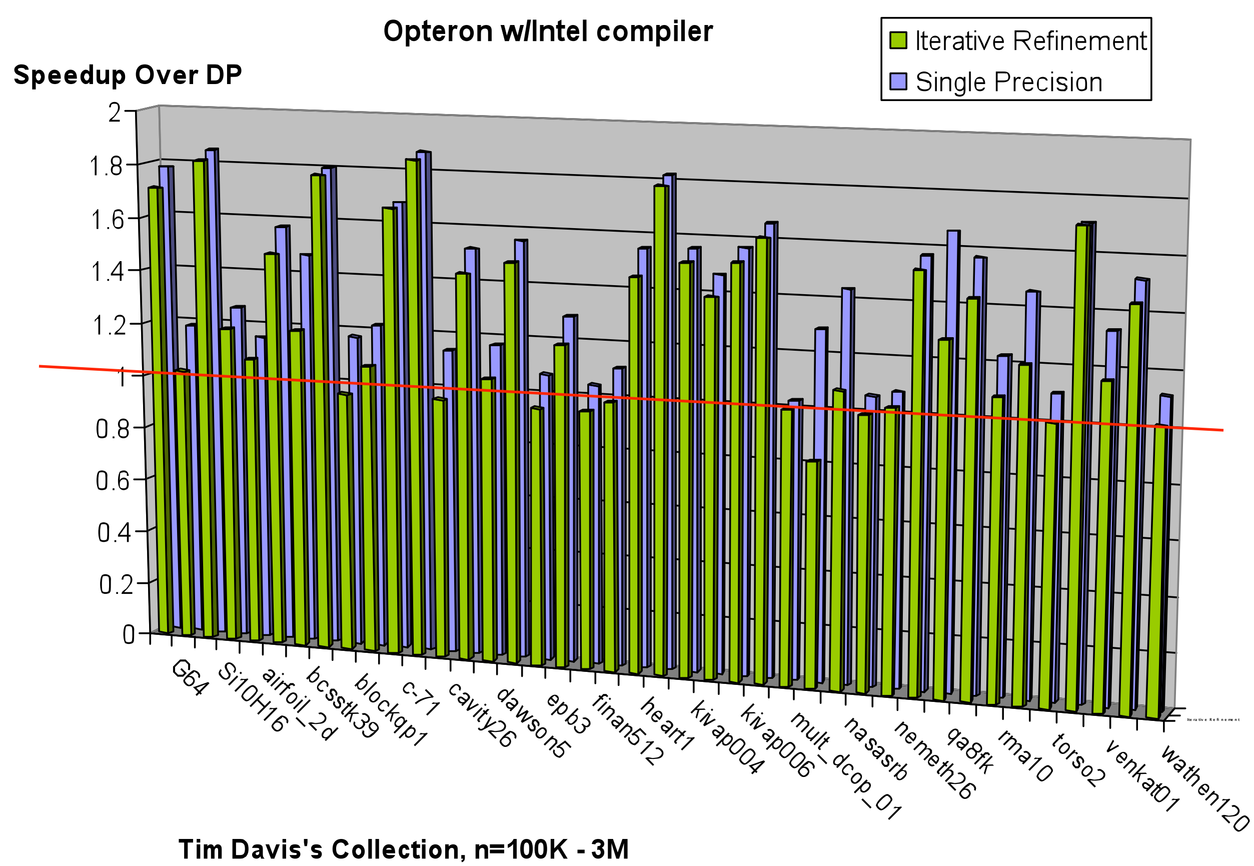
$$x_{i+1} = x_i + (LU_{SP})^{-1} P (b - A x_i)$$



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

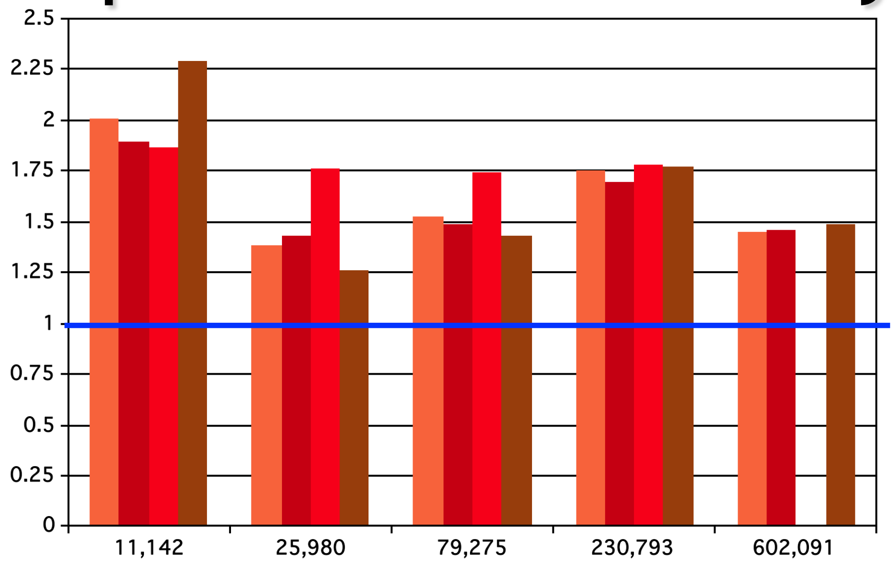
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  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

Inner iteration:
In 32 bit floating point

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  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

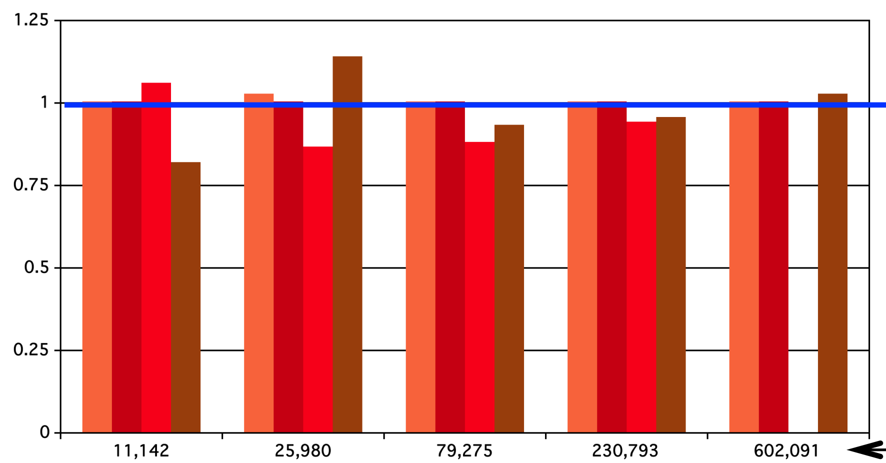
- **Outer iteration in 64 bit floating point and inner iteration in 32 bit floating point**

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
(CG^2, $GMRES^2$, PCG^2, and $PGMRES^2$ with diagonal prec.)
(***Higher is better***)

- **CG^2**
- **PCG^2**
- **$GMRES^2$**
- **$PGMRES^2$**



← Matrix size

6,021 **18,000** **39,000** **120,000** **240,000** ← Condition number

Iterations for mixed precision
SP/DP iterative methods *vs* DP/DP
(***Lower is better***)

Machine:
Intel Woodcrest (3GHz, 1333MHz bus)

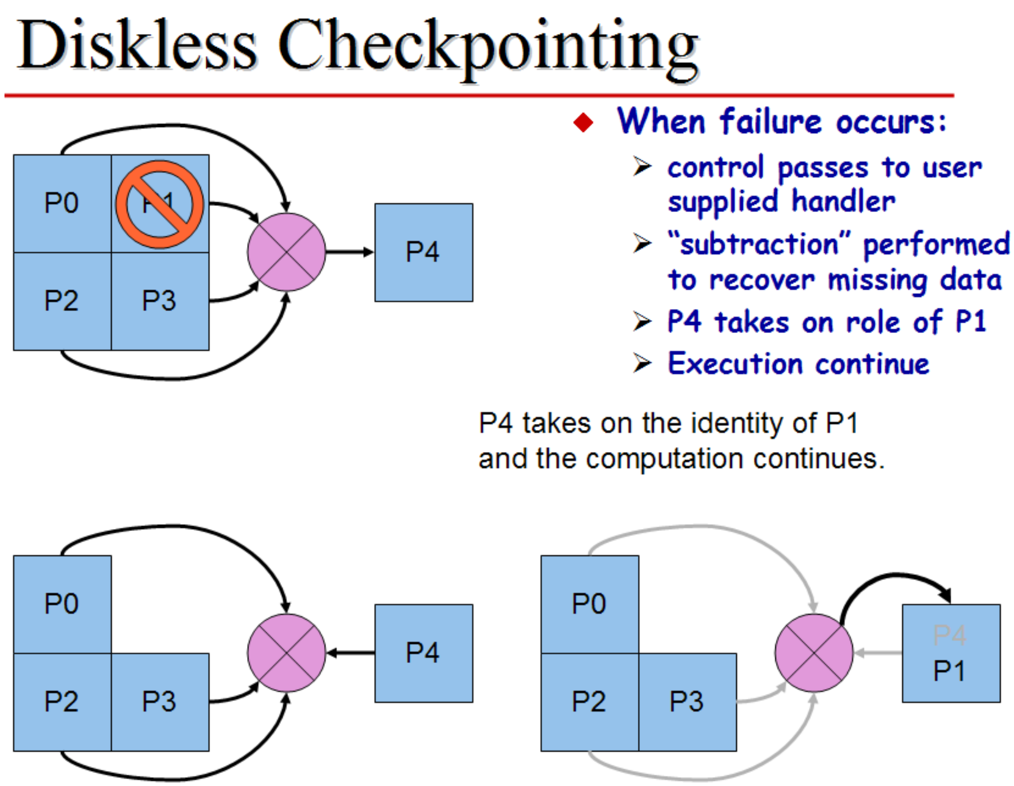
Stopping criteria:
Relative to r_0 residual reduction (10^{-12})

Reproducibility

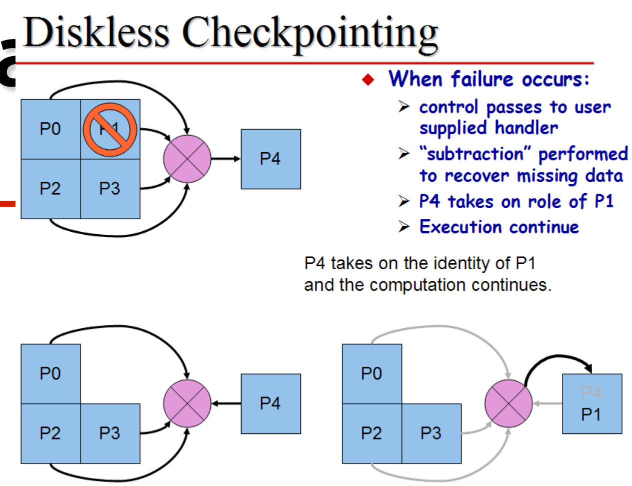
- For example $\sum x_i$ when done in parallel can't guarantee the order of operations.
- Lack of reproducibility due to floating point nonassociativity and algorithmic adaptivity (including autotuning) in efficient production mode
- Bit-level reproducibility may be unnecessarily expensive most of the time
- Force routine adoption of uncertainty quantification
  - Given the many unresolvable uncertainties in program inputs, bound the error in the outputs in terms of errors in the inputs

Three Ideas for Fault Tolerant Linear Algebra Algorithms

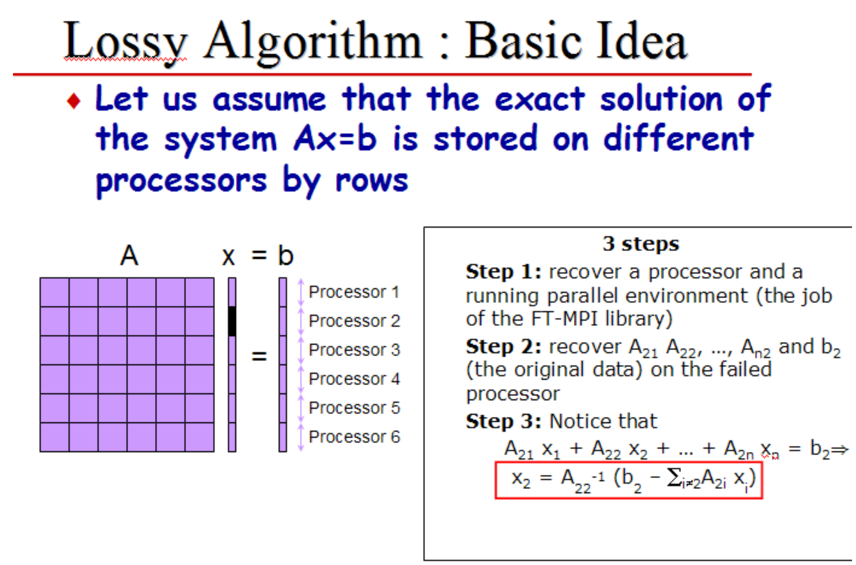
- **Lossless diskless check-pointing for iterative methods**
  - **Checksum maintained in active processors**
  - **On failure, roll back to checkpoint and continue**
  - **No lost data**



Three Ideas for Fault Tolera
Linear Algebra Algorithms

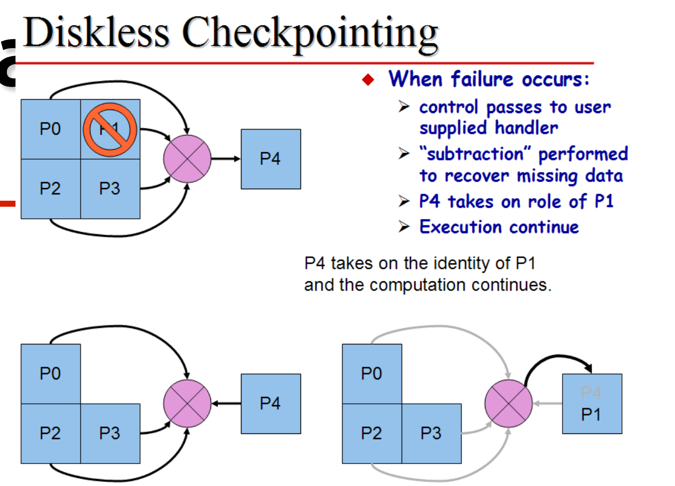


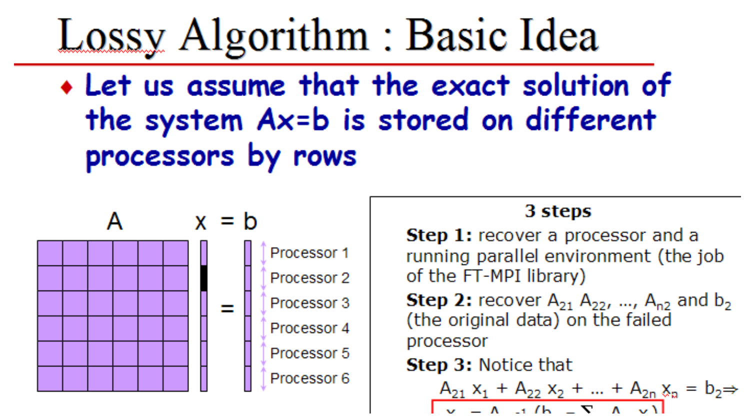
- Lossless diskless check-pointing for iterative methods
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data
- **Lossy approach for iterative methods**
  - **No checkpoint for computed data maintained**
  - **On failure, approximate missing data and carry on**
  - **Lost data but use approximation to recover**

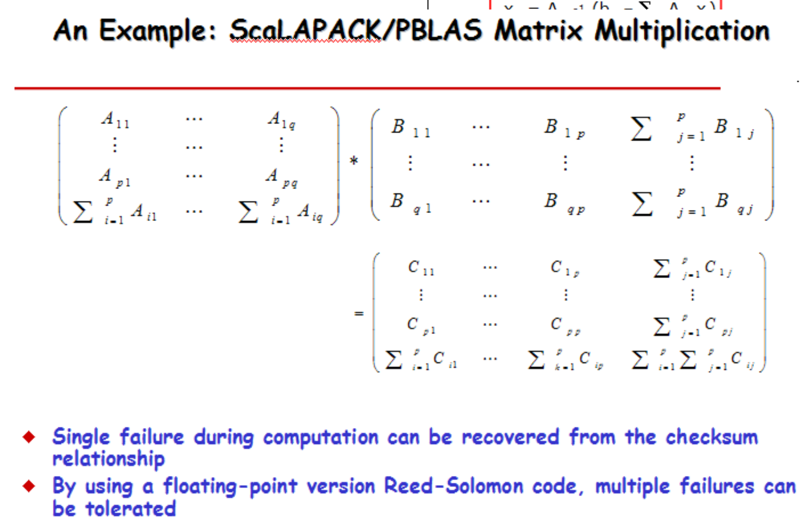


Three Ideas for Fault Tolera
Linear Algebra Algorithms

- Lossless diskless check-pointing for iterative methods
  - Checksum maintained in active processors
  - On failure, roll back to checkpoint and continue
  - No lost data
- Lossy approach for iterative methods
  - No checkpoint maintained
  - On failure, approximate missing data and carry on
  - Lost data but use approximation to recover
- **Check-pointless methods for dense algorithms**
  - **Checksum maintained as part of computation**
  - **No roll back needed; No lost data**







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.
- High Performance Ecosystem out of balance
  - Hardware, OS, Compilers, Software, Algorithms, Applications
    - No Moore's Law for software, algorithms and applications
- Our community is needed and has a great deal to offer.
- "The golden age of numerical analysis has not yet started!" - Volker Mehrmann

PLASMA

People

- **Current Team**
  - Dulceneia Becker
  - Henricus Bouwmeester
  - Jack Dongarra
  - Mathieu Faverge
  - Azzam Haidar
  - Blake Haugen
  - Jakub Kurzak
  - Julien Langou
  - Hatem Ltaief
  - Piotr Łuszczek
- **Past Members**
  - Emmanuel Agullo
  - Wesley Alvaro
  - Alfredo Buttari
  - Bilel Hadri
- **Outside Contributors**
  - Fred Gustavson
  - Lars Karlsson
  - Bo Kågström