With Extreme Scale Computing the Rules Have Changed

Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

Outline

- Overview of High Performance Computing
- With Extreme Computing the “rules” for computing have changed

White House HPC Initiative

The White House
Office of the Press Secretary

For Immediate Release July 29, 2015

Executive Order -- Creating a National Strategic Computing Initiative

EXECUTIVE ORDER

- - - - - - -

CREATING A NATIONAL STRATEGIC COMPUTING INITIATIVE

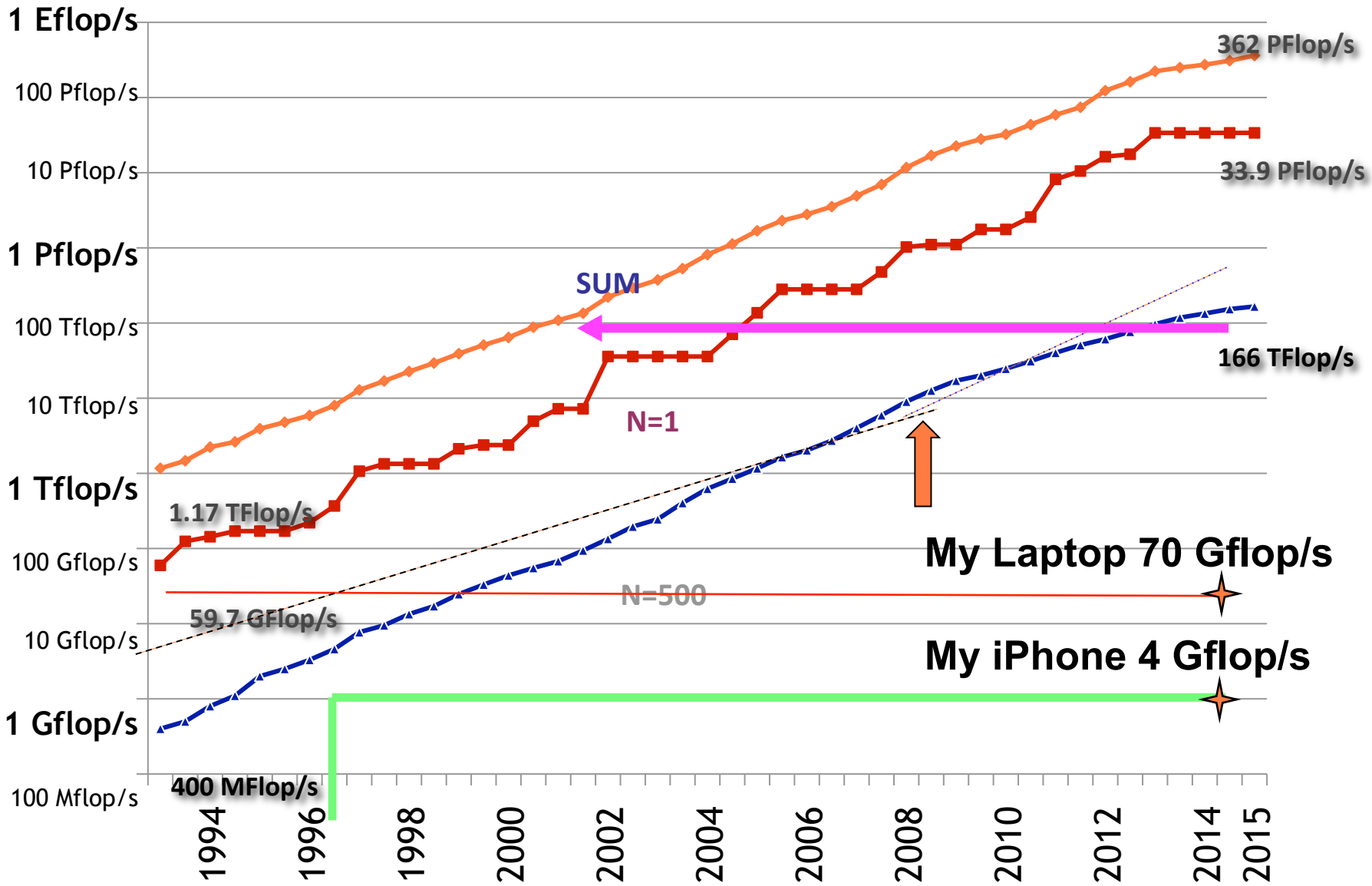
By the authority vested in me as President by the Constitution and the laws of the United States of America, and to maximize benefits of high-performance computing (HPC) research, development, and deployment, it is hereby ordered as follows:

Section 1. Policy. In order to maximize the benefits of HPC for economic competitiveness and scientific discovery, the United States Government must create a coordinated Federal strategy in HPC research, development, and deployment. Investment in HPC has contributed substantially to national economic prosperity and rapidly accelerated scientific discovery. Creating and deploying technology at the leading edge is vital to advancing my Administration's priorities and spurring innovation. Accordingly,

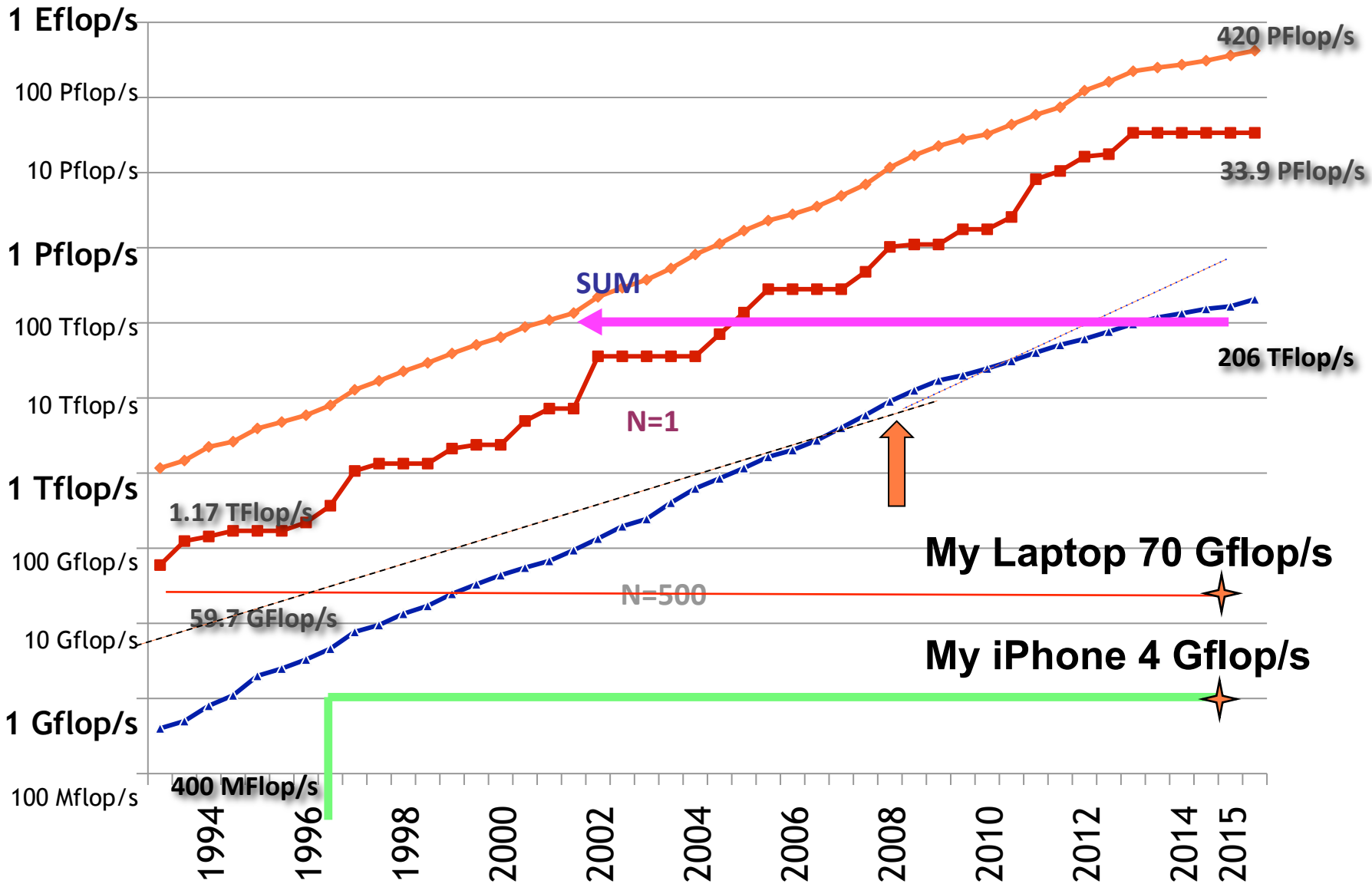
NSCI has 5 Strategic Themes

- Create systems that can apply exaflops of computing power to exabytes of data.
- Keep the United States at the forefront of HPC capabilities.
- Improve HPC application developer productivity
- Make HPC readily available
- Establish hardware technology for future HPC systems.

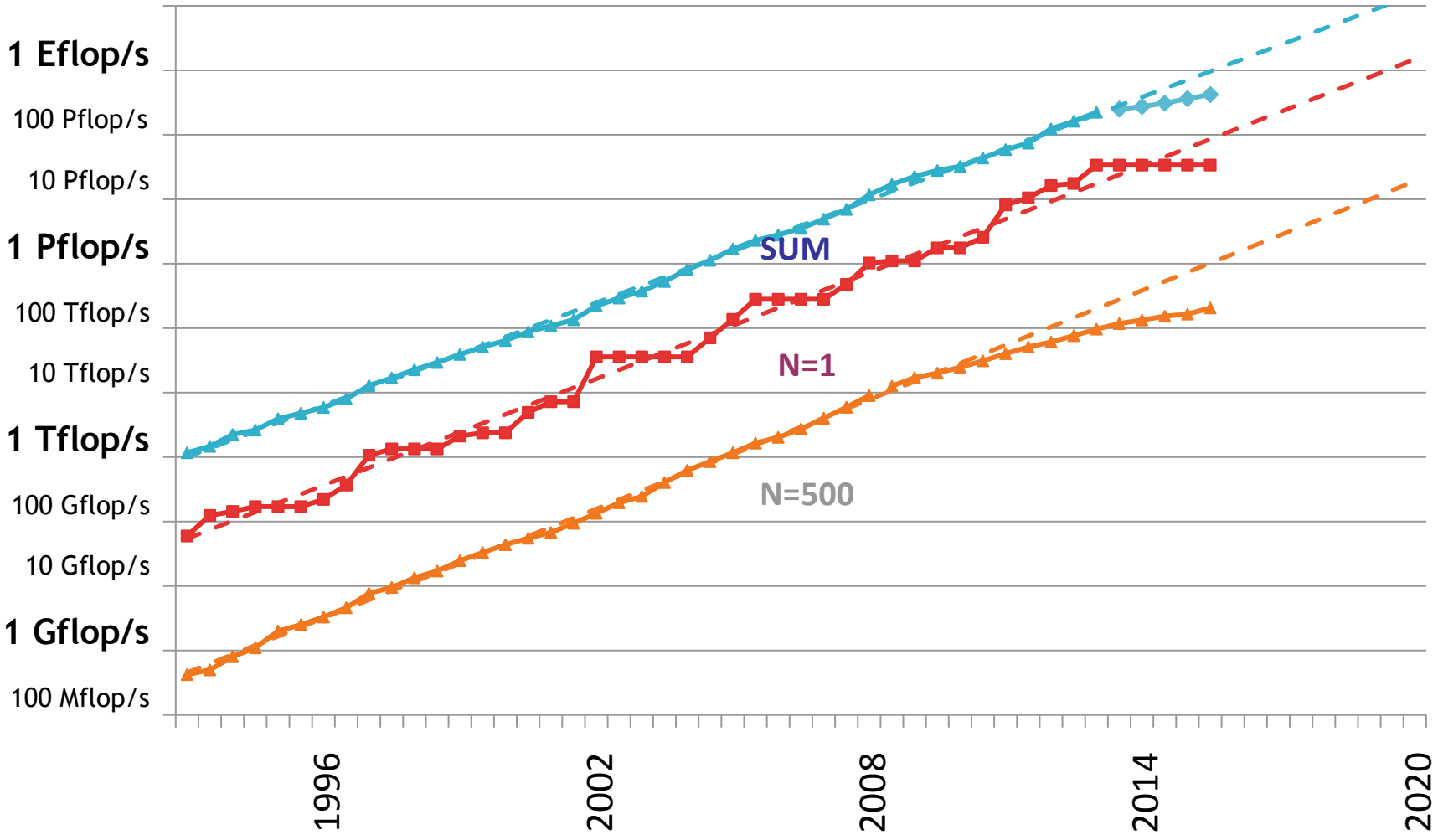
Performance Development of HPC over the Last 23 Years from the Top500



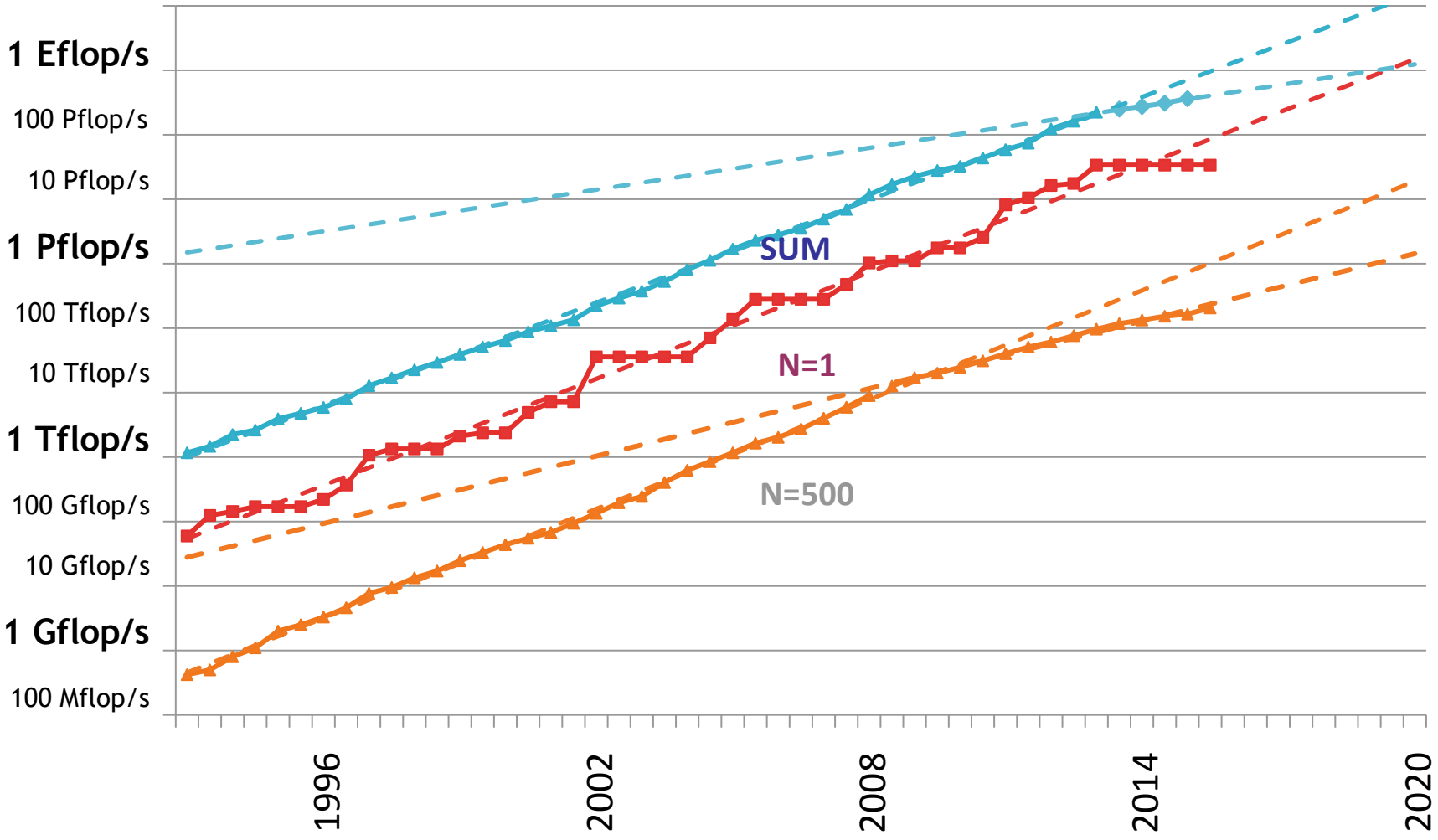
Performance Development of HPC over the Last 24 Years from the Top500



Projected Performance Development



Projected Performance Development



State of Supercomputing in 2015

- Pflops (> 10^{15} Flop/s) computing fully established with 81 systems.
- Three technology architecture possibilities or "swim lanes" are thriving.
  - Commodity (e.g. Intel)
  - Commodity + accelerator (e.g. GPUs) (104 systems)
  - Special purpose lightweight cores (e.g. IBM BG, ARM, Knights Landing)
- Interest in supercomputing is now worldwide, and growing in many new markets (over 50% of Top500 computers are in industry).
- Exascale (10^{18} Flop/s) projects exist in many countries and regions.
- Intel processors largest share, 89% followed by AMD, 4%.

July 2015: The TOP 10 Systems

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	MFlops /Watt
1	National Super Computer Center in Guangzhou	Tianhe-2 NUDT, Xeon 12C 2.2GHz + IntelXeon Phi (57c) + Custom	China	3,120,000	33.9	62	17.8	1905
2	DOE / OS Oak Ridge Nat Lab	Titan, Cray XK7, AMD (16C) + Nvidia Kepler GPU (14c) + Custom	USA	560,640	17.6	65	8.3	2120
3	DOE / NNSA L Livermore Nat Lab	Sequoia, BlueGene/Q (16c) + custom	USA	1,572,864	17.2	85	7.9	2063
4	RIKEN Advanced Inst for Comp Sci	K computer Fujitsu SPARC64 VIIIfx (8c) + Custom	Japan	705,024	10.5	93	12.7	827
5	DOE / OS Argonne Nat Lab	Mira, BlueGene/Q (16c) + Custom	USA	786,432	8.16	85	3.95	2066
6	Swiss CSCS	Piz Daint, Cray XC30, Xeon 8C + Nvidia Kepler (14c) + Custom	Swiss	115,984	6.27	81	2.3	2726
7	KAUST	Shaheen II, Cray XC30, Xeon 16C + Custom	Saudi Arabia	196,608	5.54	77	4.5	1146
8	Texas Advanced Computing Center	Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB	USA	204,900	5.17	61	4.5	1489
9	Forschungszentrum Juelich (FZJ)	JuQUEEN, BlueGene/Q, Power BQC 16C 1.6GHz+Custom	Germany	458,752	5.01	85	2.30	2178
10	DOE / NNSA L Livermore Nat Lab	Vulcan, BlueGene/Q, Power BQC 16C 1.6GHz+Custom	USA	393,216	4.29	85	1.97	2177
500 (422)	Software Comp	HP Cluster	USA	18,896	.309	48		

November 2015: The TOP 10 Systems

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	MFlops /Watt
1	National Super Computer Center in Guangzhou	Tianhe-2 NUDT, Xeon 12C + IntelXeon Phi (57c) + Custom	China	3,120,000	33.9	62	17.8	1905
2	DOE / OS Oak Ridge Nat Lab	Titan, Cray XK7, AMD (16C) + Nvidia Kepler GPU (14c) + Custom	USA	560,640	17.6	65	8.3	2120
3	DOE / NNSA L Livermore Nat Lab	Sequoia, BlueGene/Q (16c) + custom	USA	1,572,864	17.2	85	7.9	2063
4	RIKEN Advanced Inst for Comp Sci	K computer Fujitsu SPARC64 VIIIfx (8c) + Custom	Japan	705,024	10.5	93	12.7	827
5	DOE / OS Argonne Nat Lab	Mira, BlueGene/Q (16c) + Custom	USA	786,432	8.16	85	3.95	2066
6	DOE / NNSA / Los Alamos & Sandia	Trinity, Cray XC40,Xeon 16C + Custom	USA	301,056	8.10	80		
7	Swiss CSCS	Piz Daint, Cray XC30, Xeon 8C + Nvidia Kepler (14c) + Custom	Swiss	115,984	6.27	81	2.3	2726
8	HLRS Stuttgart	Hazel Hen, Cray XC40, Xeon 12C + Custom	Germany	185,088	5.64	76		
9	KAUST	Shaheen II, Cray XC30, Xeon 16C + Custom	Saudi Arabia	196,608	5.54	77	4.5	1146
10	Texas Advanced Computing Center	Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB	USA	204,900	5.17	61	4.5	1489
500 (368)	Karlsruher	MEGAWARE Intel	Germany	10,800	.206	95		

Recent Developments

- US DOE planning to deploy O(100) Pflop/s systems for 2017-2018 - $525M hardware
- Oak Ridge Lab and Lawrence Livermore Lab to receive IBM and Nvidia based systems
- Argonne Lab to receive Intel based system
  - After this Exaflops
- US Dept of Commerce is [illegible] groups from receiving In[illegible]

[illegible] cle[illegible] th the [illegible]

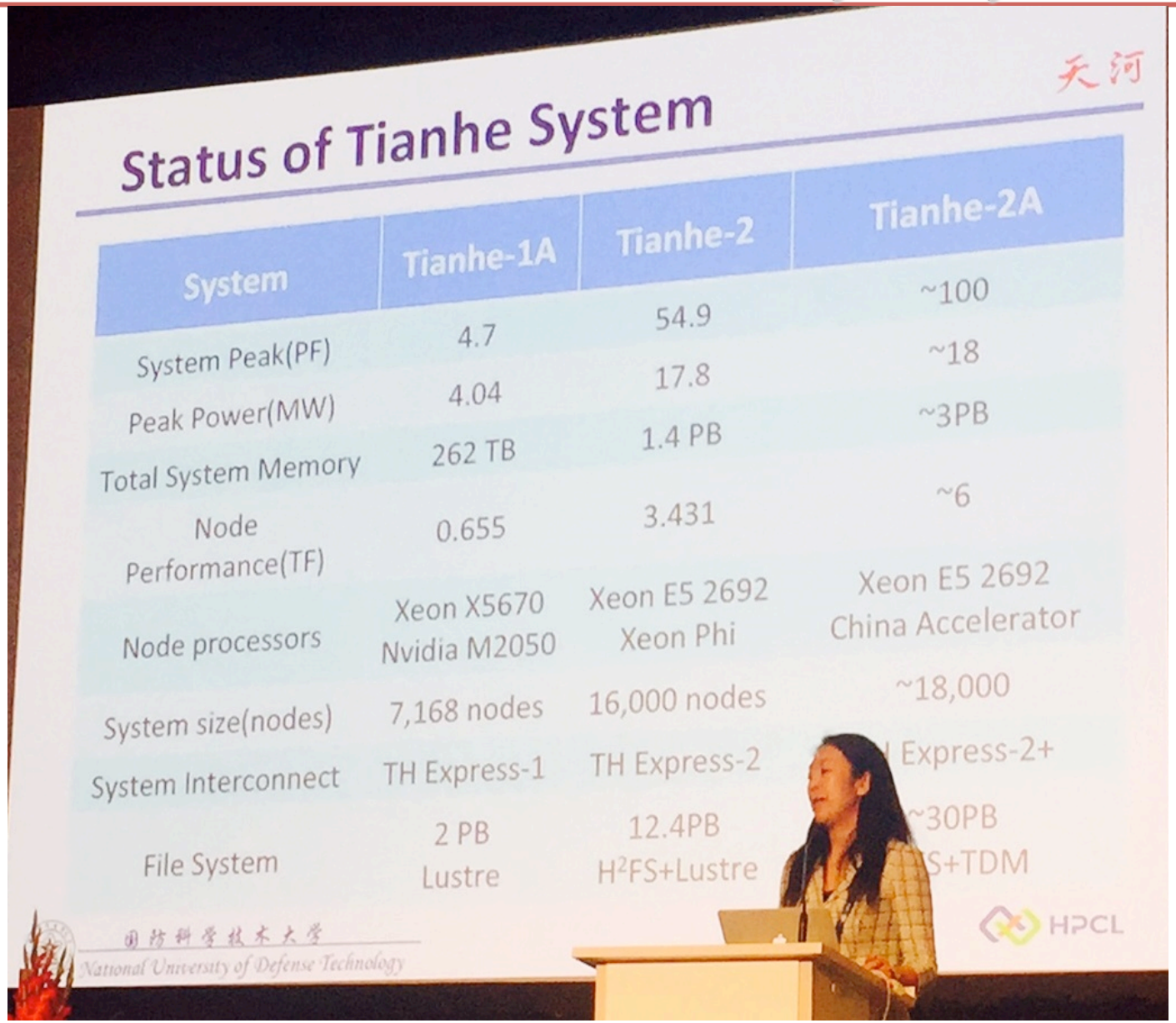
[illegible] ngz[illegible]

[illegible] njin[illegible]

[illegible] Def[illegible]

- National SC Center Changsl[illegible]

Yutong Lu from NUDT at the International Supercomputer Conference in Germany in July

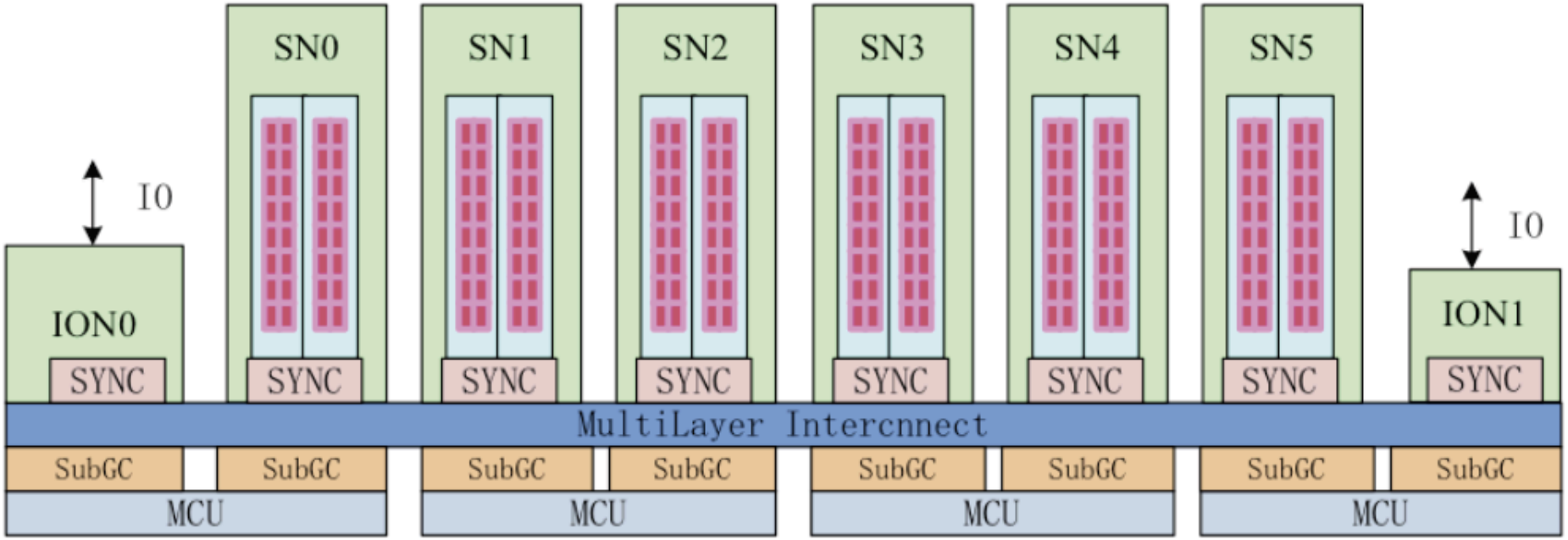


China Accelerator

天河

Matrix2000 GPDSP

- High Performance
  - 64bit Supported
  - ~2.4/4.8TFlops(DP/SP)
  - 1GHz, ~200W
- High Throughput
  - High-bandwidth Memory
  - 32~64GB
  - PCIE 3.0, 16x



China Other Machine

- Sunway Blue Light - Sunway BlueLight MPP, ShenWei processor SW1600 975.00 MHz, Infiniband QDR
  - Site:National Supercomputing Center in Jinan
  - Cores:137,200 Linpack Performance (Rmax)795.9 TFlop/s Theoretical Peak (Rpeak)1,070.16 TFlop/s
  - Processor:ShenWei processor SW1600 16C 975MHz (Alpha arch)
  - Interconnect:Infiniband QDR
- Rumored to have a 100 Pflop/s system

07

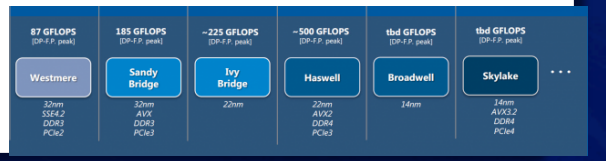
Peak Performance - Per Core

$$\text{FLOPS} = \text{cores} \times \text{clock} \times \frac{\text{FLOPs}}{\text{cycle}}$$

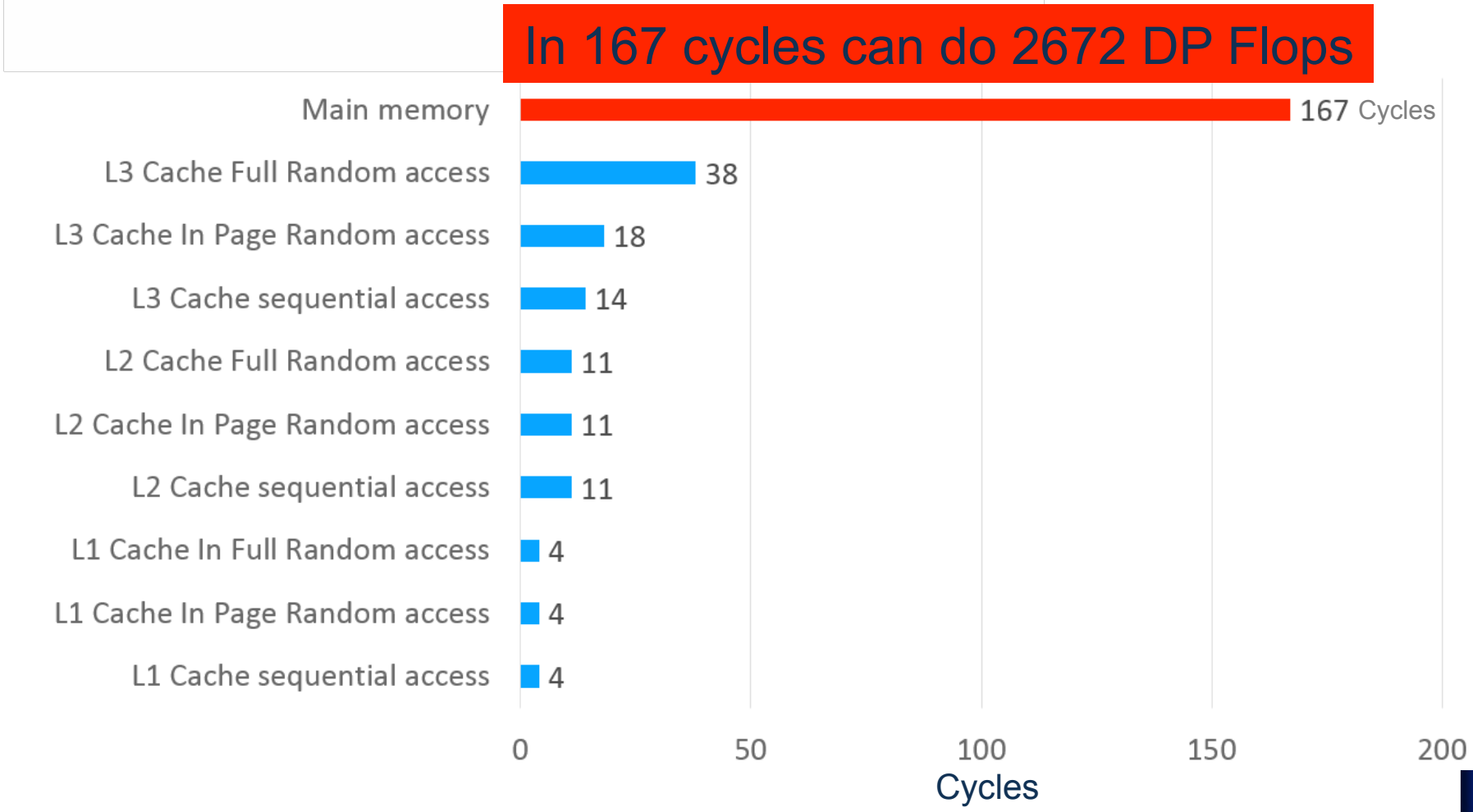
Floating point operations per cycle per core

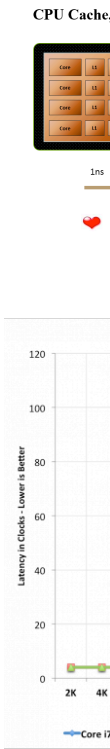
- Most of the recent computers have FMA (Fused multiple add): (i.e. x ←x + y*z in one cycle)
- Intel Xeon earlier models and AMD Opteron have SSE2
  - 2 flops/cycle DP & 4 flops/cycle SP
- Intel Xeon Nehalem ('09) & Westmere ('10) have SSE4
  - 4 flops/cycle DP & 8 flops/cycle SP
- Intel Xeon Sandy Bridge('11) & Ivy Bridge ('12) have AVX
  - 8 flops/cycle DP & 16 flops/cycle SP
- Intel Xeon Haswell ('13) & (Broadwell ('14)) AVX2
  - 16 flops/cycle DP & 32 flops/cycle SP
  - Xeon Phi (per core) is at 16 flops/cycle DP & 32 flops/cycle SP
- Intel Xeon Skylake (server) ('15) AVX 512
  - 32 flops/cycle DP & 64 flops/cycle SP

We are Here soon



CPU Access Latencies in Clock Cycles





Classical Analysis of Algorithms May Not be Valid

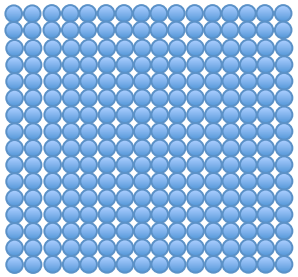
- Processors over provisioned for floating point arithmetic
- Data movement extremely expensive
- Operation count is not a good indicator of the time to solve a problem.
- Algorithms that do more ops may actually take less time.

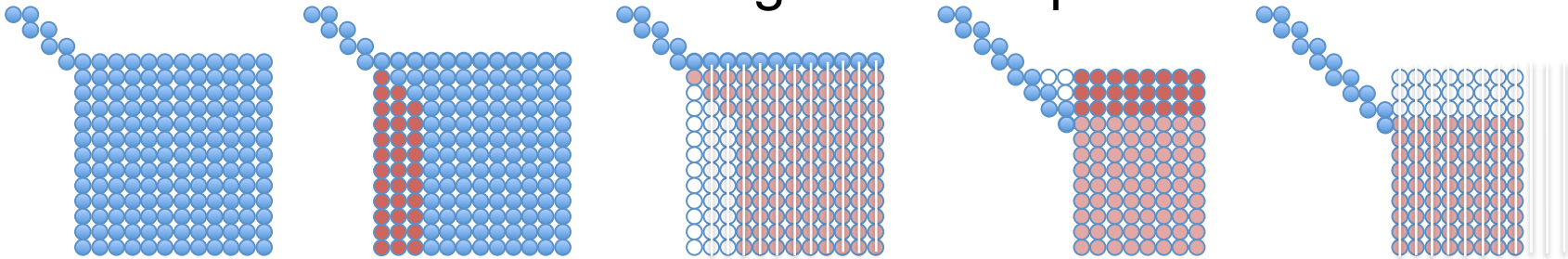
Singular Value Decomposition

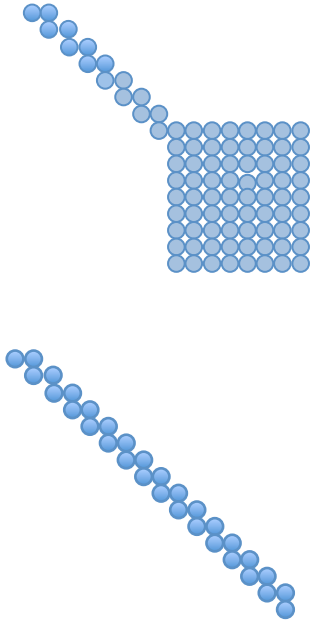
LAPACK Version 1991

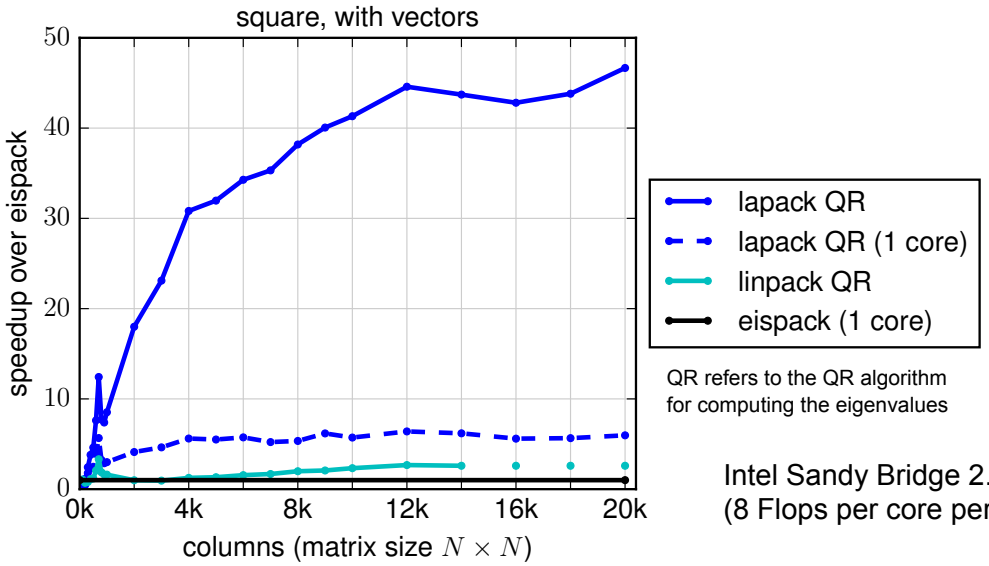
Level 1, 2, & 3 BLAS

First Stage 8/3 n^3 Ops









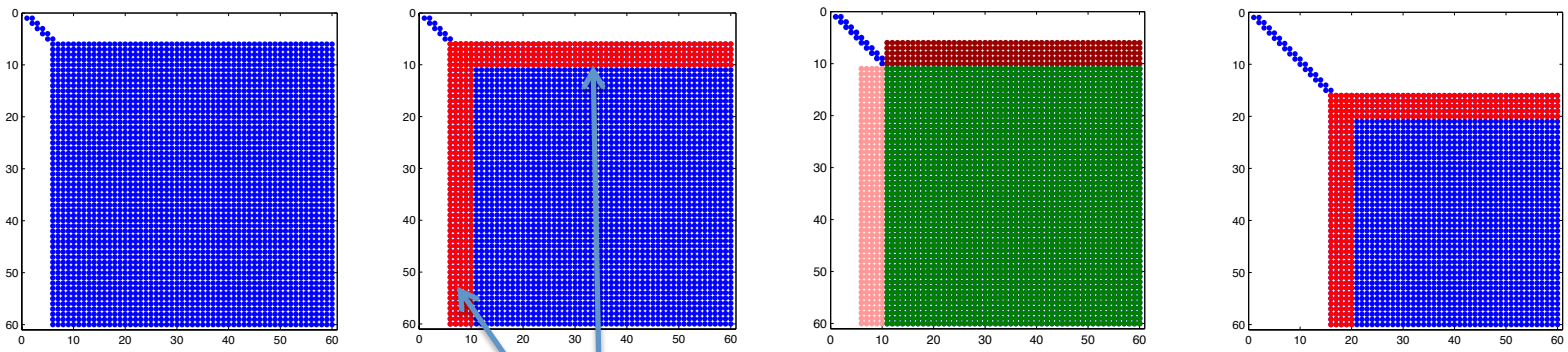
QR refers to the QR algorithm for computing the eigenvalues

Intel Sandy Bridge 2.6 GHz
(8 Flops per core per cycle)

Bottleneck in the Bidiagonalization

The Standard Bidiagonal Reduction: xGEBRD

Two Steps: Factor Panel & Update Tailing Matrix



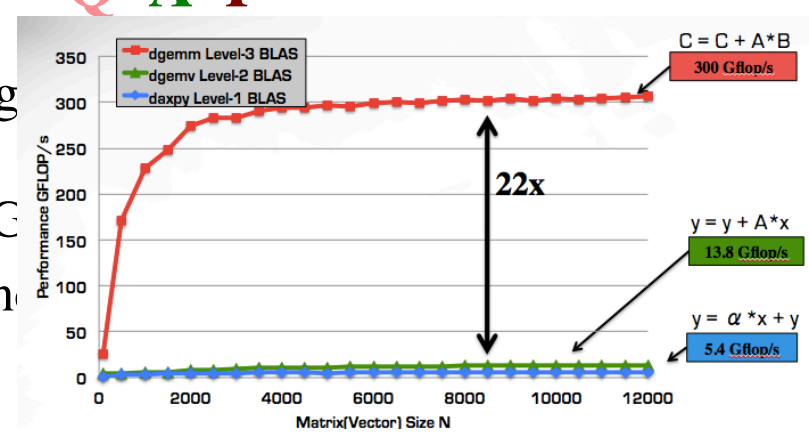
factor panel k then update ➔ **factor panel k+1**

Requires 2 GEMVs

✸ Characteristics

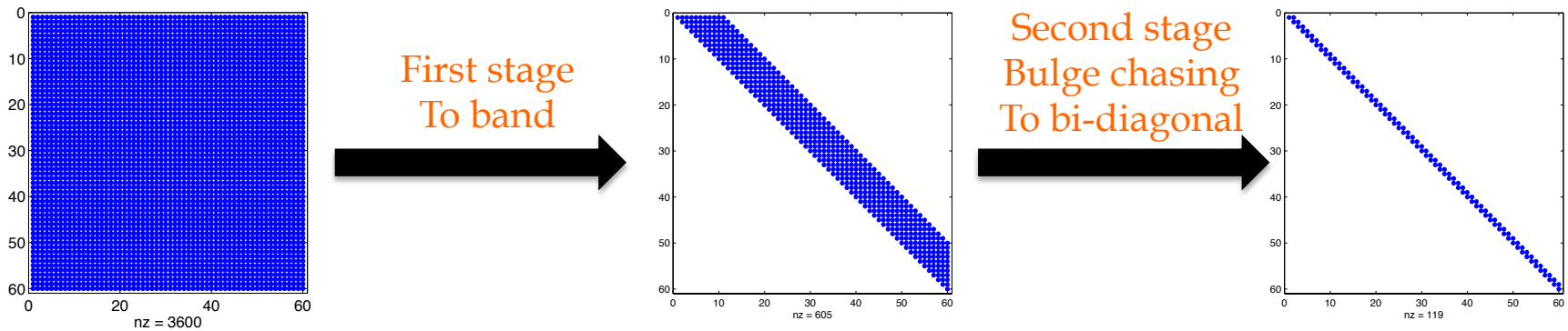
- Total cost $8n^3/3$, (reduction to bi-diag
- Too many Level 2 BLAS operations
- $4/3\ n^3$ from GEMV and $4/3\ n^3$ from G
- Performance limited to 2* performan
- ➔**Memory bound algorithm.**

Q*A*P^H



16 cores Intel Sandy Bridge, 2.6 GHz, 20 MB shared L3 cache.
The theoretical peak per core double precision is 20.4 Gflop/s per core.
Compiled with icc and using MKL 2015.3.187

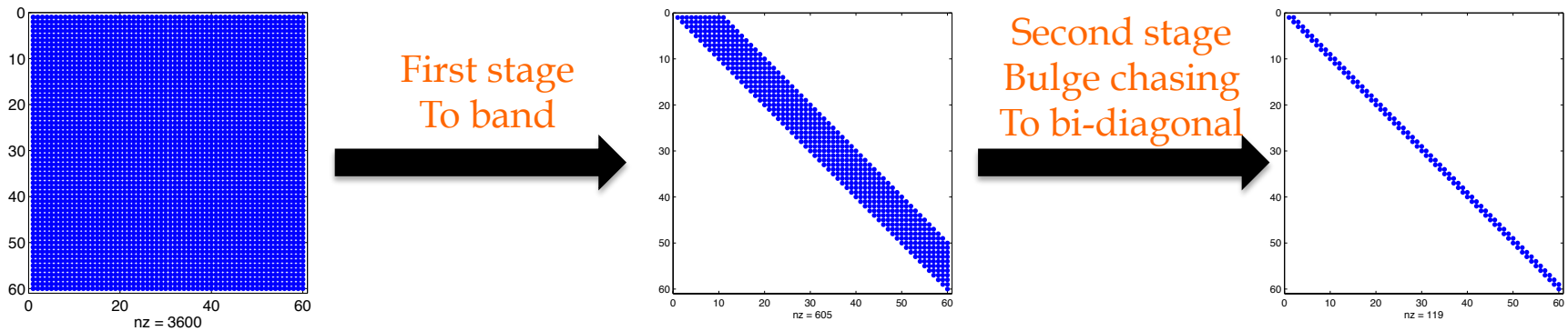
Recent Work on 2-Stage Algorithm



✸ Characteristics

- **Stage 1:**
  - Fully Level 3 BLAS
  - Dataflow Asynchronous execution
- **Stage 2:**
  - Level "BLAS-1.5"
  - Asynchronous execution
  - Cache friendly kernel (reduced communication)

Recent work on developing new 2-stage algorithm



$$\text{flops} \approx \sum_{s=1}^{\frac{n-n_b}{n_b}} 2n_b^3 + (nt-s)3n_b^3 + (nt-s)\frac{10}{3}n_b^3 + (nt-s)\times(nt-s)5n_b^3$$

$$+ \sum_{s=1}^{\frac{n-n_b}{n_b}} 2n_b^3 + (nt-s-1)3n_b^3 + (nt-s-1)\frac{10}{3}n_b^3 + (nt-s)\times(nt-s-1)5n_b^3$$

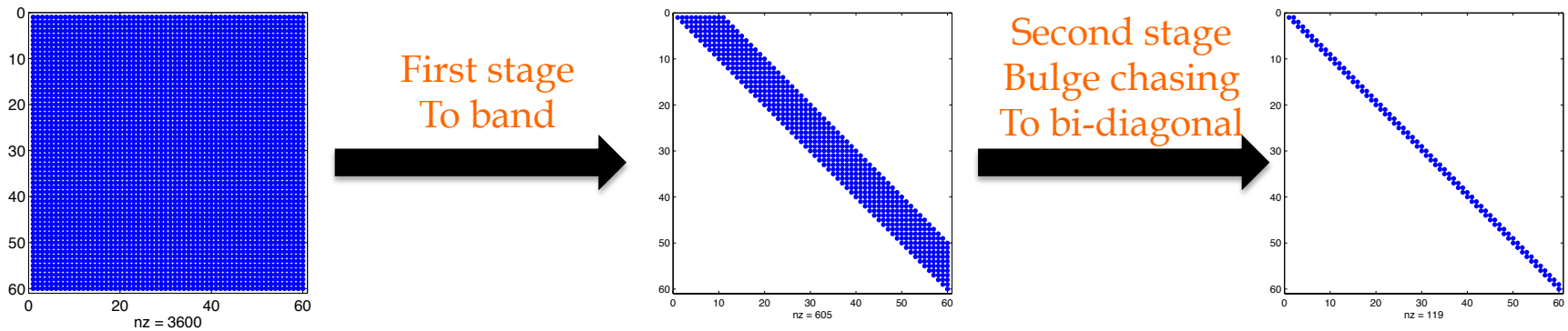
$$\approx \frac{10}{3}n^3 + \frac{10n_b}{3}n^2 + \frac{2n_b}{3}n^3$$

$$\approx \frac{10}{3}n^3(\text{gemm})_{\text{first stage}}$$

$$\text{flops} = 6\times n_b \times n^2(\text{gemv})_{\text{second stage}}$$

More Flops, original did 8/3 n ^{3}

Recent work on developing new 2-stage algorithm

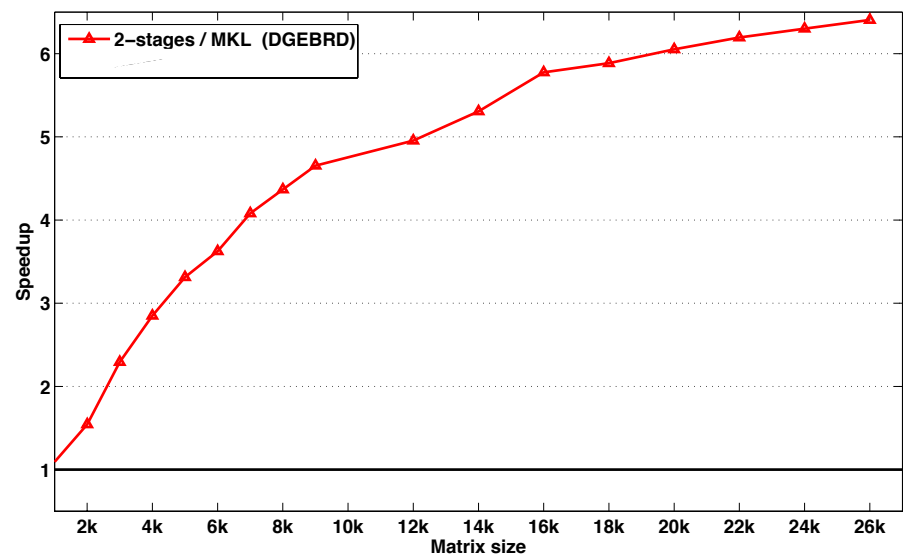


$$\textbf{speedup} = \frac{\textbf{time of one-stage}}{\textbf{time of two-stage}}$$

$$= \frac{4n^3/3P_{gemv} + 4n^3/3P_{gemm}}{10n^3/3P_{gemm} + 6n_b n^2/P_{gemv}}$$

$$\Longrightarrow \frac{84}{70} \leq \textbf{Speedup} \leq \frac{84}{15}$$

$$\Longrightarrow 1.8 \leq \textbf{Speedup} \leq 7$$

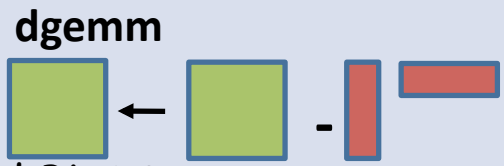


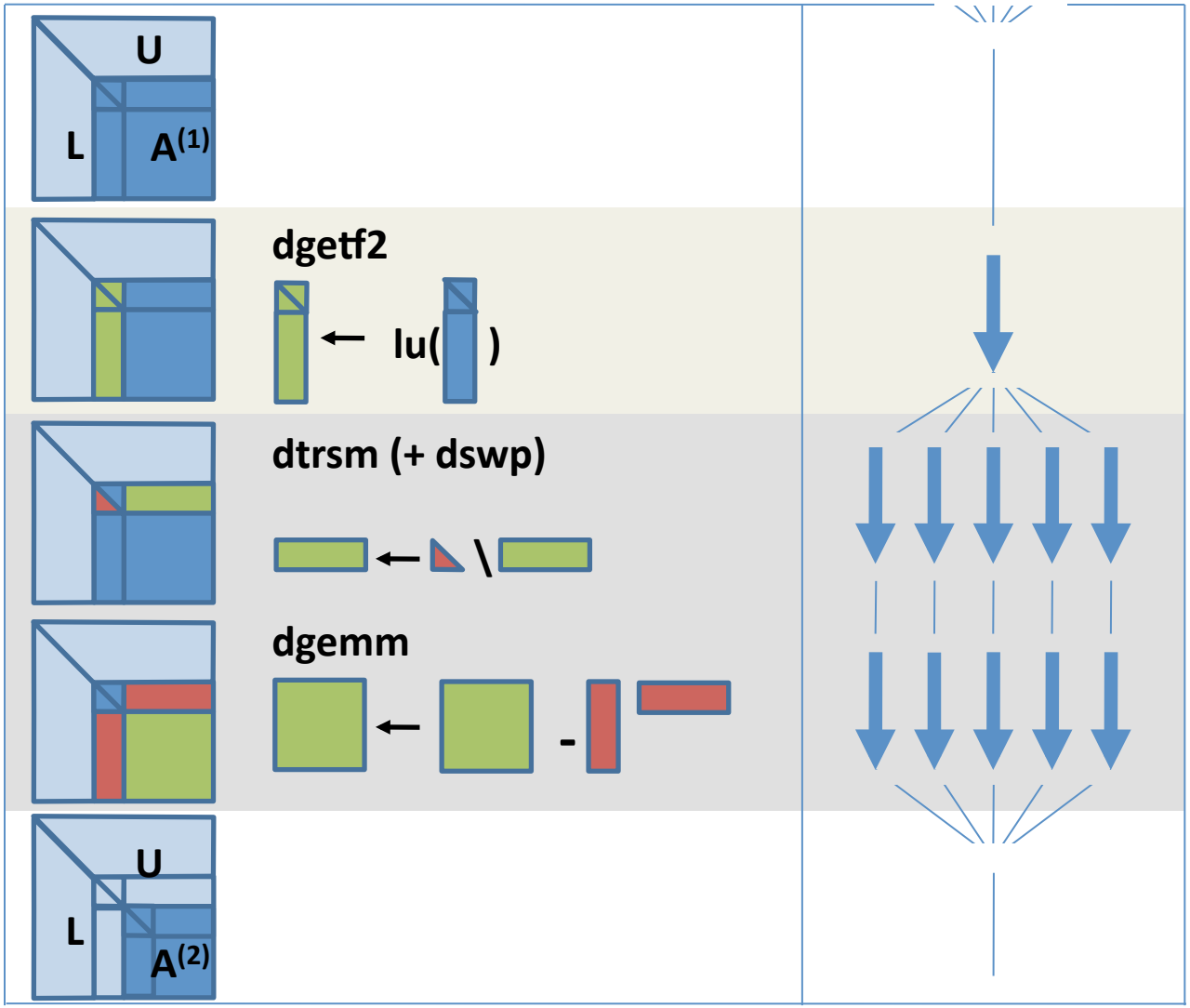
if P_{gemm} **is about 22x** P_{gemv} **and** $120 \leq n_b \leq 240$**.**

Parallelization of LU and QR.

Parallelize the update:

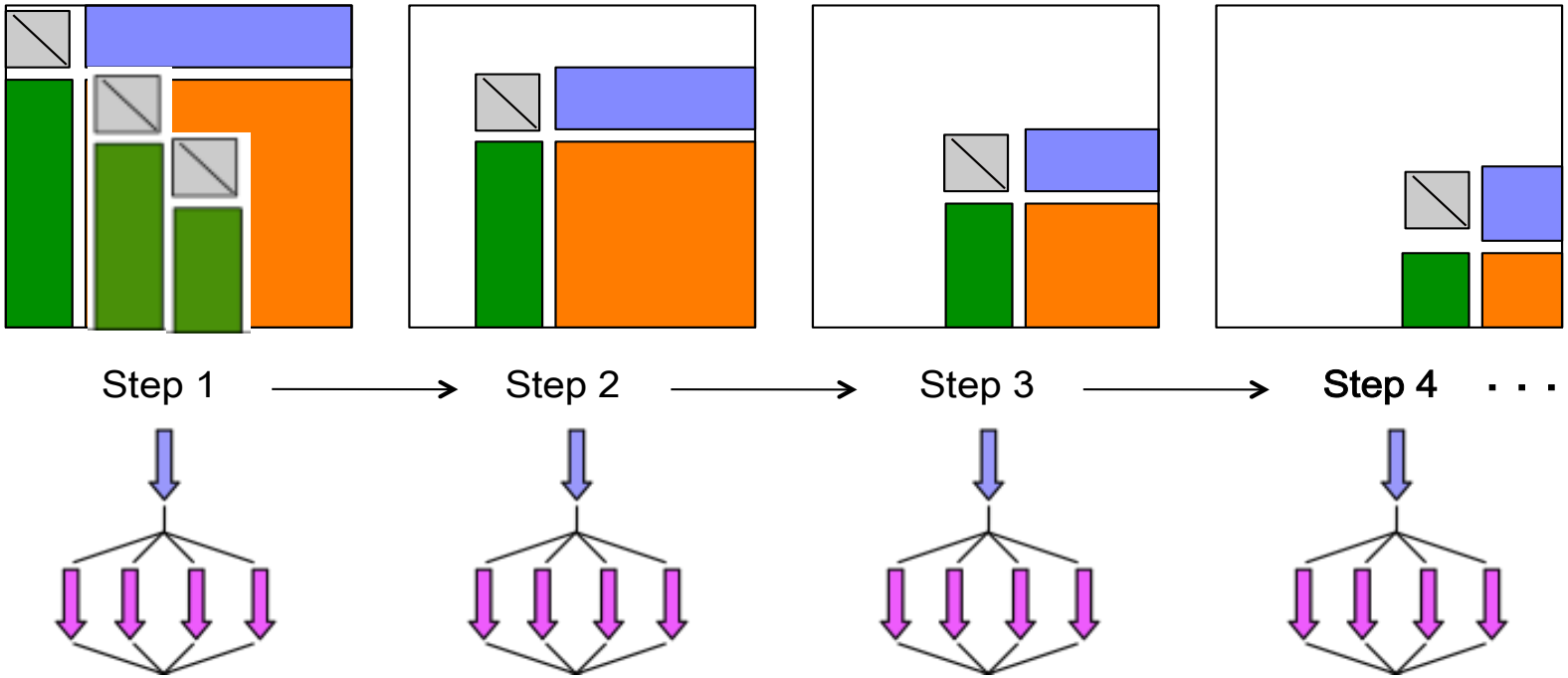
- Easy and done in any reasonable software.
- This is the $2/3n^3$ term in the FLOPs count.
- Can be done efficiently with LAPACK+multithreaded BLAS

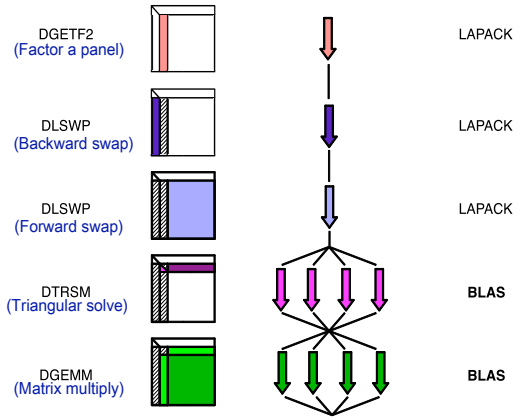




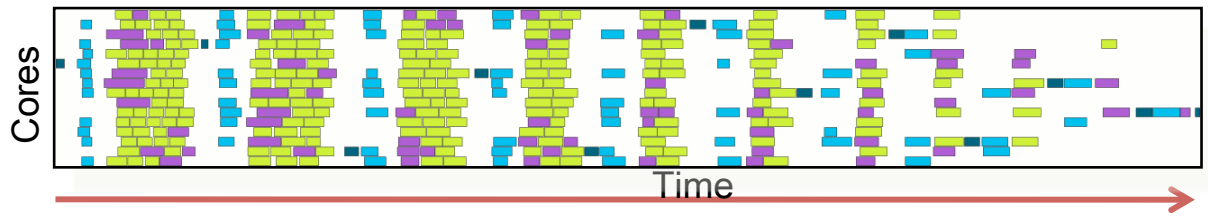
Fork - Join parallelism
Bulk Sync Processing

Synchronization (in LAPACK LU)





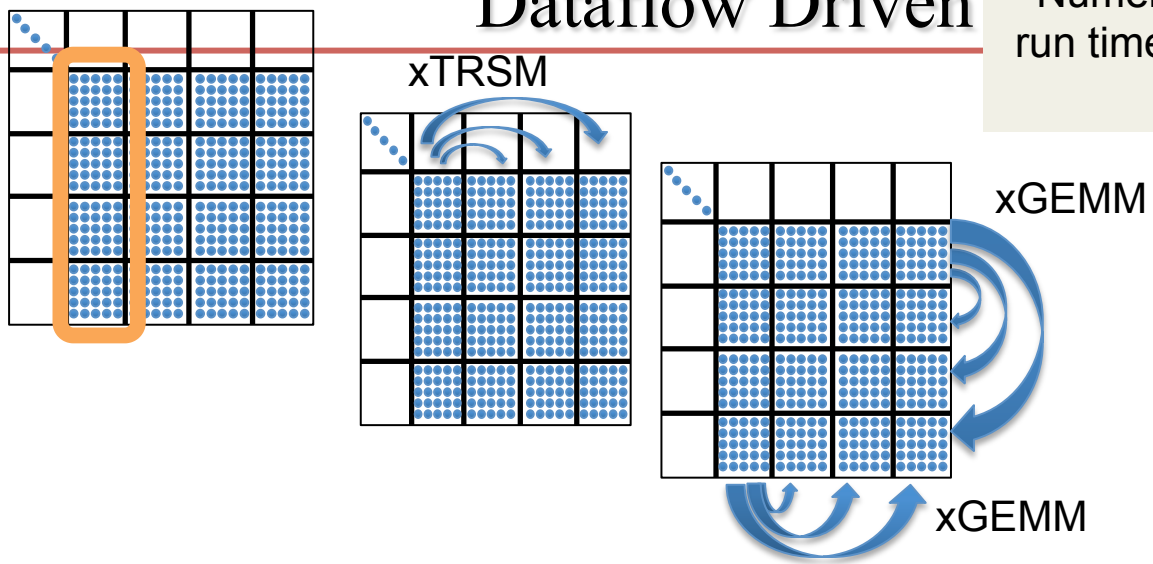
- fork join
- bulk synchronous processing



PLASMA LU Factorization

Dataflow Driven

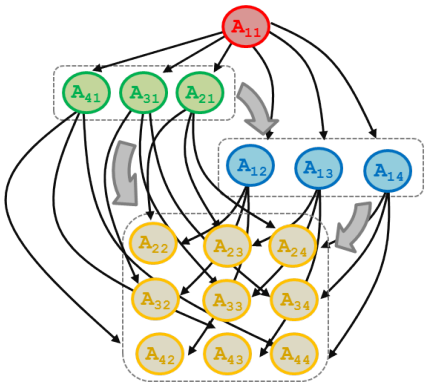
Numerical program generates tasks and run time system executes tasks respecting data dependences.



Sparse / Dense Matrix System

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

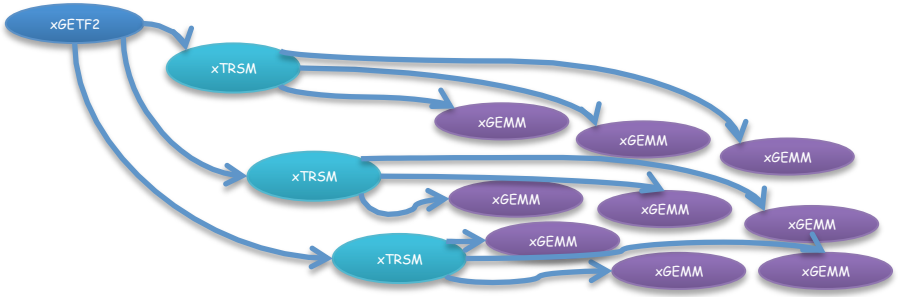
DAG-based factorization



Batched LA

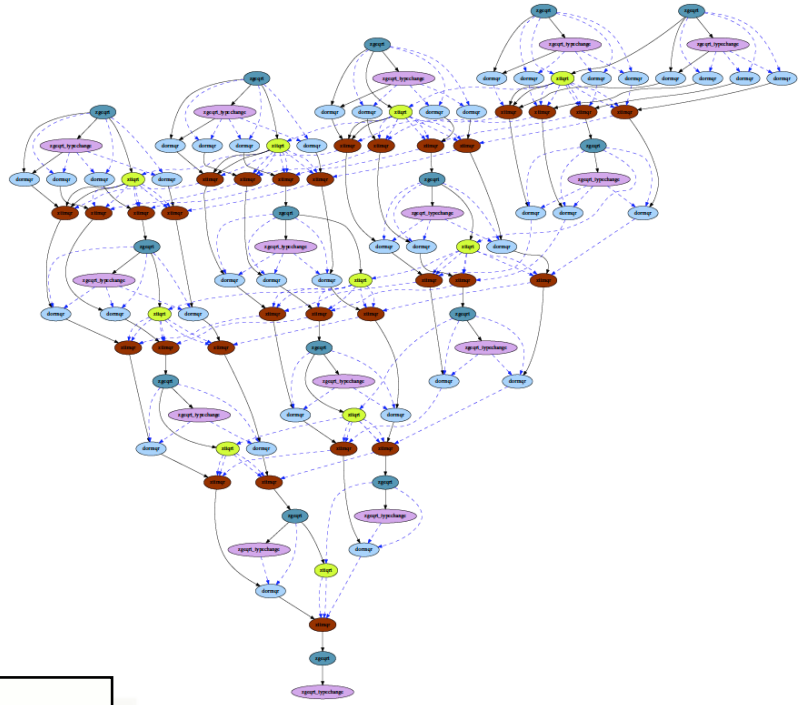
- **LU, QR,** or **Cholesky** on small diagonal matrices
- **TRSM**s, **QR**s, or **LU**s
- **TRSM**s, **TRMM**s
- **Updates (Schur complement) GEMM**s, **SYRK**s, **TRMM**s

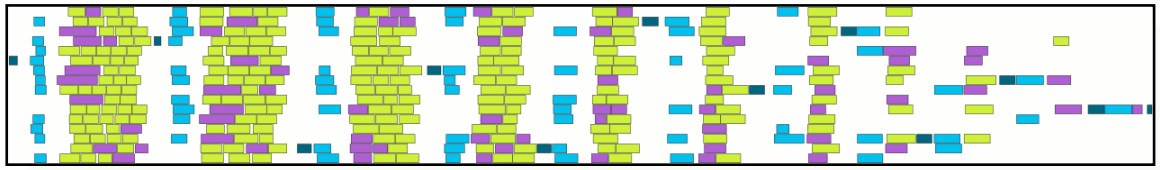
And many other BLAS/LAPACK, e.g., for application specific solvers, preconditioners, and matrices

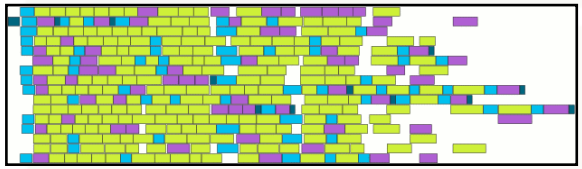


Dataflow Based Design

- Objectives
  - High utilization of each core
  - Scaling to large number of cores
  - Synchronization reducing algorithms
- Methodology
  - Dynamic DAG scheduling
  - Explicit parallelism
  - Implicit communication
  - Fine granularity / block data layout
- Arbitrary DAG with dynamic scheduling





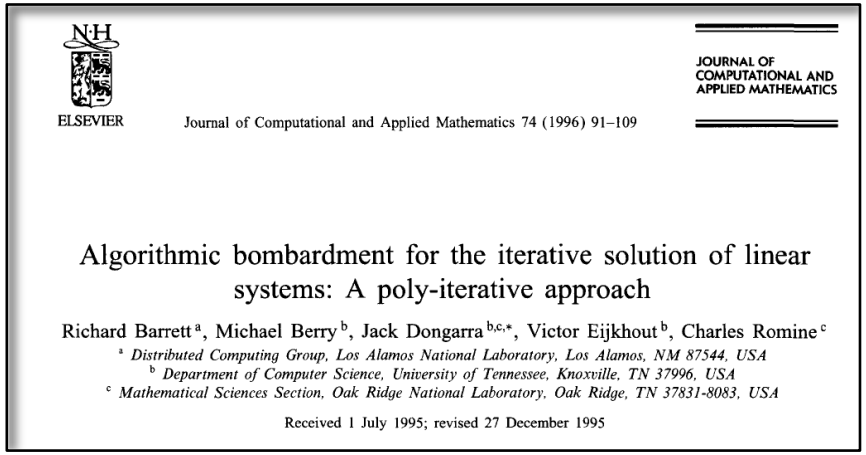


DAG scheduled parallelism

Fork-join parallelism
Notice the synchronization penalty in the presence of heterogeneity.

Algorithmic Bombardment

- Interesting in solving a large sparse non-symmetric system on a parallel computer.
- Many algorithms to choose
  - CG Square
  - Bi CG Stabilized
  - QMR
  - GMRES
  - etc.
- Best choice not clear
- Some may not converge
- Develop a "poly-algorithm" combining multiple algorithms in one, leveraging data movement.



ELSEVIER

Journal of Computational and Applied Mathematics 74 (1996) 91–109

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Algorithmic bombardment for the iterative solution of linear systems: A poly-iterative approach

Richard Barrett [a], Michael Berry [b], Jack Dongarra [b,c,*], Victor Eijkhout [b], Charles Romine [c]

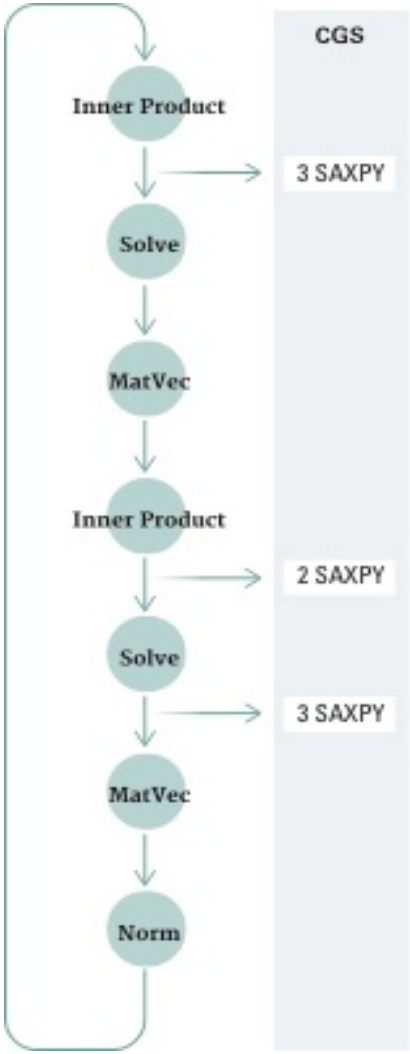
[a] *Distributed Computing Group, Los Alamos National Laboratory, Los Alamos, NM 87544, USA*

[b] *Department of Computer Science, University of Tennessee, Knoxville, TN 37996, USA*

[c] *Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831-8083, USA*

Received 1 July 1995; revised 27 December 1995

Poly-algorithm Iterating in Lock Step



Bombardment:
Combine three iterative methods into multi-iterative solver.

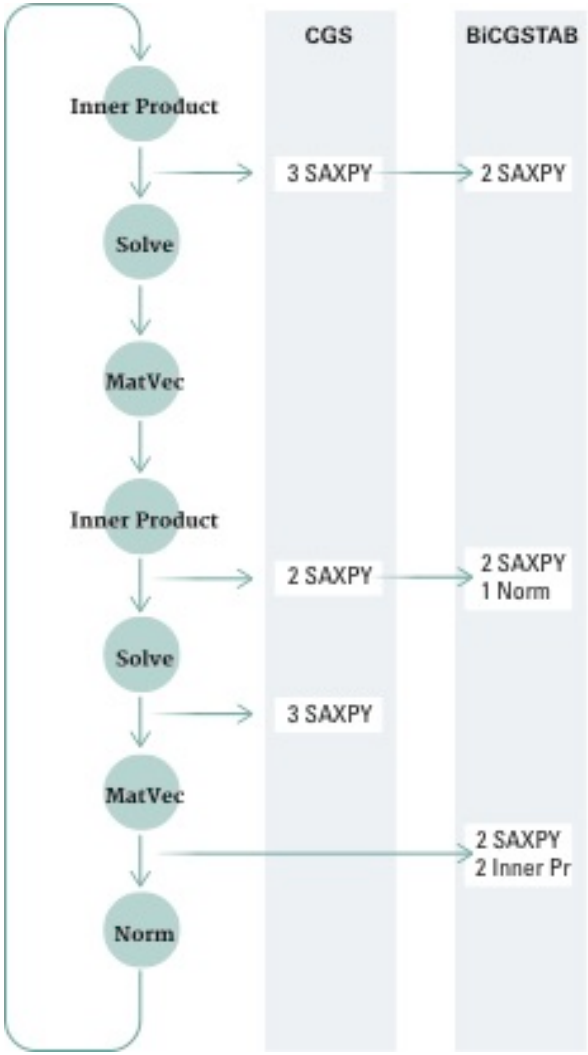
Amount of work / iteration

Method	$\alpha = x^T y$	$y = \alpha x + y$	$y = Ax$
CGS	2	6	2

Number of Communications / iteration

Method	$\alpha = x^T y$	$y = \alpha x + y$	$y = Ax$	$x = M^{-1} y$	Storage
CGS	2	0	2	2	A + 6n

Poly-algorithm Iterating in Lock Step



Bombardment:
Combine three iterative methods into multi-iterative solver.

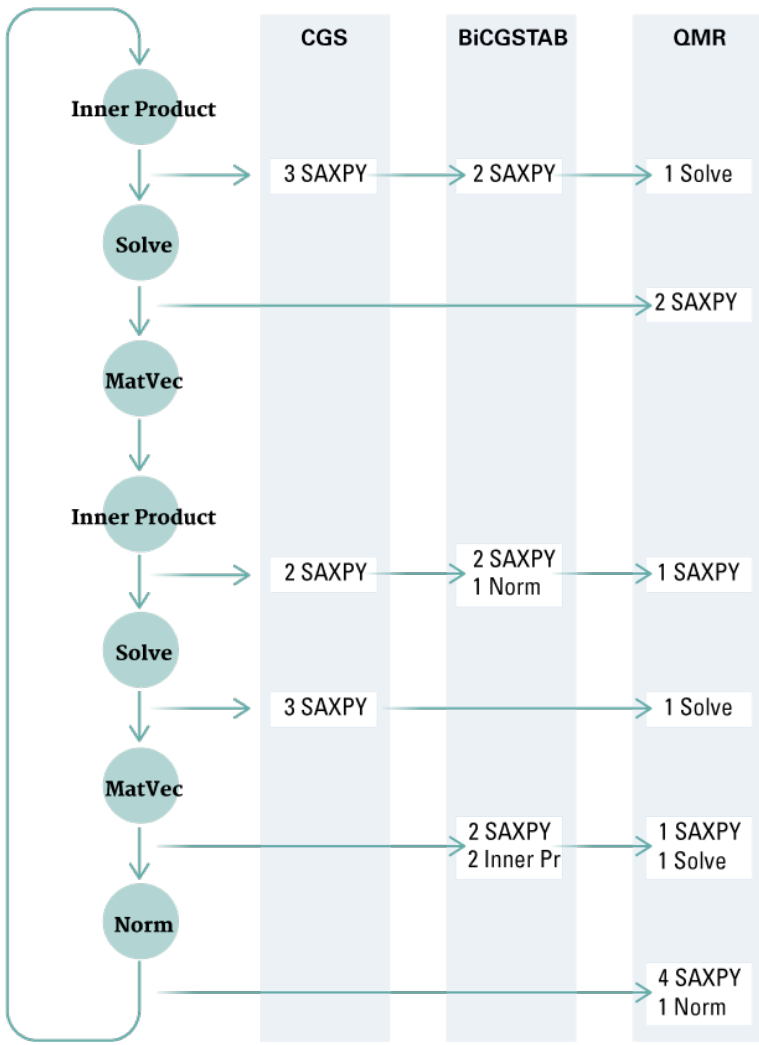
Amount of work / iteration

Method	$\alpha = x^T y$	$y = \alpha x + y$	$y = Ax$
CGS	2	6	2
Bi-CGSTAB	4	6	2

Number of Communications / iteration

Method	$\alpha = x^T y$	$y = \alpha x + y$	$y = Ax$	$x = M^{-1}y$	Storage
CGS	2	0	2	2	A + 6n
Bi-CGSTAB	3	0	2	2	A + 6n
Bombardment	3	0	2	3	A + 26n

Poly-algorithm Iterating in Lock Step



Bombardment:
Combine three iterative methods into multi-iterative solver.

Amount of work / iteration

Method	$\alpha = x^T y$	$y = \alpha x + y$	$y = Ax$
CGS	2	6	2
Bi-CGSTAB	4	6	2
QMR	2	12	2

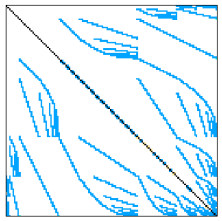
Number of Communications / iteration

Method	$\alpha = x^T y$	$y = \alpha x + y$	$y = Ax$	$x = M^{-1}y$	Storage
CGS	2	0	2	2	A + 6n
Bi-CGSTAB	3	0	2	2	A + 6n
QMR	2	0	2	2	A + 16n
Bombardment	3	0	2	3	A + 26n

Algorithmic Bombardment using Multi-Iterative Method

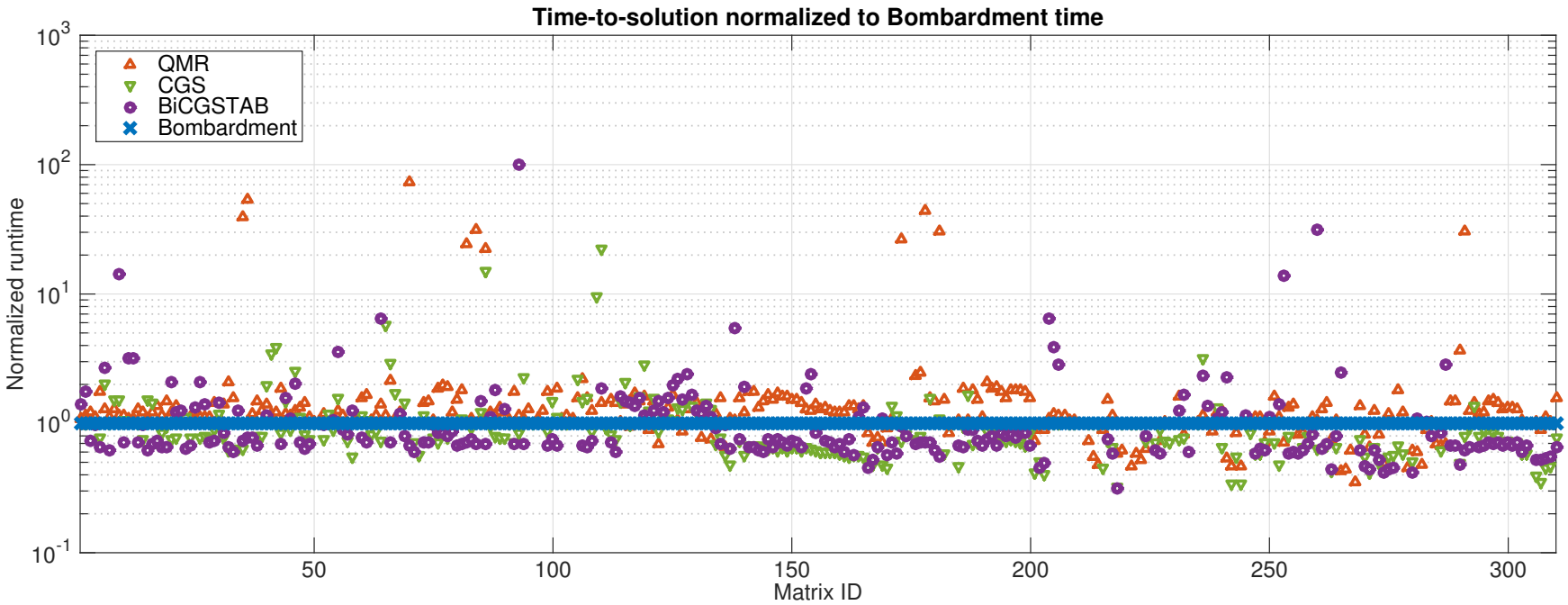
- Which is the iteration method of choice for a given problem?
- Combine three iterative solvers combined into multi-iterative solver: Bombardment combines CGS, QMR, BiCGSTAB in one method

- Target architecture: Graphics processing units (NVIDIA K40 GPU)
- Bombardment can be optimized for this architecture :
  - Benefits from blocking the distinct SpMV into an SpMM
  - Benefits from blocking the global reductions
  - Benefits from overlapping kernels
- 311 Sparse matrices from the University of Florida Matrix Collection
  - One of the solvers converges with < 5,000 iterations
  - Iteration limit: 100,000 iterations
  - Runtime is target metric



https://www.cise.ufl.edu/research/sparse/matrices/

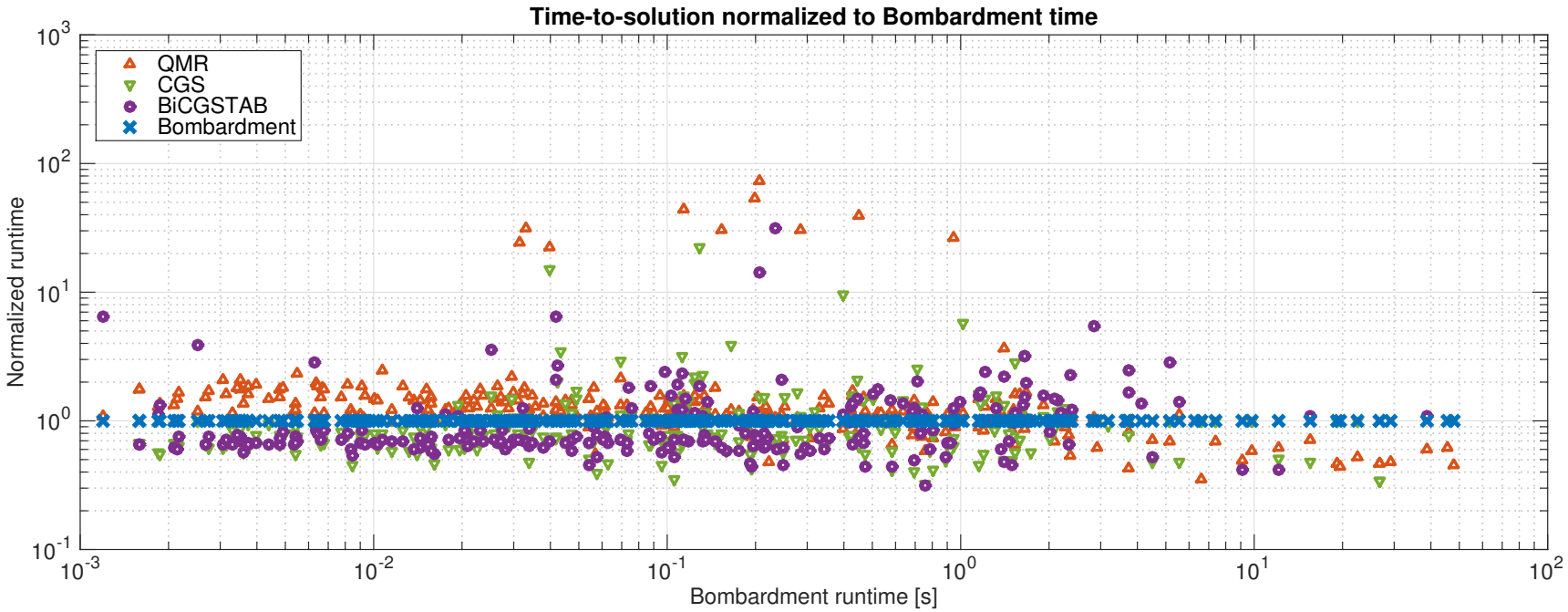
Algorithmic Bombardment using Multi-Iterative Method



Stats for 311 test matrices	QMR		CGS		BiCGSTAB	
Fastest method:	38	(12%)	93	(30%)	181	(58%)
Converges:	299	(96%)	262	(84%)	248	(80%)
Fails:	12	(4%)	49	(16%)	63	(20%)
Bombardment Speedup*	2.42 x		1.11 x		1.66 x	

*only considering converging test cases

Algorithmic Bombardment using Multi-Iterative Method



Stats for 311 test matrices	QMR	CGS	BiCGSTAB
Fastest method:	38 (12%)	93 (30%)	181 (58%)
Converges:	299 (96%)	262 (84%)	248 (80%)
Fails:	12 (4%)	49 (16%)	63 (20%)
Bombardment Speedup*	2.42 x	1.11 x	1.66 x

*only considering converging test cases

Avoiding Synchronization

- "Responsibly Reckless" Algorithms
  - Try fast algorithm (unstable algorithm) that might fail (but rarely)
  - Check for instability
  - If needed, recompute with stable algorithm

Avoid Pivoting; Minimize Synchronization

- To solve $Ax = b$:
  - Compute $A_r = U^T A V$, with U and V random matrices
  - Factorize A_r without pivoting (GENP)
  - Solve $A_r\, y = U^T b$ and then Solve $x = Vy$
- U and V are Recursive Butterfly Matrices
  - Randomization is cheap ($O(n^2)$ operations)
  - GENP is fast (can take advantage of the GPU)
  - Accuracy is in practice similar to GEPP (with iterative refinement), but…

Think of this as a preconditioner step.

Goal: Transform A into a matrix that would be sufficiently "random" so that, with a probability close to 1, pivoting is not needed.

A **butterfly matrix** is defined as any n-by-n matrix of the form:

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R & S \\ R & -S \end{pmatrix}$$

where R and S are random diagonal matrices.

$B =$

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.

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

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

37

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.

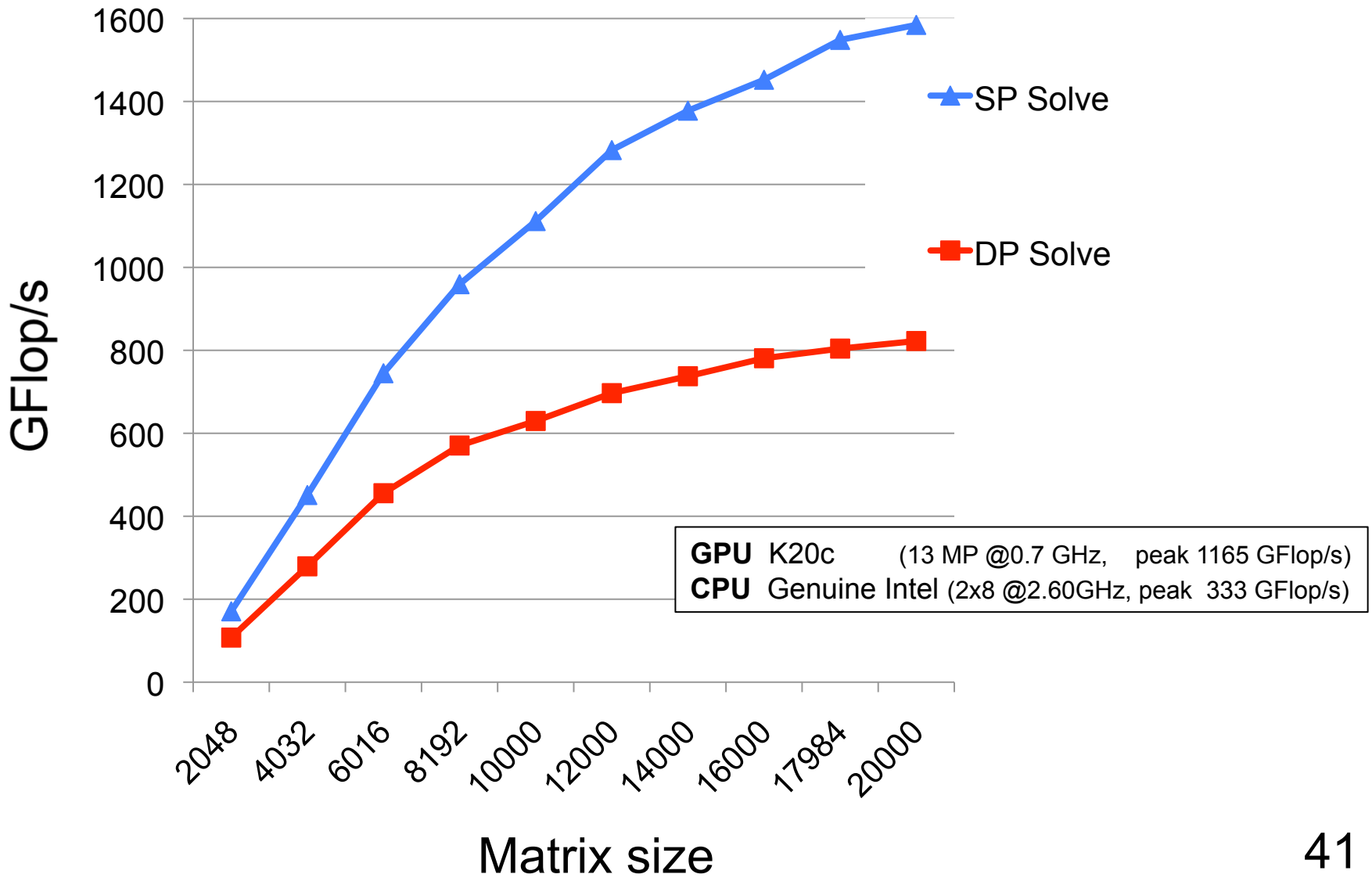
L U = lu(A)	SINGLE	$O(n^3)$
x = L\(U\b)	SINGLE	$O(n^2)$
r = b – Ax	DOUBLE	$O(n^2)$
WHILE || r || not small enough		
z = L\(U\r)	SINGLE	$O(n^2)$
x = x + z	DOUBLE	$O(n^1)$
r = b – Ax	DOUBLE	$O(n^2)$
END		

  - 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; $O(10^8)$

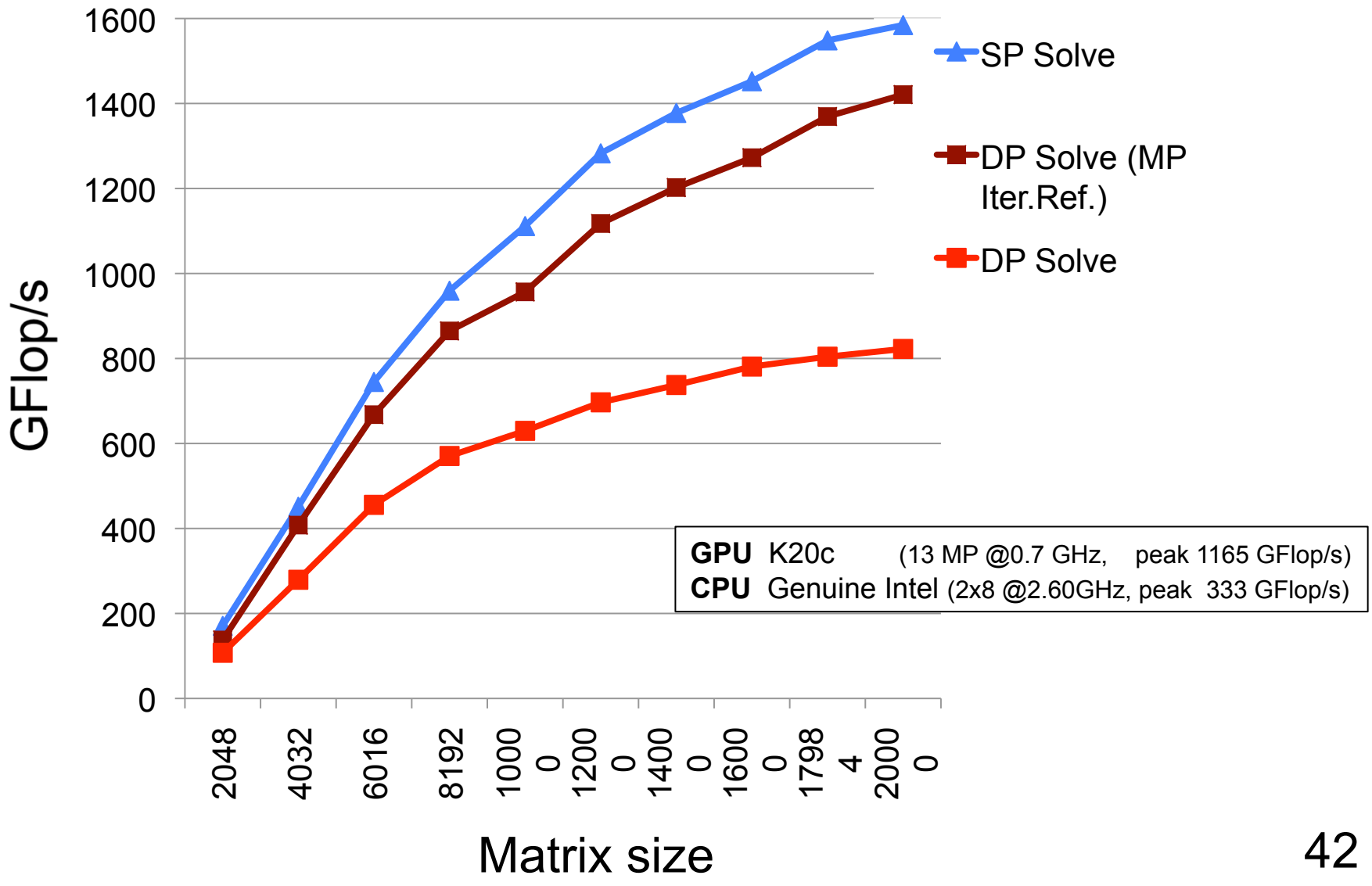
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement

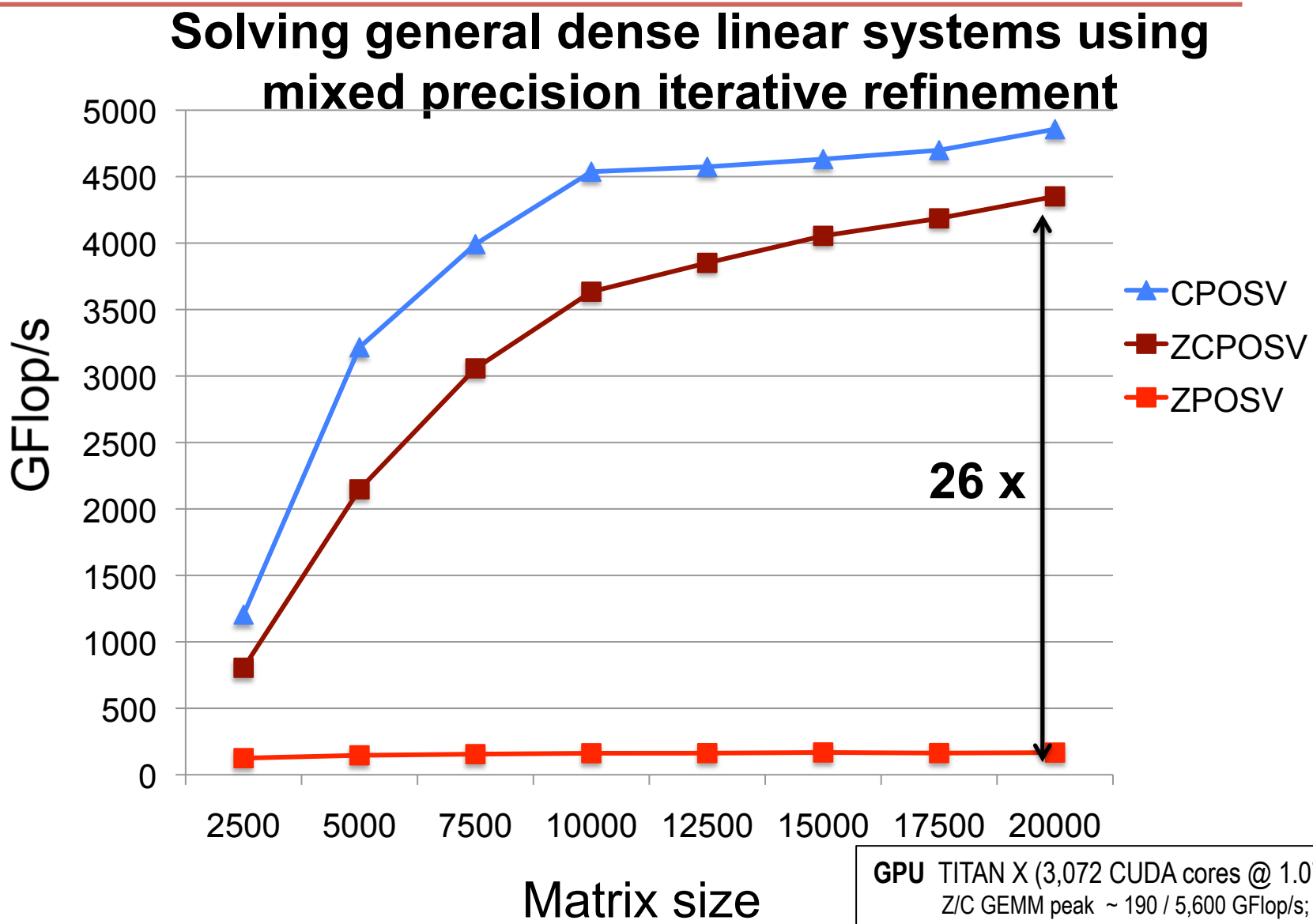


Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement

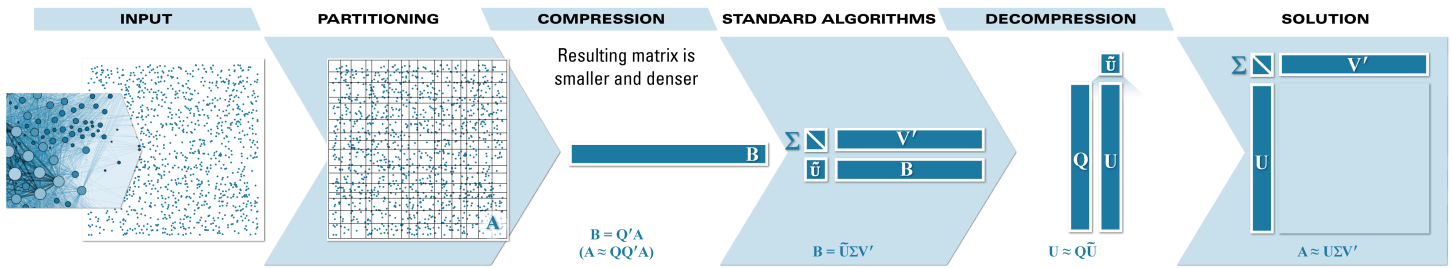


Mixed precision iterative refinement



GPU TITAN X (3,072 CUDA cores @ 1.076 GHz)
Z/C GEMM peak ~ 190 / 5,600 GFlop/s; Maxwell
CPU Intel Xeon X5660@2.80GHz (2 x 6 cores)

Methods for Matrix Approximation via Random Sampling



- Random sampling algorithms address the size problem by generating much smaller approximate representations of large matrices
  - Then solved using classic linear algebra algorithms on platforms that would not otherwise be usable for this purpose
- Deliver results that are approximate solutions.
  - They are faster, easier to code, easier to map to modern computer architectures, often more numerically robust, and display a higher level of parallelism than traditional algorithms.

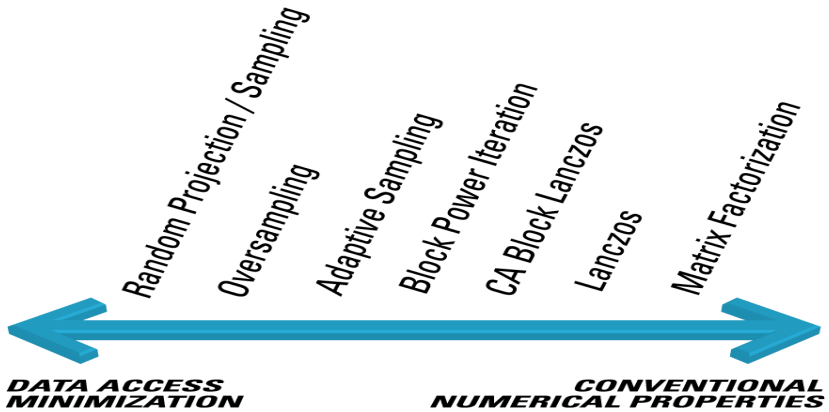
Computing rank-k approximation of A,

$$A \approx A_k = U_k V_k^T,$$

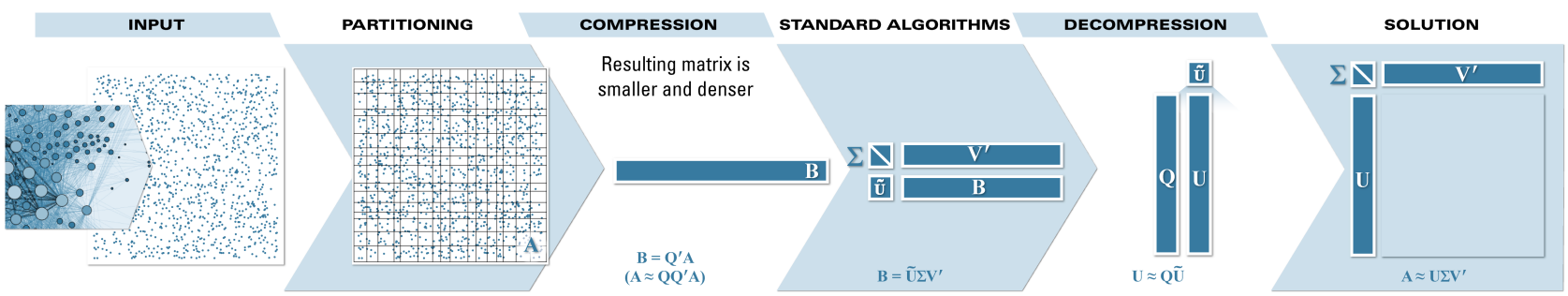
plays important roles in many applications with different constraints:

- SVD
  - $A_k = U_k \Sigma_k V_k^T$ to minimize $||A-A_k||$
  - for latent semantic indexing and population clustering [SC'15]
- CX, and then Rank-Reviling QR
  - $A_k = A(:,J_{k+l})X_{k+l}$, and then $AP \approx Q_k R_k$
  - with datasets from Hapmap project [SC'15]
- CUR
  - $A_k = A(:,J_k)U_k A(I_k,:)$ by ACA, etc.
  - Data analysis, linear solver, etc.
- Matrix completion:
  - Minimize $\Sigma_{i,j}(A(i,j)-A_k(i,j))^2$ for $A(i,j) \neq 0$ by ALS, SGD, etc.
  - Recommendation system, missing data, etc.
- etc.

- Different algorithms for computing approximations, e.g., for SVD



- Increasing interests in randomized algorithms
  - Demand from architecture (e.g., high cost of data access)
  - Demand from data (e.g., processing "BigData")
  - Aim to extract as much info as possible from each data access
  - May not work in all cases, good for low rank matrices where factorization is overkill.



Random sampling framework: normalized block power iteration

Generate P and Q from Q

do

2. SpMM + Ortho

   $\widehat{P} = AQ$, and

   $PR_p = \text{TSQR}(\widehat{P})$

3. Restart (if not done)

   $\widehat{Q} = A^T P$, and

   $QR_q = \text{TSQR}(\widehat{Q})$

while 'not done'

Extract SVD from projection

$R_p = X\Sigma Y^T$ with $R_p = P^T AQ$

$U_k = PX_k$ and $V_k = QY_k$

- Iterate to improve approximation
  - Sample A and A^T (e.g., SpMM) to approximate left and right singular vectors
- Block to improve convergence
  - Q had k+l columns with oversampling parameter l
- "normalize" to maintain stability
  - many options; e.g., different algorithms, one-sided

Random sampling framework: normalized block power iteration

- Potential to integrate different randomization/sampling schemes
  - Gaussian random sampling (SpMM/GEMM with Gaussian random vector Q)
  - "pruned" FFT instead of GEMM [SC'14]
  - Sampling based on probabilistic distribution
    (e.g., population clustering by SNP, leverage scores)
- Nice relation to "traditional algorithm"
  e.g., block (CA) Lanczos for bidiagonalization [IEEE BigData workshop'14]
- If sampling converges to desired accuracy in a couple of iterations,
  it could outperform others, especially if data access is expensive

Generate P and Q from Q

do

2. SpMM + Ortho
   $\widehat{P} = AQ$, and
   $PR_p = \text{TSQR}(\widehat{P})$
3. Restart (if not done)
   $\widehat{Q} = A^T P$, and
   $QR_q = \text{TSQR}(\widehat{Q})$

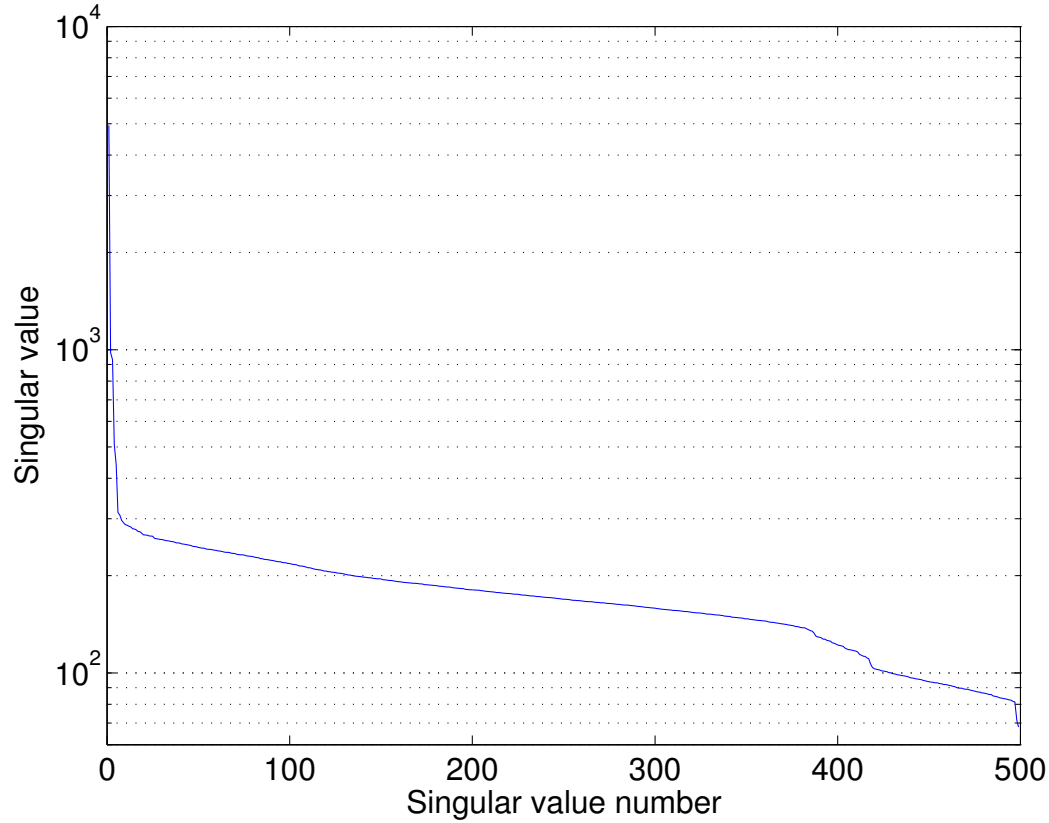
while 'not done'

Extract SVD from projection

$R_p = X\Sigma Y^T$ with $R_p = P^T AQ$

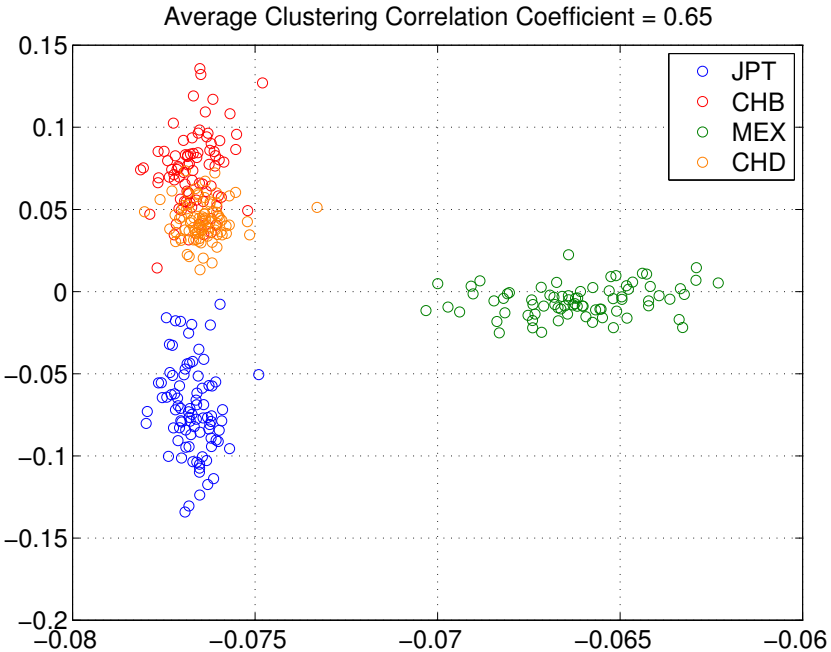
$U_k = PX_k$ and $V_k = QY_k$

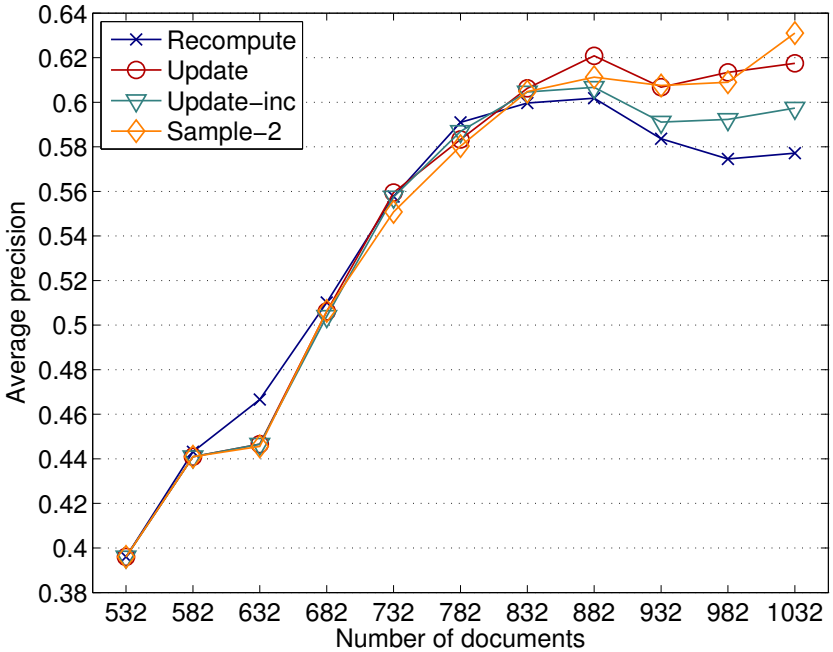
Singular value distribution of an SNP matrix (TS matrix)O(100Kx500)



- approximate "low-rank-plus-shift"
  - Big enough gap to obtain desired low accuracy in a couple of iterations
  - Small enough gap for stability (e.g., CholQR for othogonalization)
- many matrices of interests seem to have such distributions

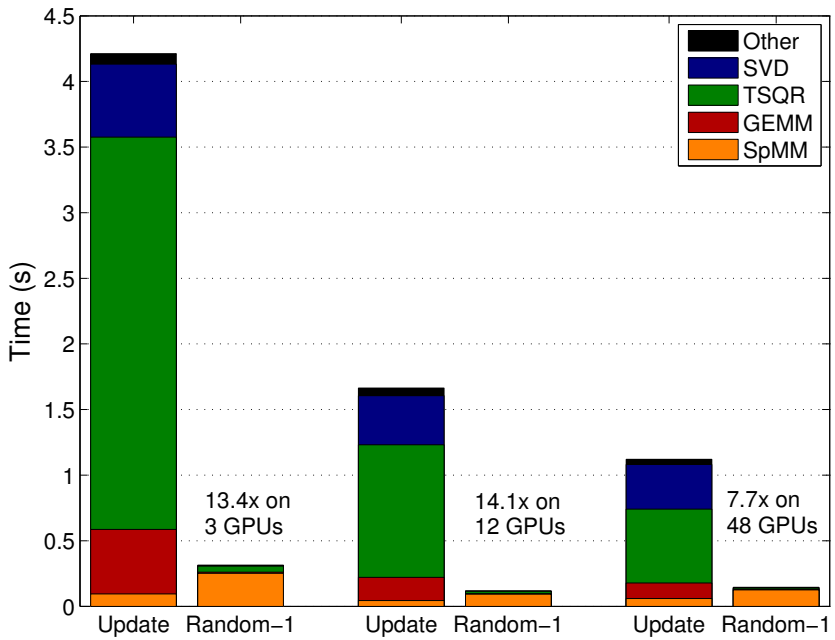
Case studies: population clustering and latent semantic indexing

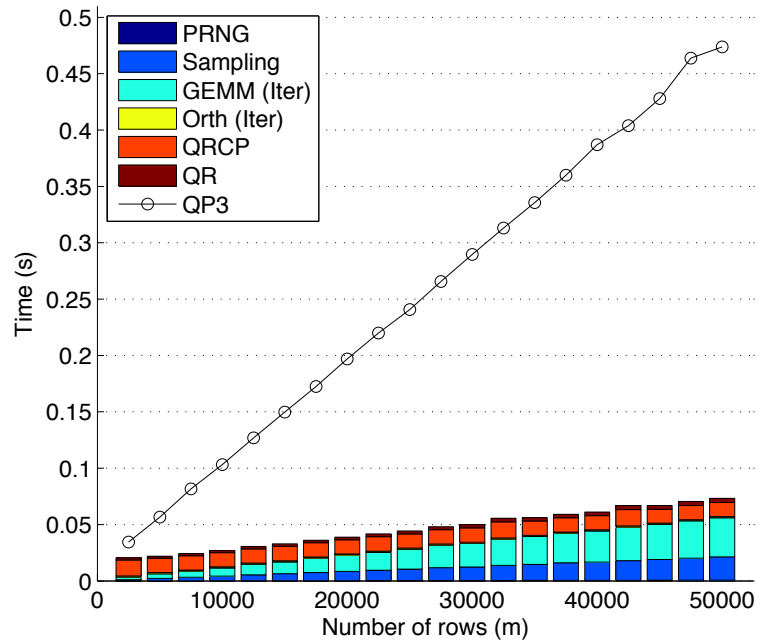




- Desired accuracy obtained after a couple of iterations

Performance studies on hybrid CPU/GPU architecture

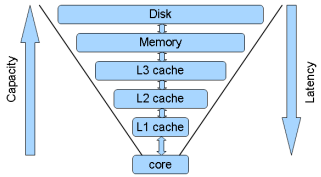


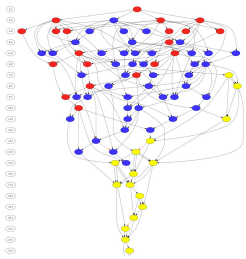


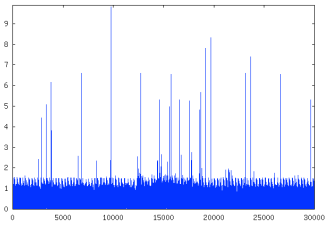
- Sparse truncated SVD on Tsubame Supercomputer: each node with two six-core Intel Xeon CPUs and three NVIDIA Tesla K20Xm GPUs
- Dense truncated RRQR on one node with two eight-core Intel SandyBridge CPUs and three NVIDIA Tesla K40c GPUs

Software and Algorithm Must Keep Pace with the Changes in Hardware

- Classical analysis of algorithms is not valid,
  - # of floating point ops ≠ computation time.
- Algorithms and software must take advantage by reducing data movement.
  - Need latency tolerance in our algorithms
- Communication and synchronization reducing algorithms and software are critical.
  - As parallelism grows
- Hardware presents a dynamically changing environment
  - Turbo Boost and OS jitter
- Many existing algorithms can't fully exploit the features of modern architecture







Summary

- Major Challenges are ahead for extreme computing
  - Parallelism $O(10^9)$
    - Programming issues
  - Hybrid
    - Peak and HPL may be very misleading
    - No where near close to peak for most apps
  - Fault Tolerance
    - Today Sequoia BG/Q node failure rate is 1.25 failures/day
  - Power
    - 50 Gflops/w (today at 2 Gflops/w)
- We will need completely new approaches and technologies to reach the Exascale level