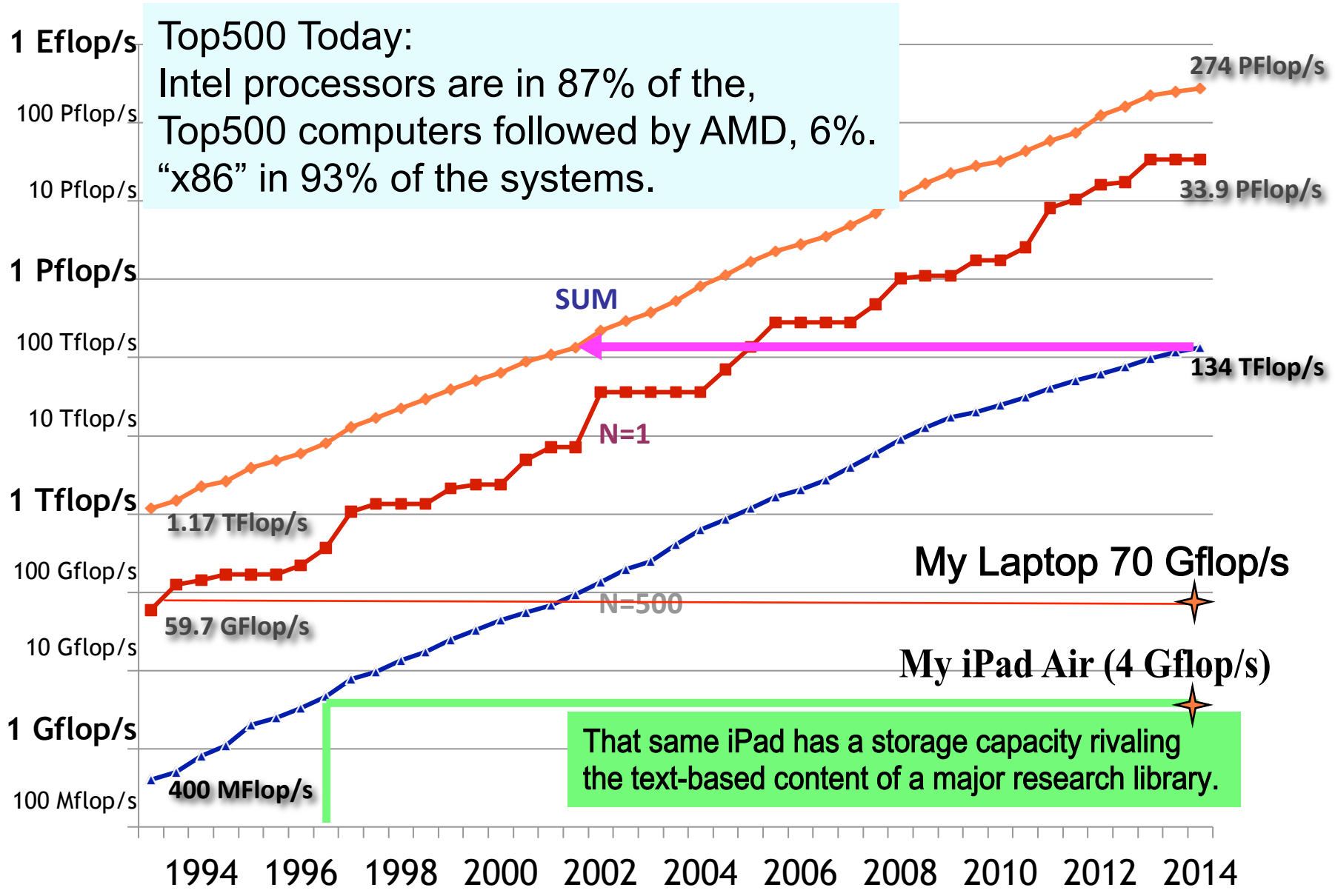
Algorithmic and Software Challenges at Extreme Scales

Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

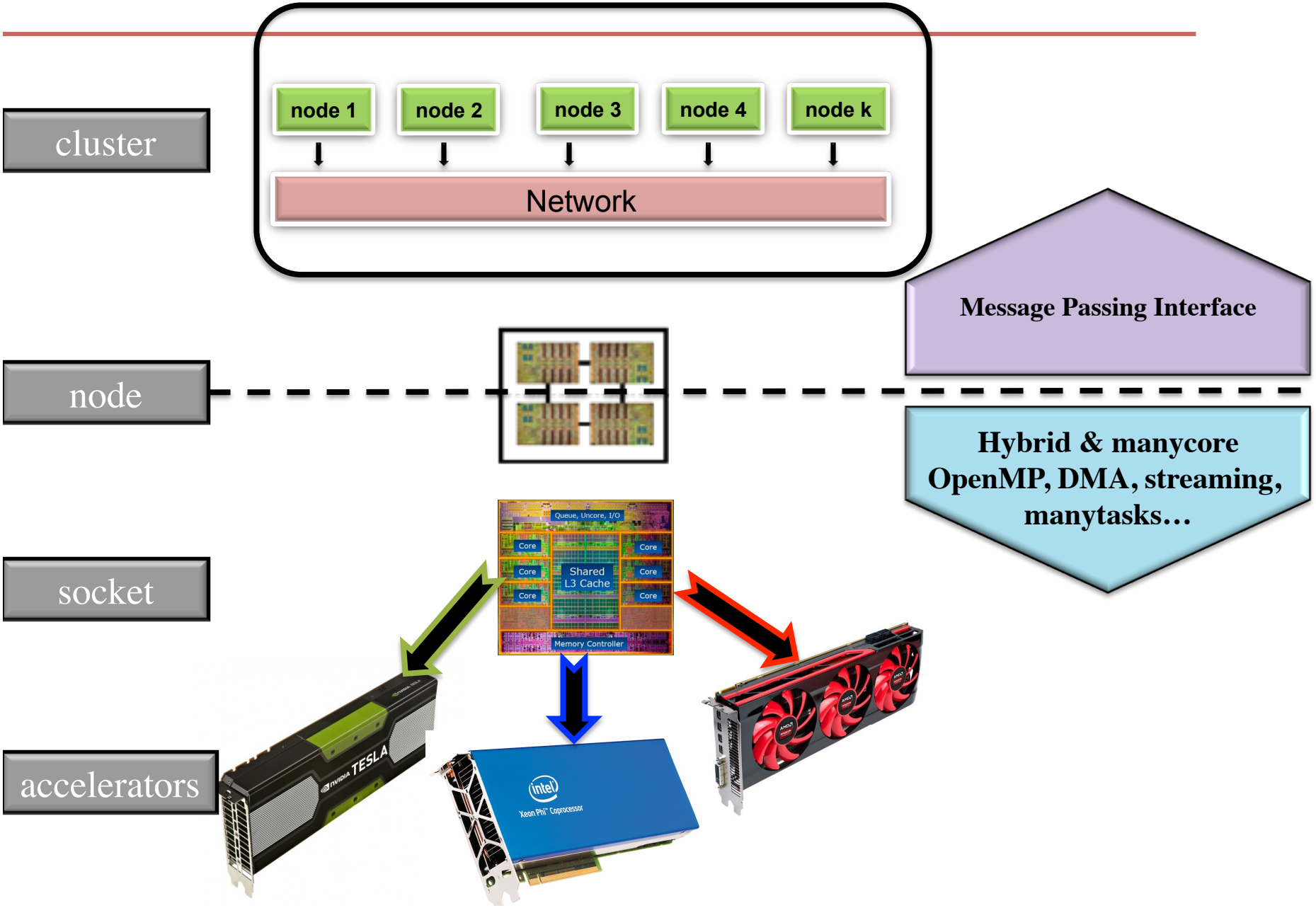
Performance Development of HPC over the Last 22 Years from the Top500



June 2014: 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 (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	Texas Advanced Computing Center	Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB	USA	204,900	5.17	61	4.5	1489
8	Forschungszentrum Juelich (FZJ)	JuQUEEN, BlueGene/Q, Power BQC 16C 1.6GHz+Custom	Germany	458,752	5.01	85	2.30	2178
9	DOE / NNSA L Livermore Nat Lab	Vulcan, BlueGene/Q, Power BQC 16C 1.6GHz+Custom	USA	393,216	4.29	85	1.97	2177
10	Government	Cray XC30, Xeon E5 12C 2.7GHz, Custom	USA	225,984	3.14	64		
500	Meteorological	Cray XC30	Germany	7280	.134	91		

Mixed Multi-GPU and Multi-Coprocessor Environments



Commodity plus Accelerator Today

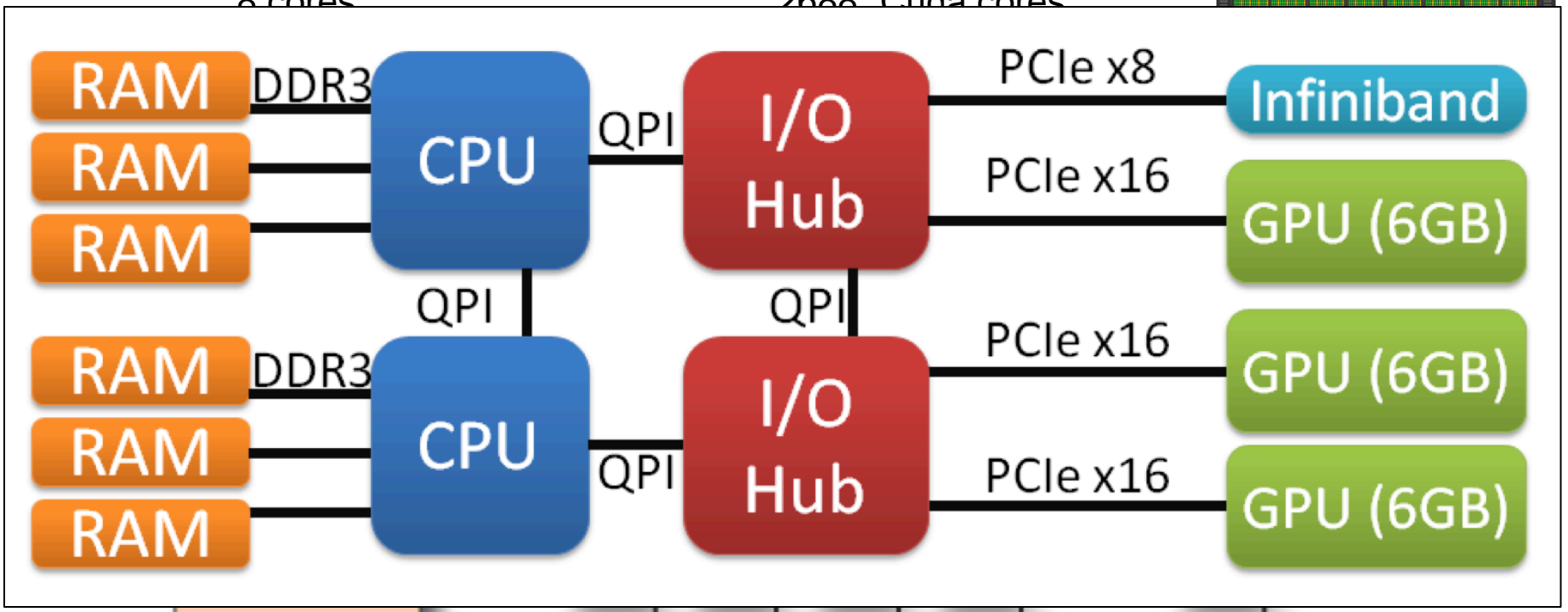
Commodity

Intel Xeon
8 cores

Accelerator (GPU)

Nvidia K20X “Kepler”
2688 “Cuda cores”

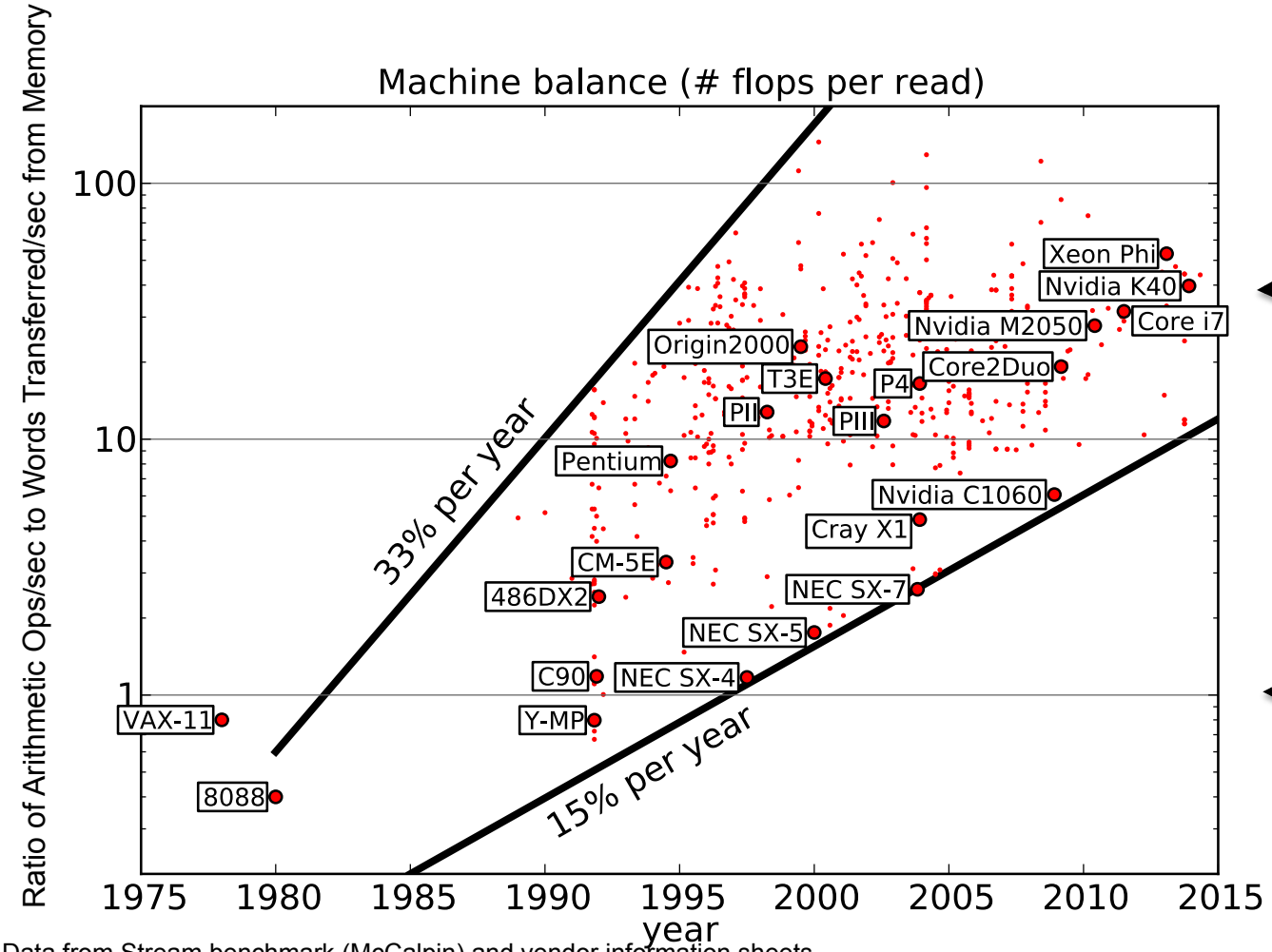
192 Cuda cores/SMX
2688 “Cuda cores”



Interconnect
PCI-e Gen2/3 16 lane
64 Gb/s (8 GB/s)
1 GW/s

Ratio of Floating Point Speed to Memory Bandwidth has Been Increasing 15-33% per Year

Computers are over previsioned for floating point
Would like Performance Ratio of flops to data movement of O(1)



- Flops "free," memory expensive
- Good for dense, BLAS-3 operations (matrix multiply)

- Flops & memory access balanced
- Good for sparse & vector operations

Data from Stream benchmark (McCalpin) and vendor information sheets

Peak Performance - Per Core

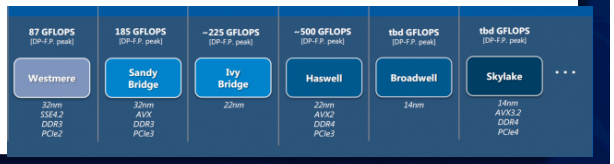
$$\text{FLOPS} = \text{cores} \times \text{clock} \times \frac{\text{FLOPs}}{\text{cycle}}$$

Floating point operations per cycle per core

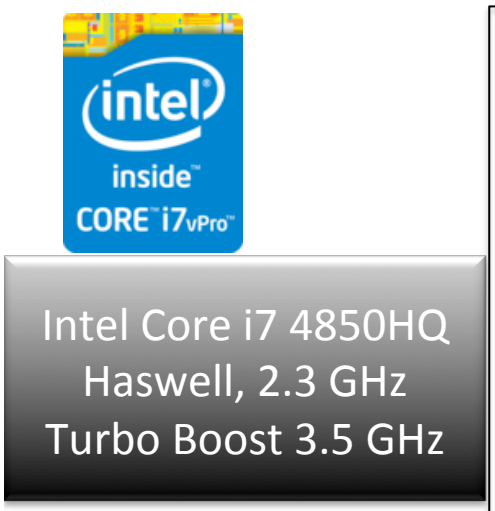
- Most of the recent computers have FMA (Fused multiple add): (i.e. x ←x + y*z in one cycle)
- Intel Xeon earlier models and AMD Opteron have SSE2
  - 2 flops/cycle DP & 4 flops/cycle SP
- Intel Xeon Nehalem ('09) & Westmere ('10) have AVX
  - 4 flops/cycle DP & 8 flops/cycle SP
- Intel Xeon Sandy Bridge('11) & Ivy Bridge ('12) have AVX
  - 8 flops/cycle DP & 16 flops/cycle SP

We are here

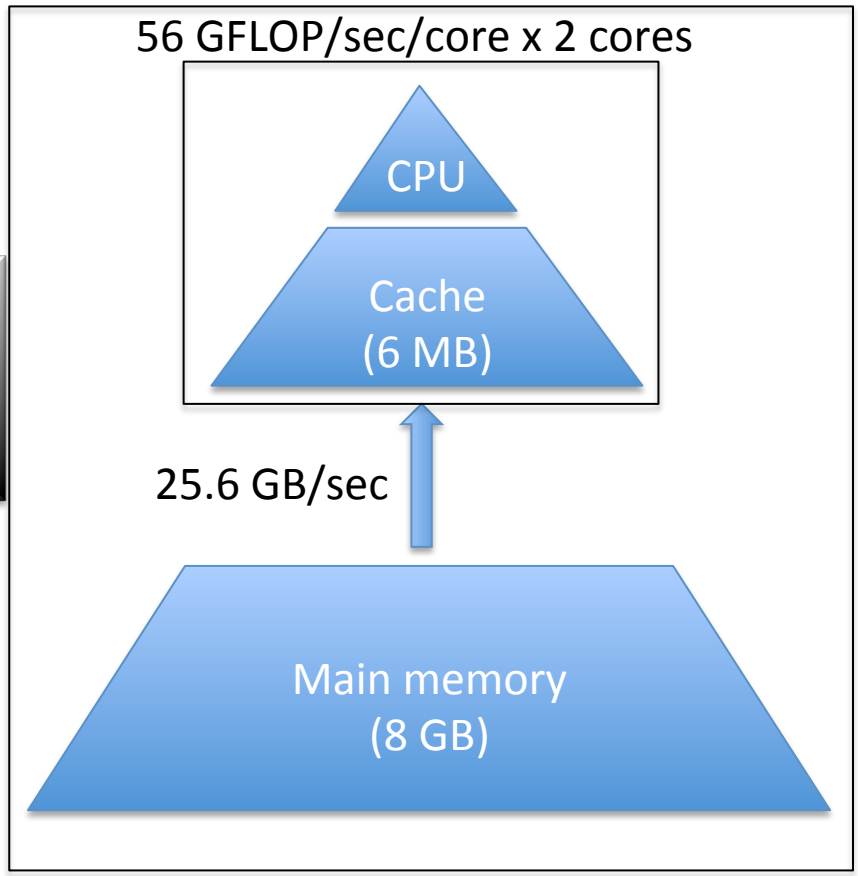
- Intel Xeon Haswell ('13) & (Broadwell ('14)) AVX2
  - 16 flops/cycle DP & 32 flops/cycle SP
  - Xeon Phi (per core) is at 16 flops/cycle DP & 32 flops/cycle SP
- Intel Xeon Skylake ('15)
  - 32 flops/cycle DL & 64 flops/cycle SP



Memory transfer
(Its All About Data Movement)
Example on my laptop: One level of memory



Intel Core i7 4850HQ
Haswell, 2.3 GHz
Turbo Boost 3.5 GHz

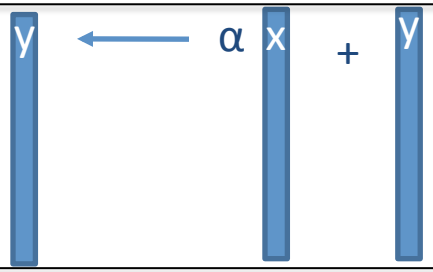


(Omitting latency here.)

The model IS simplified (see next slide) but it provides an upper bound on performance as well. I.e., we will never go faster than what the model predicts.
(And, of course, we can go slower …)

FMA: fused multiply-add

AXPY:

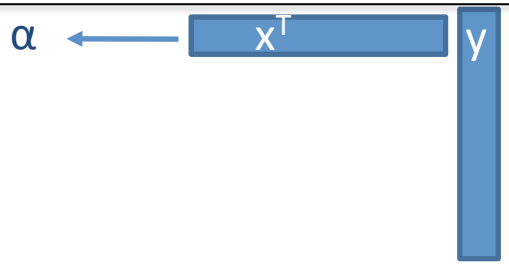



```
for ( j = 0; j < n; j++)
      y[i] += a * x[i];
```

(without increment)

n MUL
n ADD
2n FLOP
n FMA

DOT:




```
alpha = 0e+00;
for ( j = 0; j < n; j++)
      alpha += x[i] * y[i];
```

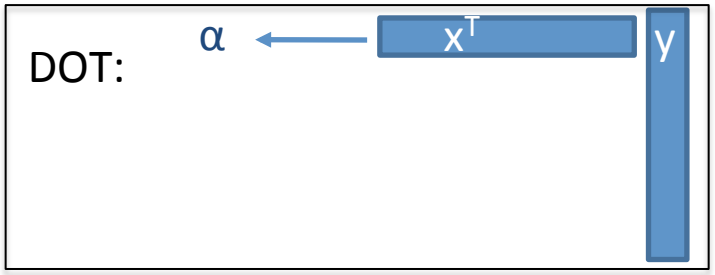
(without increment)

n MUL
n ADD
2n FLOP
n FMA

Note: It is reasonable to expect the one loop codes shown here to perform as well as their Level 1 BLAS counterpart (on multicore with an OpenMP pragma for example).

The true gain these days with using the BLAS is (1) Level 3 BLAS, and (2) portability.

- Take two double precision vectors x and y of size n=375,000.



- Data size:
  - (375,000 double) * (8 Bytes / double) = 3 MBytes per vector

  (Two vectors fit in cache (6 MBytes). OK.)

- Time to move the vectors from memory to cache:
  - (6 MBytes) / (25.6 GBytes/sec) = **0.23 ms**

- Time to perform computation of DOT:
  - (2n flop) / (56 Gflop/sec) = **0.01 ms**

Vector Operations

$$\text{total\_time} \geq \text{time\_comm} + \text{time\_comp}$$

$$= 0.23\text{ms} + 0.01\text{ms} = 0.24\text{ms}$$

1. The equation assumes that communication and computation do not overlap. This assumption is not really true. For a AXPY (without INCX), a nice streaming/pipeline occurs. Data locality is perfect.

2. We will assume overlapping communication and computation. We are looking for a strict lower bound, and there is no need to second guessing and make assumptions.

Vector Operations

total_time ≥ max (time_comm , time_comp)

= max (0.23ms , 0.01ms) = 0.23ms

Performance = (2 x 375,000 flops)/.23ms = 3.2 Gflop/s

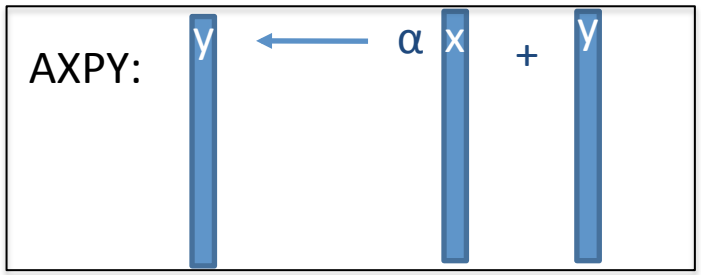
Performance for DOT ≤ 3.2 Gflop/s

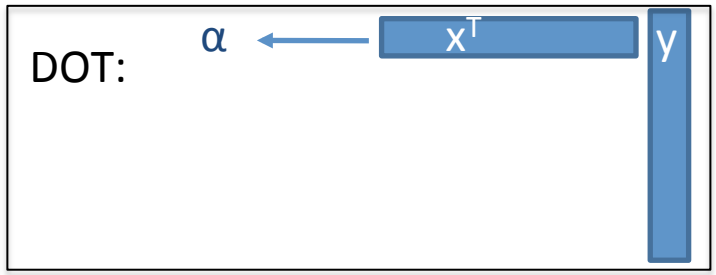
Peak is 56 Gflop/s

We say that the operation is communication bounded. No reuse of data.

Level 1, 2 and 3 BLAS

Level 1 BLAS Matrix-Vector operations

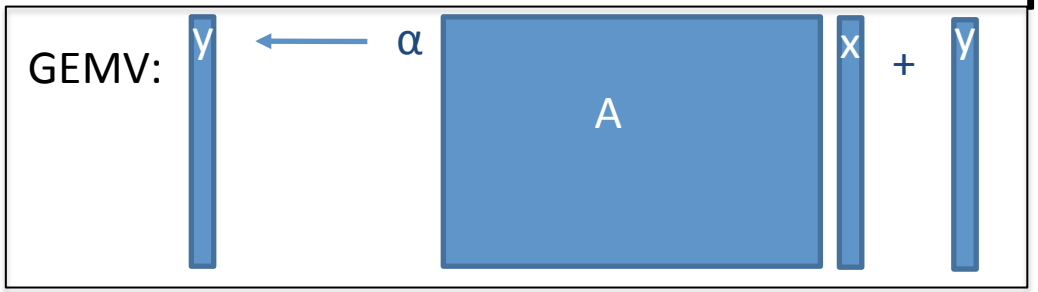




2n FLOP
2n memory reference
AXPY: 2n READ, n WRITE
DOT: 2n READ

RATIO: 1

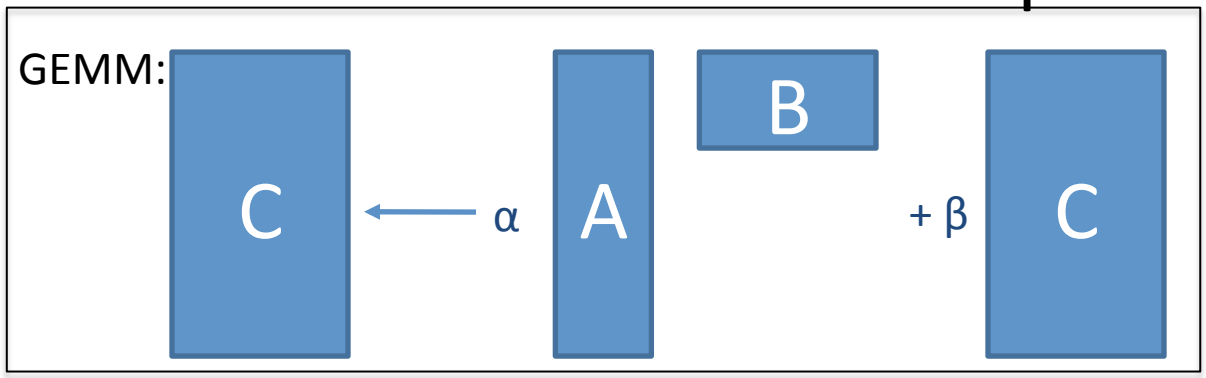
Level 2 BLAS Matrix-Vector operations



$2n^2$ FLOP
n^2 memory references

RATIO: 2

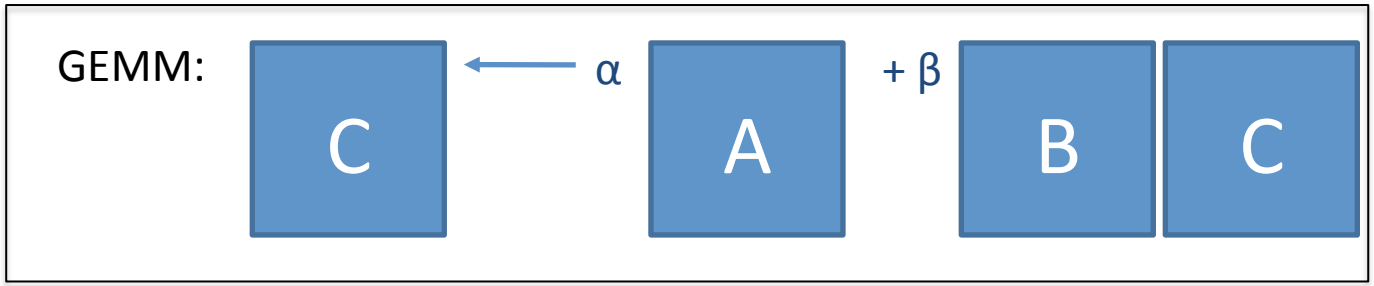
Level 3 BLAS Matrix-Matrix operations



$2n^3$ FLOP
$3n^2$ memory references
$3n^2$ READ, n^2 WRITE

RATIO: 2/3 n

- Take two double precision vectors x and y of size n=500.



- Data size:
  - (500^2 double) * (8 Bytes / double) = 2 MBytes per matrix

  (Three matrices fit in cache (6 MBytes). OK.)

- Time to move the matrices in cache:
  - (6 MBytes) / (25.6 GBytes/sec) = **0.23 ms**

- Time to perform computation in GEMM:
  - ($2n^3$ flop) / (4.4 Gflop/sec) = **4.46 ms**

Matrix Matrix Operations

total_time ≥ max (time_comm , time_comp)

= max(0.23ms , 4.46ms) = 4.46ms

For this example, communication time is less than 6% of the computation time.

Performance = (2 x 500^3 flops)/4.69ms = 53.3 Gflop/s

There is a lots of data reuse in a GEMM; 2/3n per data element. Has good temporal locality.

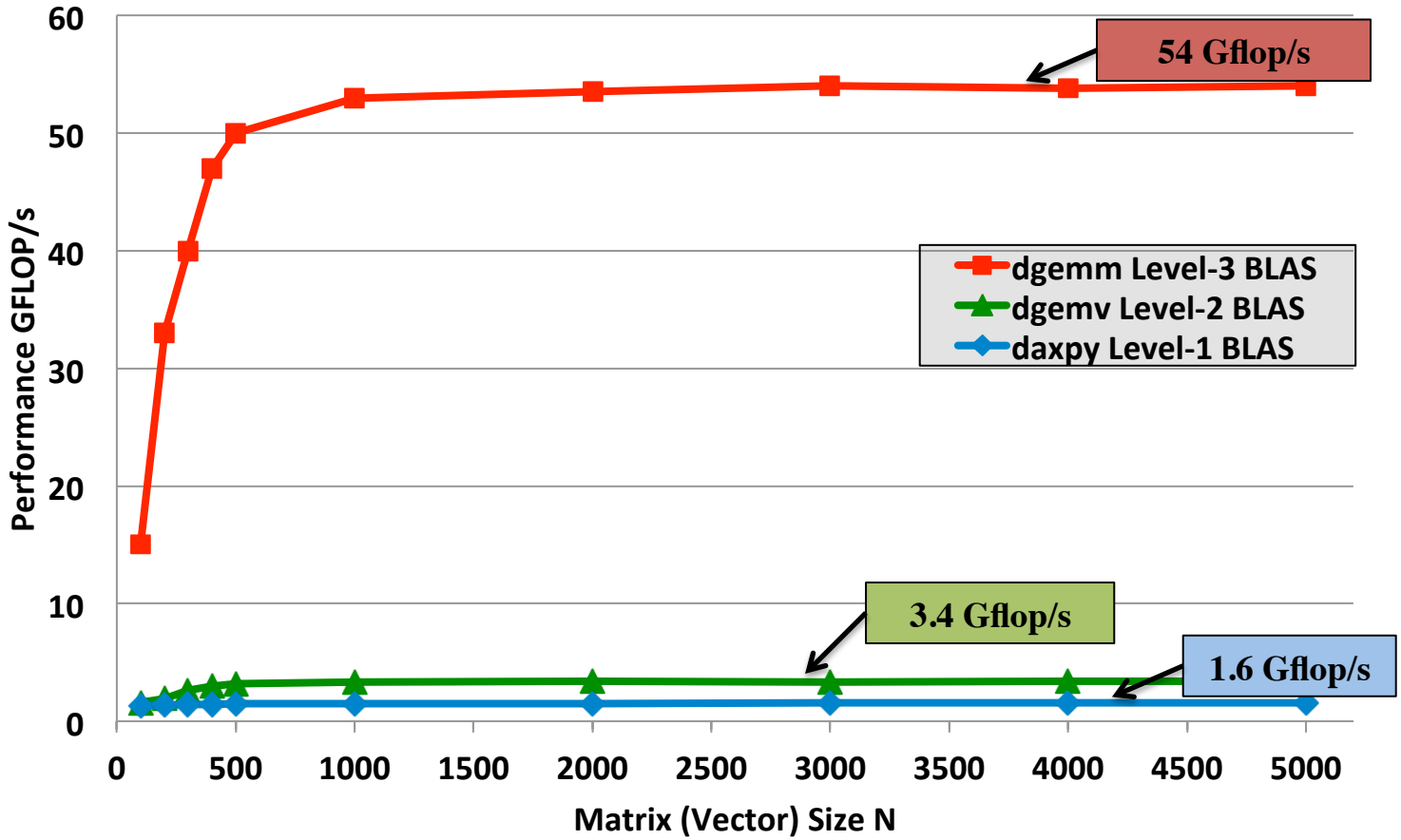
If we assume total_time ≈ time_comm +time_comp, we get

performance for GEMM ≈ 53.3 Gflop/sec

(Out of 56 Gflop/sec possible, so that would be 95% peak performance efficiency.)

Level 1, 2 and 3 BLAS

1 core Intel Haswell i7-4850HQ, 2.3 GHz (Turbo Boost at 3.5 GHz);
Peak = 56 Gflop/s

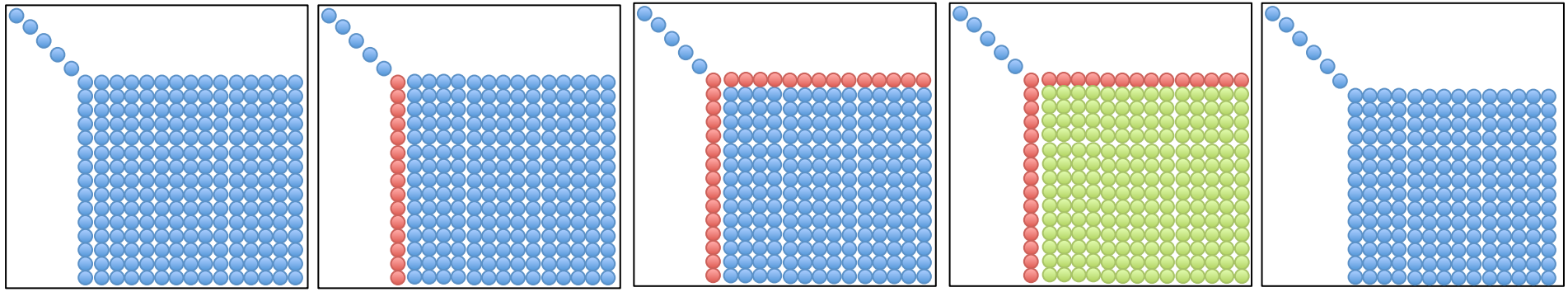


1 core Intel Haswell i7-4850HQ, 2.3 GHz, Memory: DDR3L-1600MHz
6 MB shared L3 cache, and each core has a private 256 KB L2 and 64 KB L1.
The theoretical peak per core double precision is 56 Gflop/s per core.
Compiled with gcc and using Veclib

Issues

- Reuse based on matrices that fit into cache.
- What if you have matrices bigger than cache?

The Standard LU Factorization LINPACK 1970's HPC of the Day: Vector Architecture



Factor column with Level 1 BLAS

Divide by Pivot row

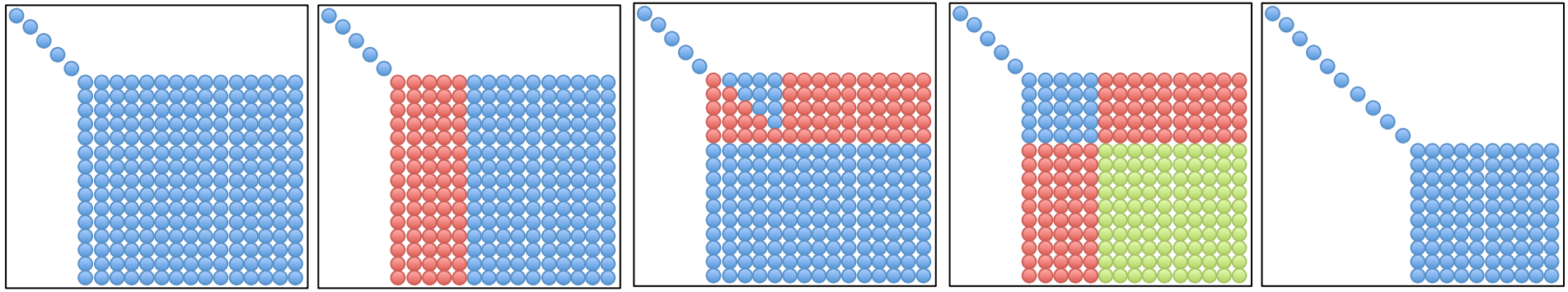
Schur complement update (Rank 1 update)

Next Step

Main points

- Factorization column (zero) mostly sequential due to memory bottleneck
- Level 1 BLAS
- Divide pivot row has little parallelism
- Rank -1 Schur complement update is the only easy parallelize task
- Partial pivoting complicates things even further
- Bulk synchronous parallelism (fork-join)
  - Load imbalance
  - Non-trivial Amdahl fraction in the panel
  - Potential workaround (look-ahead) has complicated implementation

The Standard LU Factorization LAPACK 1980's HPC of the Day: Cache Based SMP



Factor panel with Level 1,2 BLAS

Triangular update

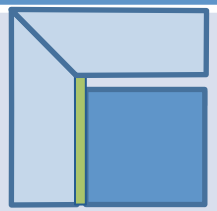
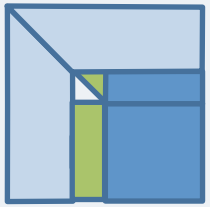
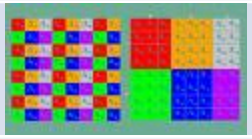
Schur complement update

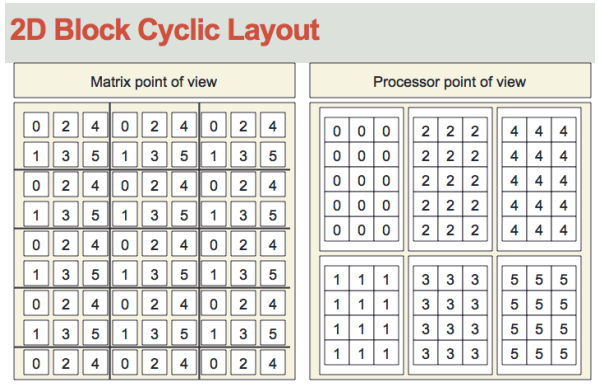
Next Step

Main points

- Panel factorization mostly sequential due to memory bottleneck
- Triangular solve has little parallelism
- Schur complement update is the only easy parallelize task
- Partial pivoting complicates things even further
- Bulk synchronous parallelism (fork-join)
  - Load imbalance
  - Non-trivial Amdahl fraction in the panel
  - Potential workaround (look-ahead) has complicated implementation

Last Generations of DLA Software

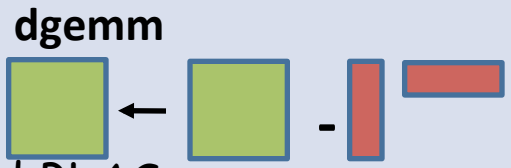
Software/Algorithms follow hardware evolution in time		
LINPACK (70's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (80's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
ScaLAPACK (90's) (Distributed Memory)		Rely on - PBLAS Mess Passing

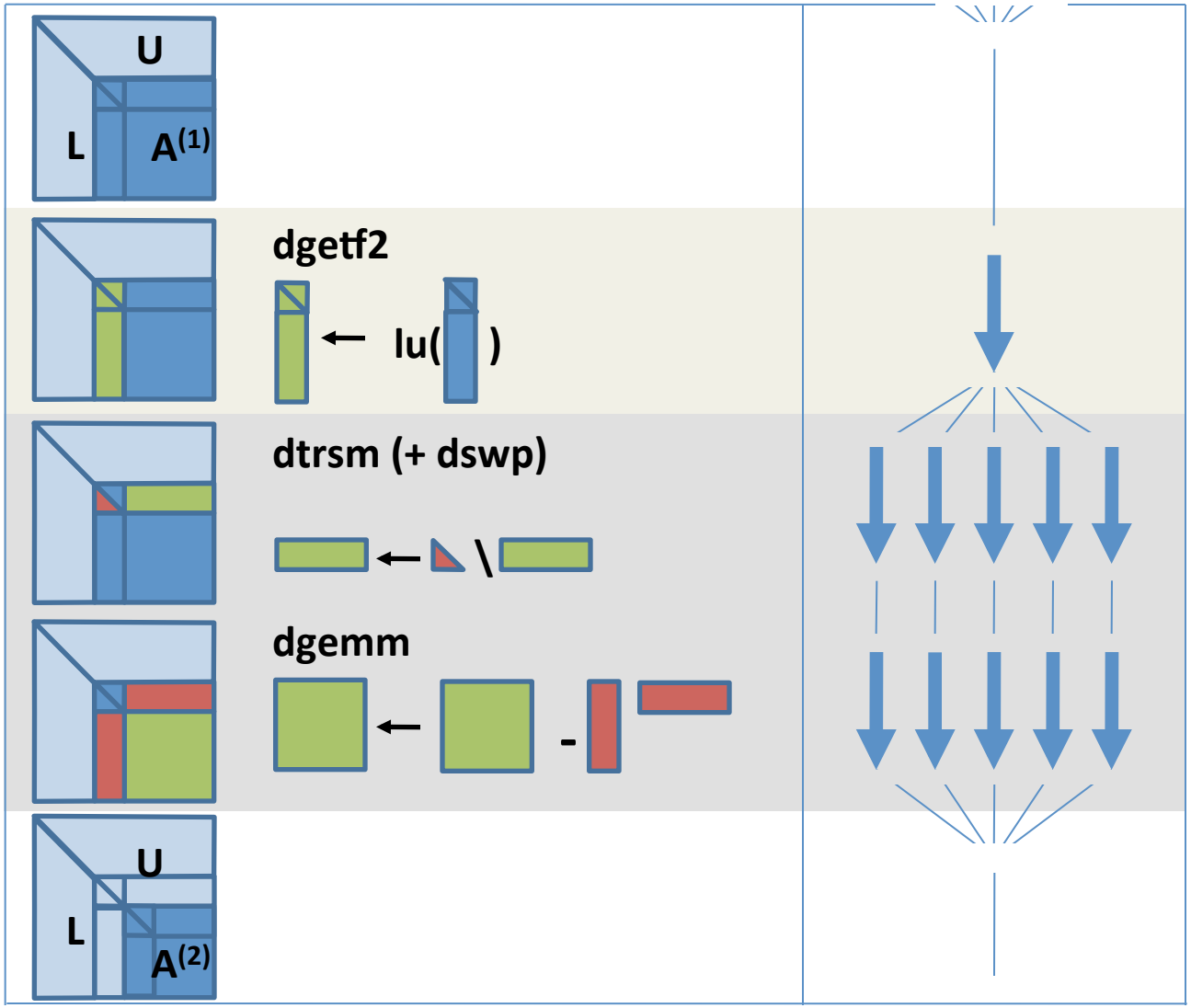


Parallelization of LU and QR.

Parallelize the update:

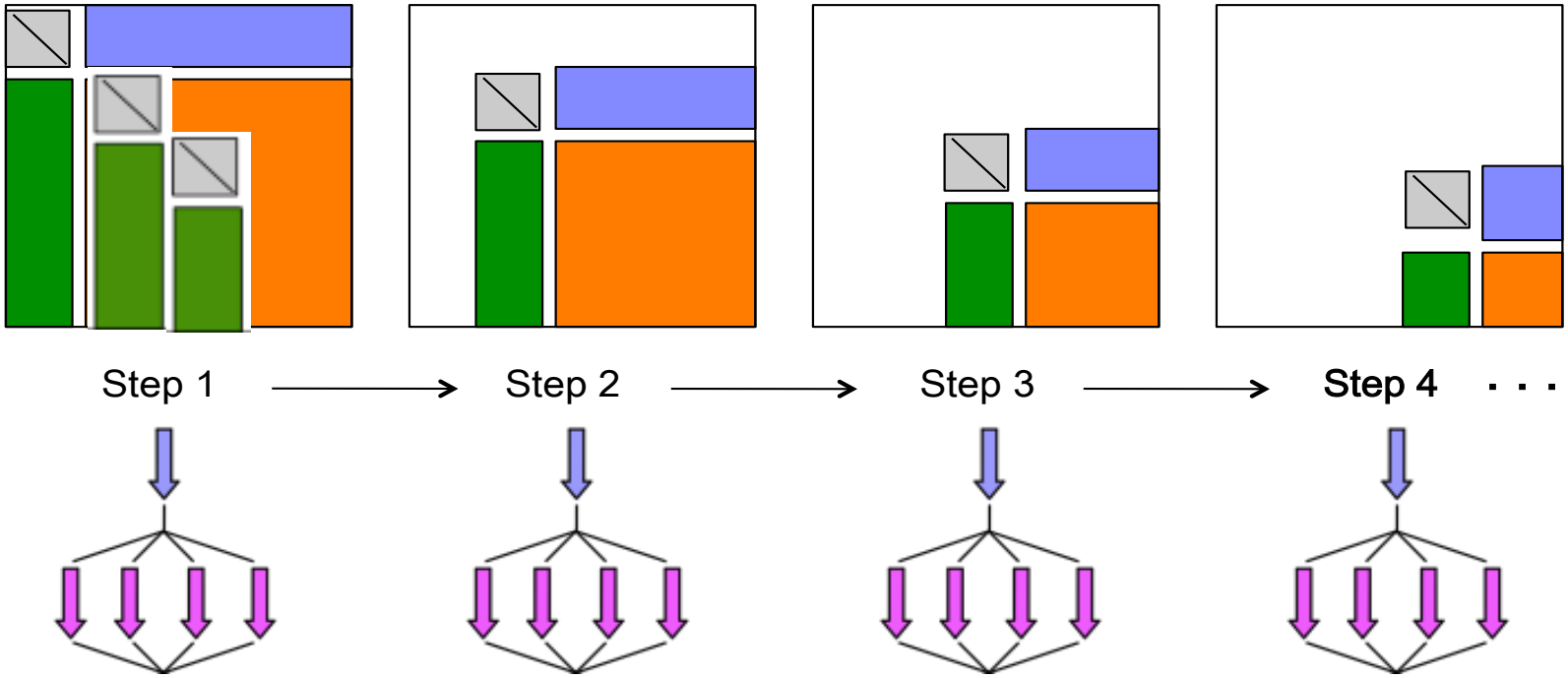
- Easy and done in any reasonable software.
- This is the $2/3n^3$ term in the FLOPs count.
- Can be done efficiently with LAPACK+multithreaded BLAS

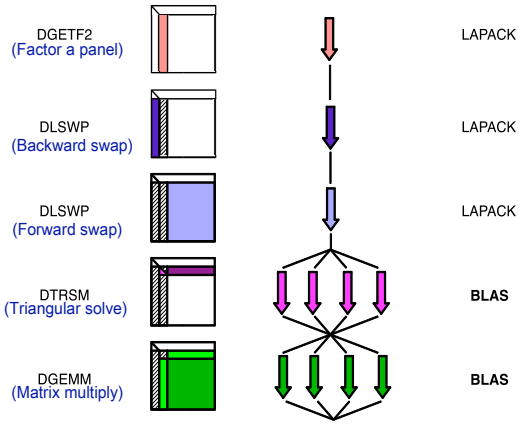




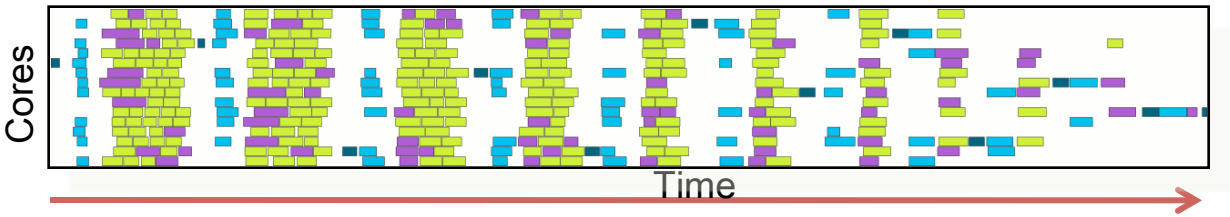
Fork - Join parallelism
Bulk Sync Processing

Synchronization (in LAPACK LU)



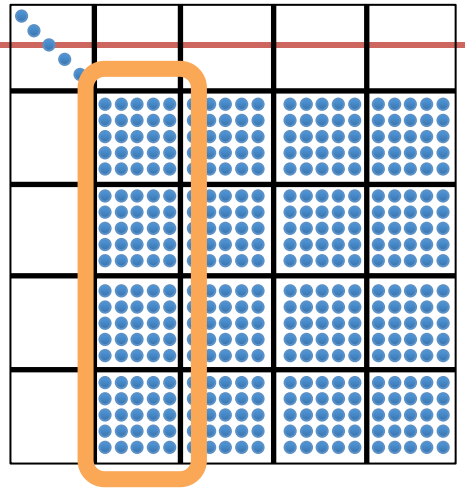


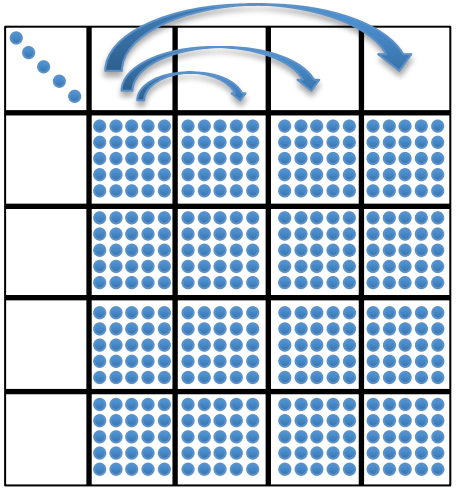
- fork join
- bulk synchronous processing

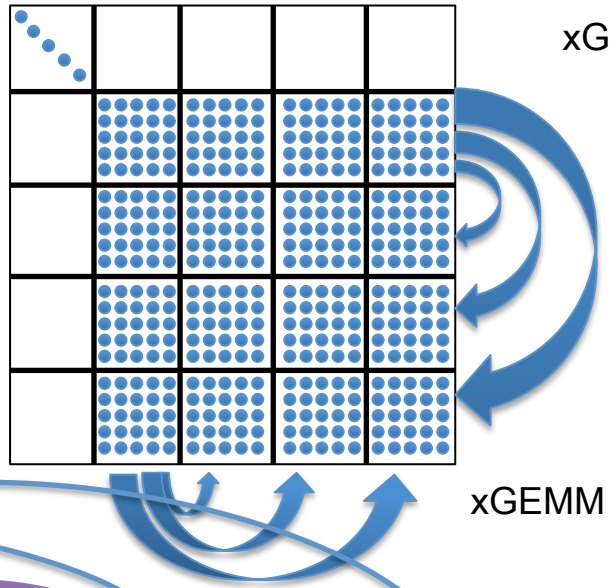


PLASMA LU Factorization

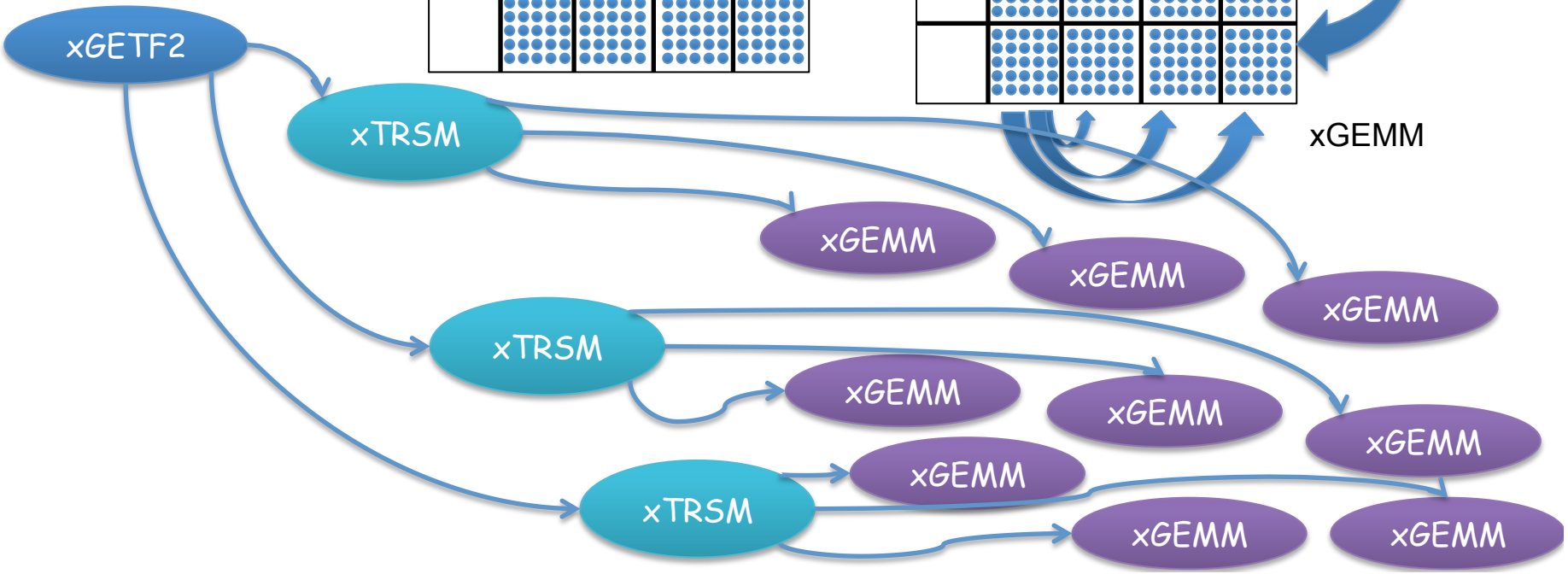
Dataflow Driven



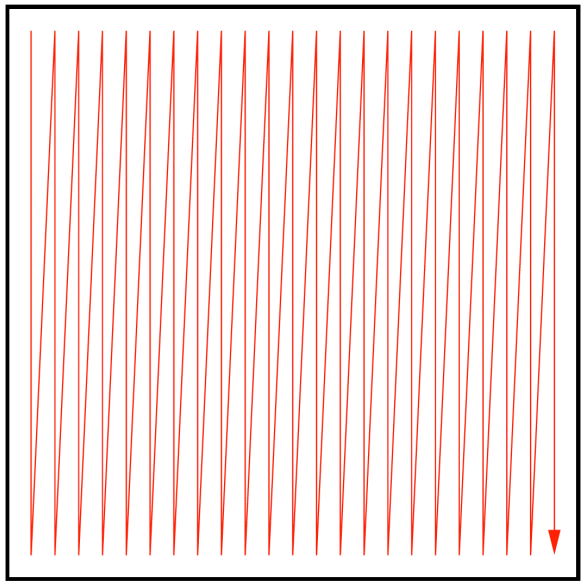


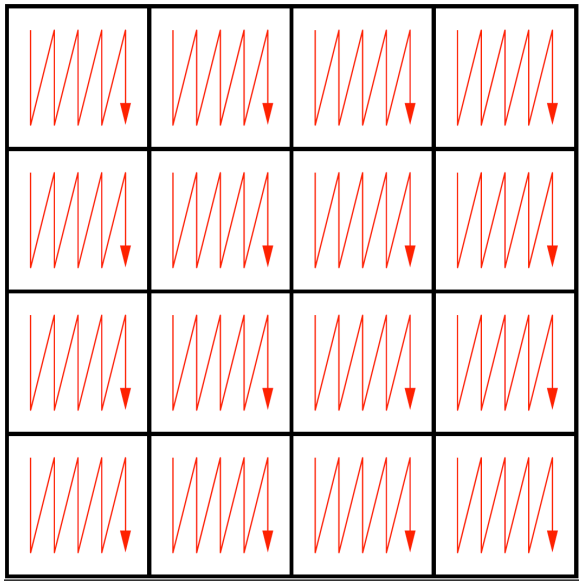


Numerical program generates tasks and run time system executes tasks respecting data dependences.



Data Layout is Critical



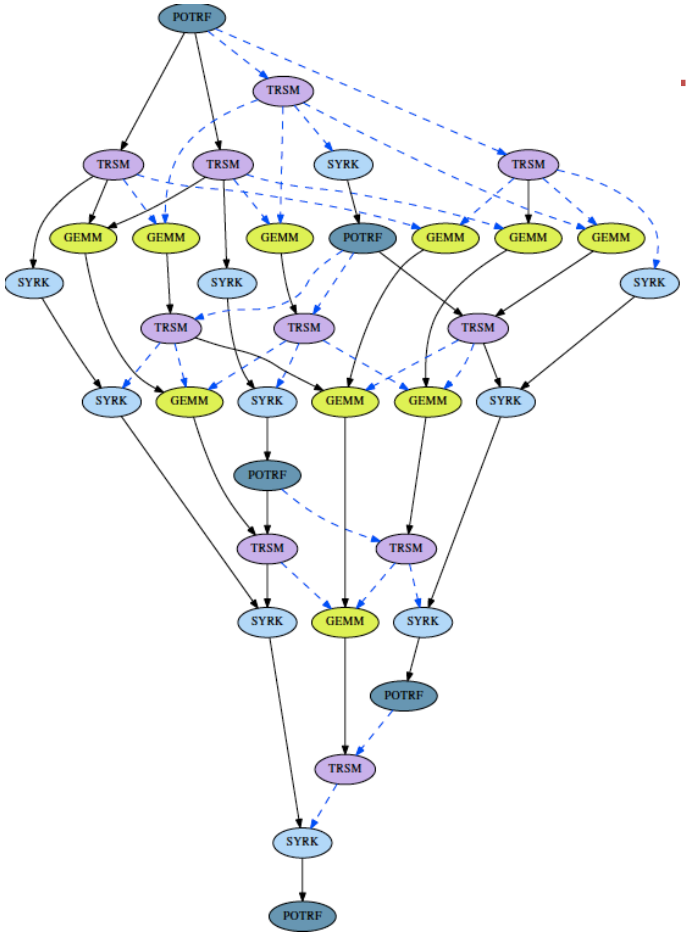


- Tile data layout where each data tile is contiguous in memory
- Decomposed into several fine-grained tasks, which better fit the memory of the small core caches

PLASMA: Tile Algorithms and Nested Parallelism

- Operates on one, two, or three matrix tiles at a time using a single core
  - This is called a kernel; executed independently of other kernels
  - Mostly Level 3 BLAS are used
- Data flows between kernels as prescribed by the programmer
- Coordination is done transparently via runtime scheduler (QUARK)
  - Parallelism level adjusted at runtime
  - Look-ahead adjusted at runtime
- Uses single-threaded BLAS with all the optimization benefits
- Panel is done on multiple cores

Run Time System - QUARK



A runtime environment for the dynamic execution of precedence-constraint tasks (DAGs) in a multicore machine

- Translation
- If you have a serial program that consists of computational kernels (tasks) that are related by data dependencies, QUARK can help you execute that program (relatively efficiently and easily) in parallel on a multicore machine

QUARK

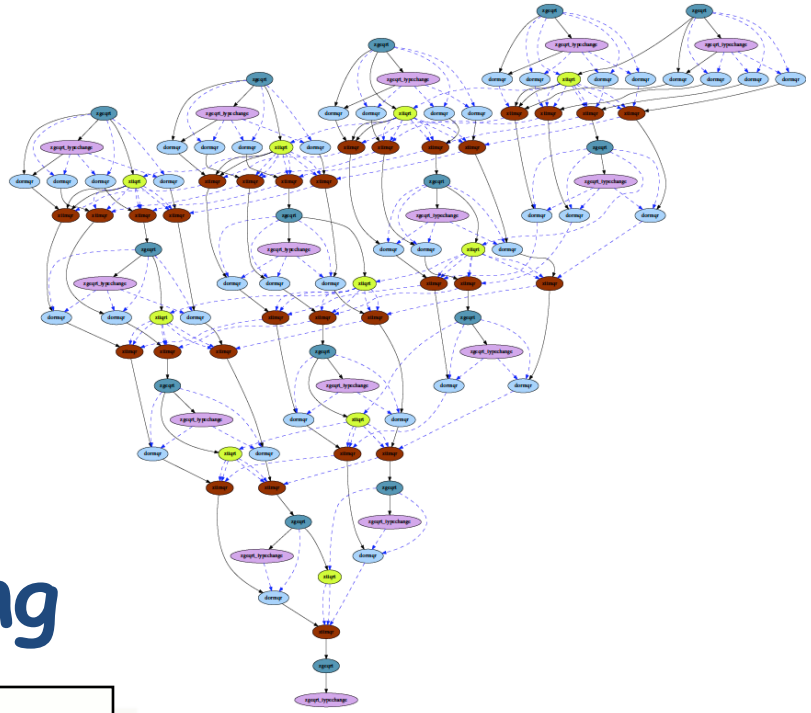
Shared Memory Superscalar Scheduling

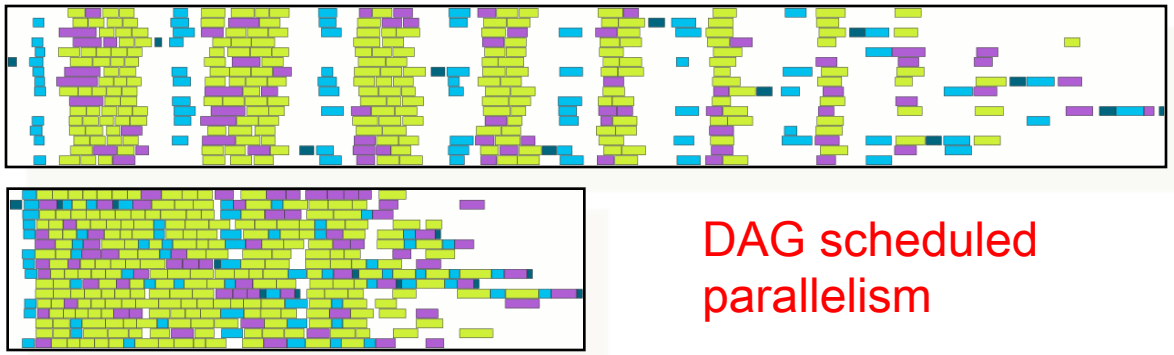
```
FOR k = 0..TILES-1
   A[k][k] ← DPOTRF(A[k][k])
   FOR m = k+1..TILES-1
      A[m][k] ← DTRSM(A[k][k], A[m][k])
   FOR m = k+1..TILES-1
      A[m][m] ← DSYRK(A[m][k], A[m][m])
      FOR n = k+1..m-1
         A[m][n] ← DGEMM(A[m][k], A[n][k], A[m][n])
```

definition – pseudocode

The Purpose of QUARK's Runtime

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

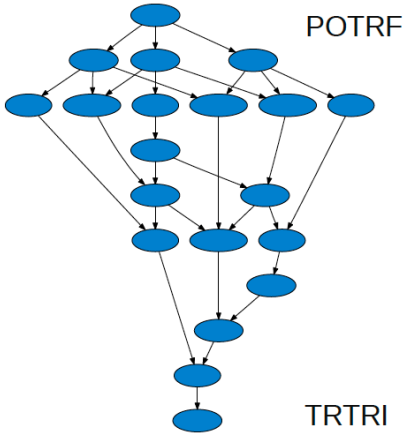


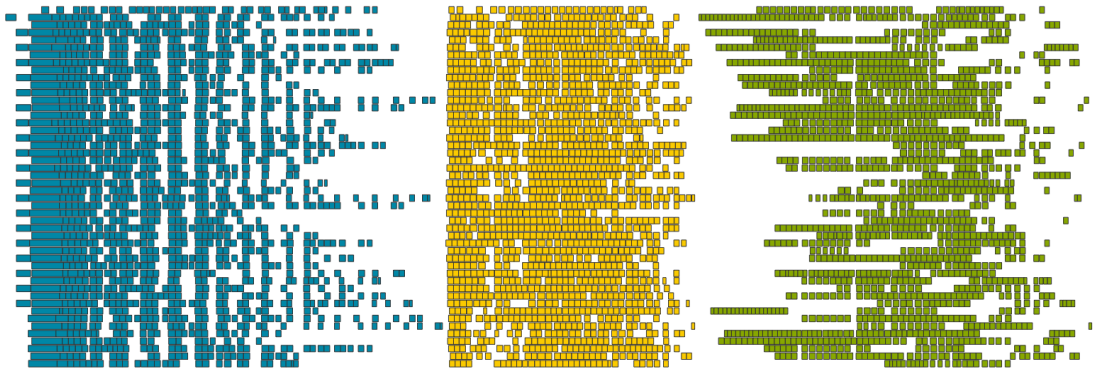


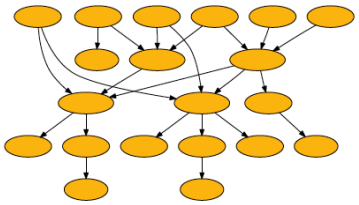
DAG scheduled parallelism

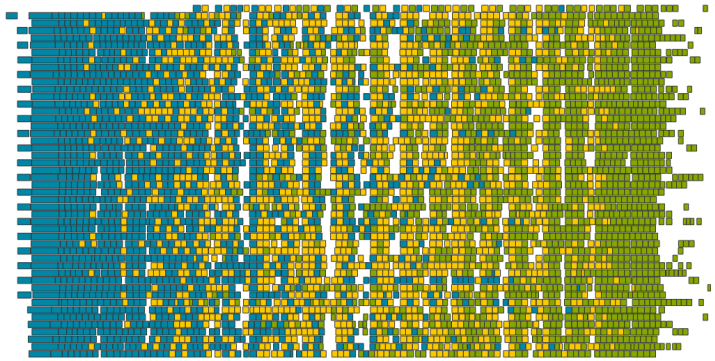
Fork-join parallelism
Notice the synchronization penalty in the presence of heterogeneity.

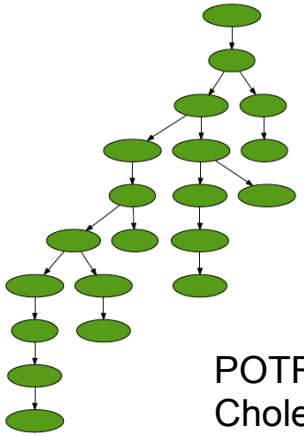
Pipelining: Cholesky Inversion
3 Steps: Factor, Invert L, Multiply L's





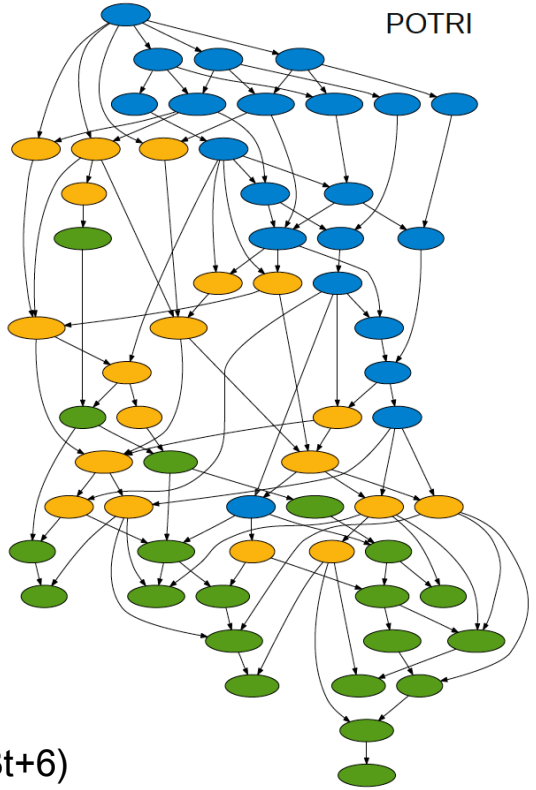






48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200,

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



Pipelined: 18 (3t+6)

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.

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

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

30

Ideas Have Been Used

- Systems of Equations
- Symmetric Eigenvalue Problem
- General Eigenvalue Problem
  - Numerical issues with non-linear elementary divisors
- Singular Value Decomposition
- Large Sparse Problems
  - Iterative methods
  - Communication Avoiding GMRES

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.

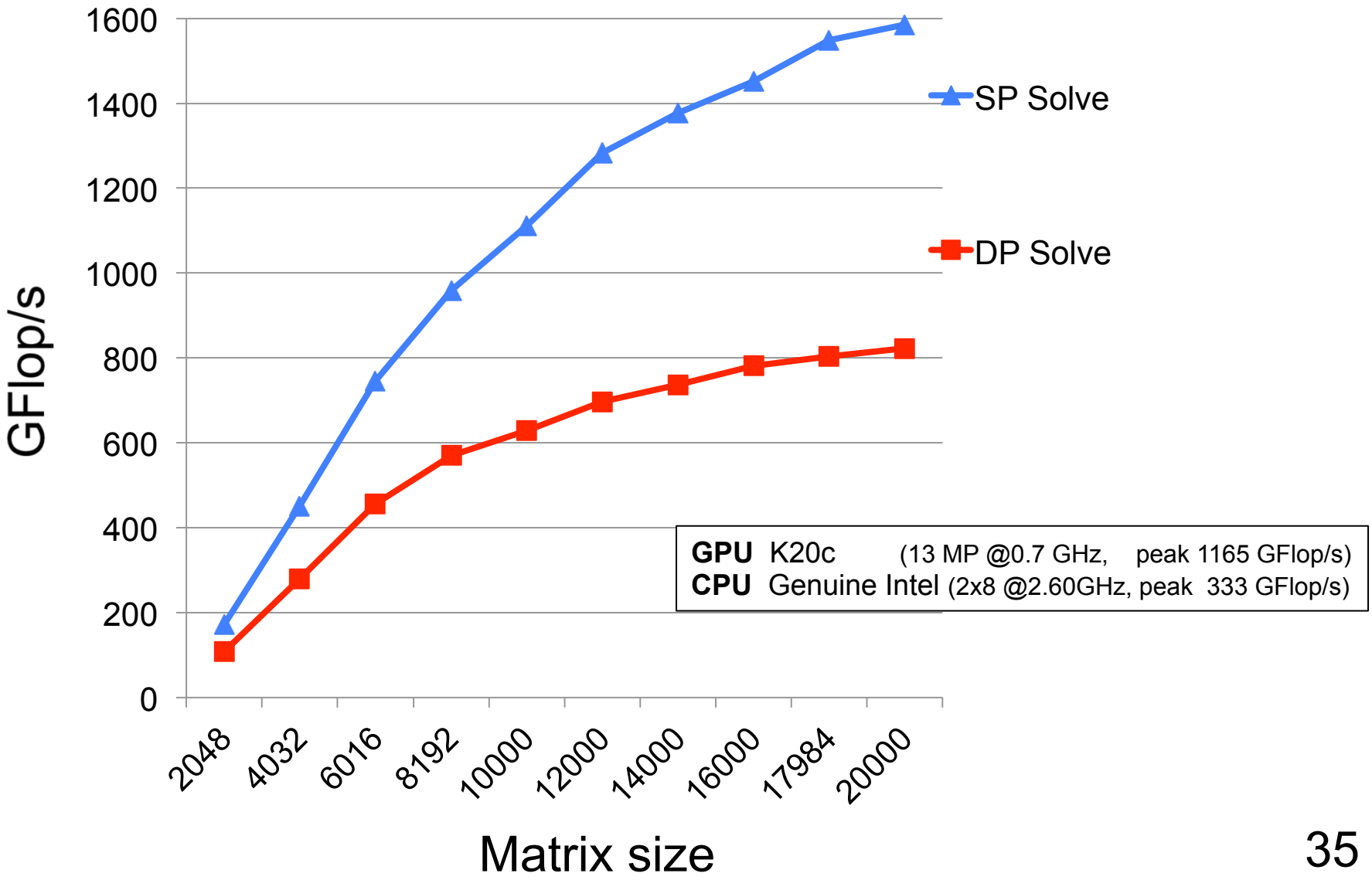
L U = lu(A)	SINGLE	$O(n^3)$
x = L\(U\b)	SINGLE	$O(n^2)$
r = b – Ax	DOUBLE	$O(n^2)$
WHILE || r || not small enough		
z = L\(U\r)	SINGLE	$O(n^2)$
x = x + z	DOUBLE	$O(n^1)$
r = b – Ax	DOUBLE	$O(n^2)$
END		

- 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; $O(10^8)$

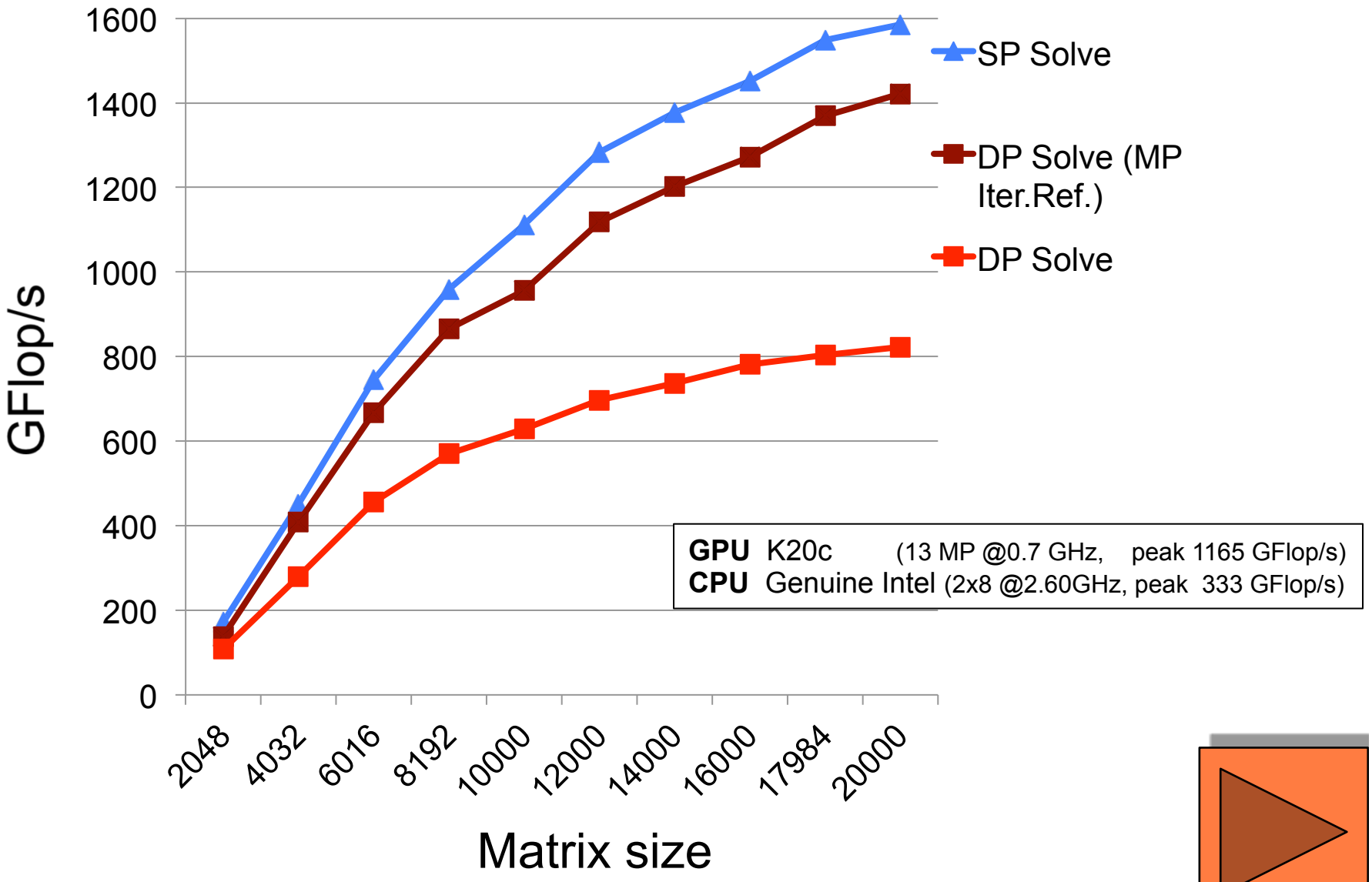
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



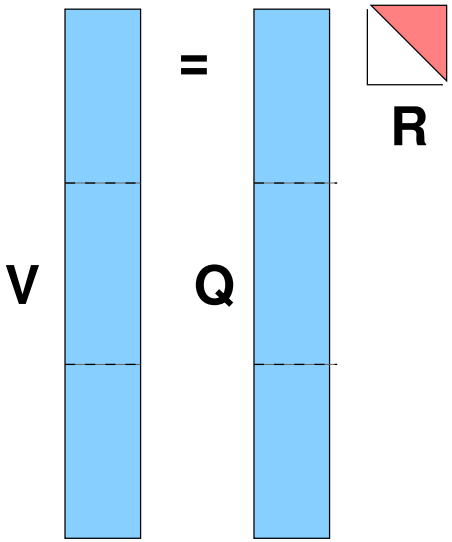
ICL UT

Mixed-precision QR and its application to CA-GMRES(s)

Sparse Iterative Method

Mixed-precision QR
and its application to CA-GMRES: TSQR: Tall-Skinny QR

- **Orthogonalizes a set of dense columns vectors V (m-by-s, m ≫ s),**



where Q is a set of orthogonal vectors, and R is upper triangular

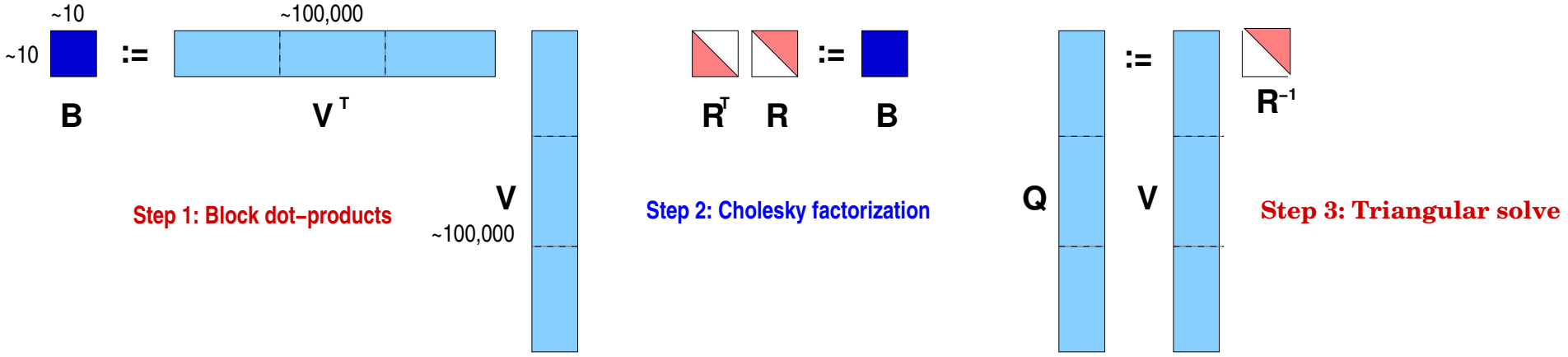
- **important computational kernels:**
  - **iterative subspace projection methods for solving large linear systems or eigenvalue problems**
  - **solution of over-determined least square problem,**
    x := argmin || A y = b || with m >> n.
    $x \in R_2$
  - **projection methods for low-rank approximation, eigen/singular problems**

TSQR Algorithms

- **Many ways to compute TSQR:**
  - **Householder QR** (with O(s) reductions)
    - Householder transform each column based on BLAS-1,2 xGEQR2
  - **Modified Gram-Schmidt** (with $O(s^2)$ reductions)
    - ortho each column against each column based on BLAS-1 xDOT and xAXPY
  - **Classical Gram-Schmidt** (with O(s) reductions)
    - ortho each column against prev columns based on BLAS-2 xGEMV
  - **Cholesky QR (or SVQR)** (with O(1) reductions)
    - ortho all columns against prev columns based on BLAS-3 xGEMM, xTRSM
  - **CAQR** (with O(1) reductions)
    - ortho all columns against prev columns based on tree-reduction BLAS-1,2 xGEQR2

CholQR factorization for TSQR

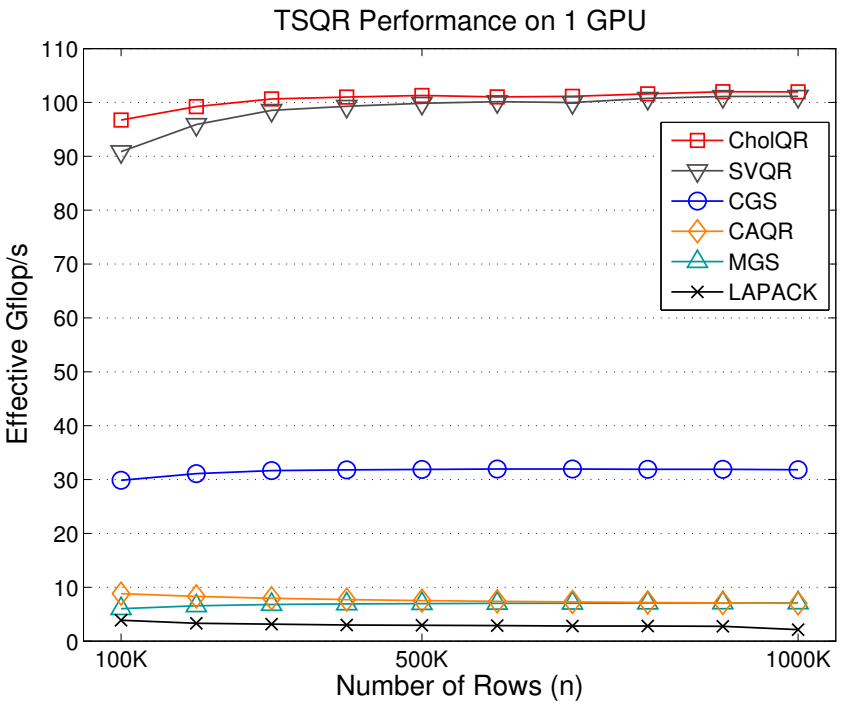
- Step 1 Gram-matrix formation $B := V^T V$ ($ns^2/2$ ops on GPUs)
- Step 2 Cholesky factorization $R^T R := B$ ($s^3/6$ ops on CPUs)
- Step 3 Backward-substitution $Q := VR^{-1}$ ($ns^2/2$ ops on GPUs)

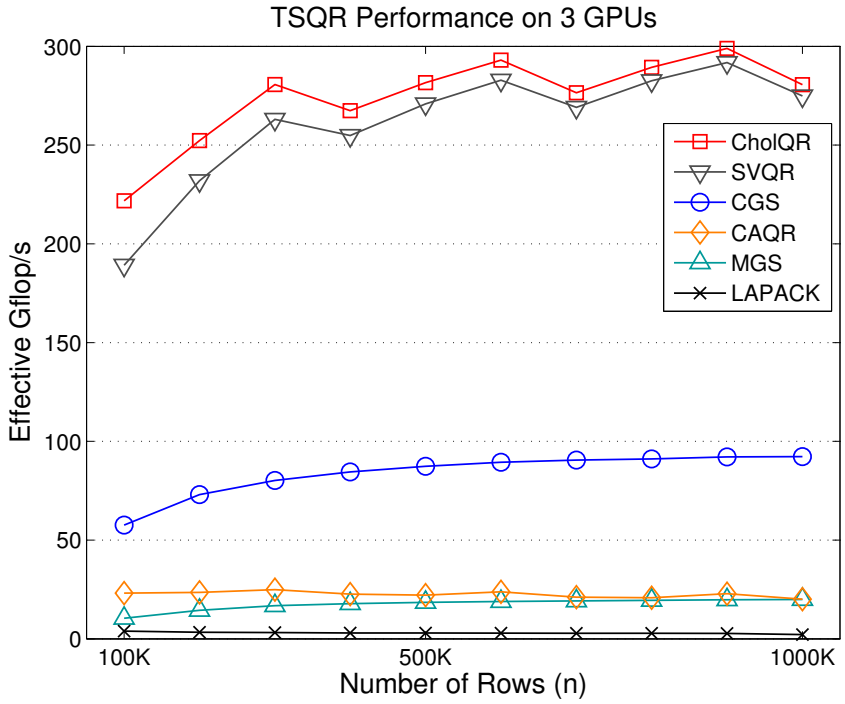


- Most of flops using BLAS-3
- Only one global reduction

TSQR Performance

(16-core Sandy Bridge with three M2090 Fermi, s = 30)





- **CholQR shows superior performance based on BLAS-3**
- **performance depends more on intra-comm (BLAS performance) than on inter-comm.**
- **it scales well over 3 GPUs.**

TSQR Stability

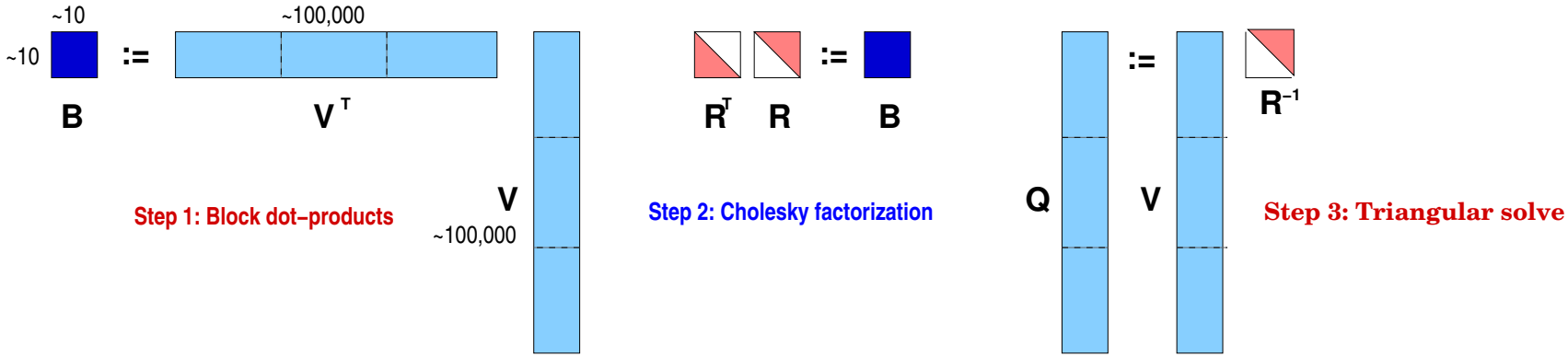
- **Trade-off between performance and stability**
  - **CholQR performs most of computation using BLAS-3**
  - **Condition number of Gram matrix B is square of A**

	$\|I - Q^T Q\|$	# flops, GPU kernel	# GPU-CPU comm.
MGS	$O(\epsilon\kappa(V))$	$2ns^2$, BLAS-1 xDOT	$O(s^2)$
CGS	$O(\epsilon\kappa(V)^2)$	$2ns^2$, BLAS-2 xGEMV	$O(s)$
CholQR	$O(\epsilon\kappa(V)^2)$	$2ns^2$, BLAS-3 xGEMM	$O(1)$
SVQR	$O(\epsilon\kappa(V)^2)$	$2ns^2$, BLAS-3 xGEMM	$O(1)$
CAQR	$O(\epsilon)$	$4ns^2$, BLAS-1,2 xGEQR2	$O(1)$

- **It pften requires reorthogonalization**
- **CA-GMRES may not converge**

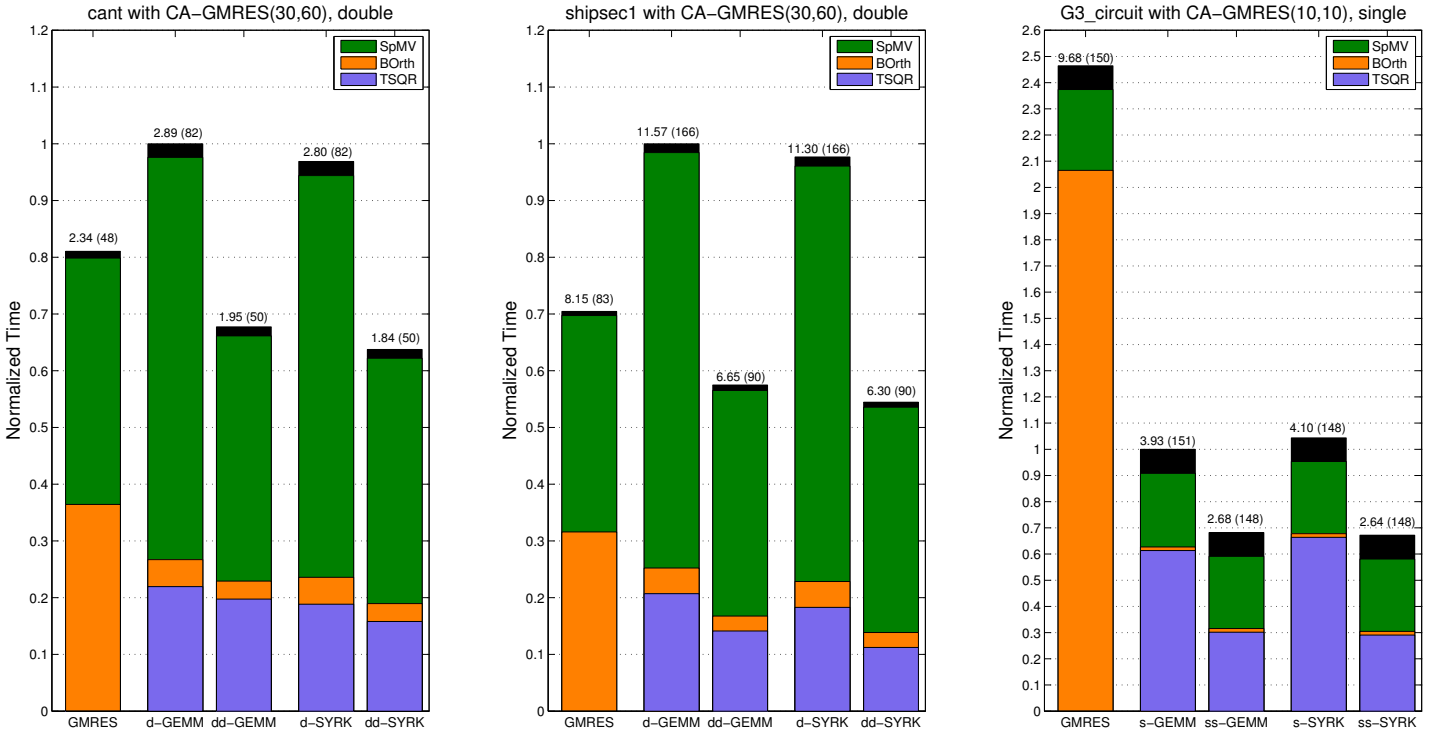
Mixed Precision CholQR

- **Remove "square" in error bound by selectively using "extended" precision:**
  - Step 1 Gram-matrix formation $B := V^T V$ (V in double)16× ops on GPUs in extended-precision
  - Step 2 Cholesky factorization $R^T R := B$ 16× ops on CPUs in extended-precision
  - Step 3 Backward-substitution $Q := VR^{-1}$ on 1× ops on GPUs in working-precision



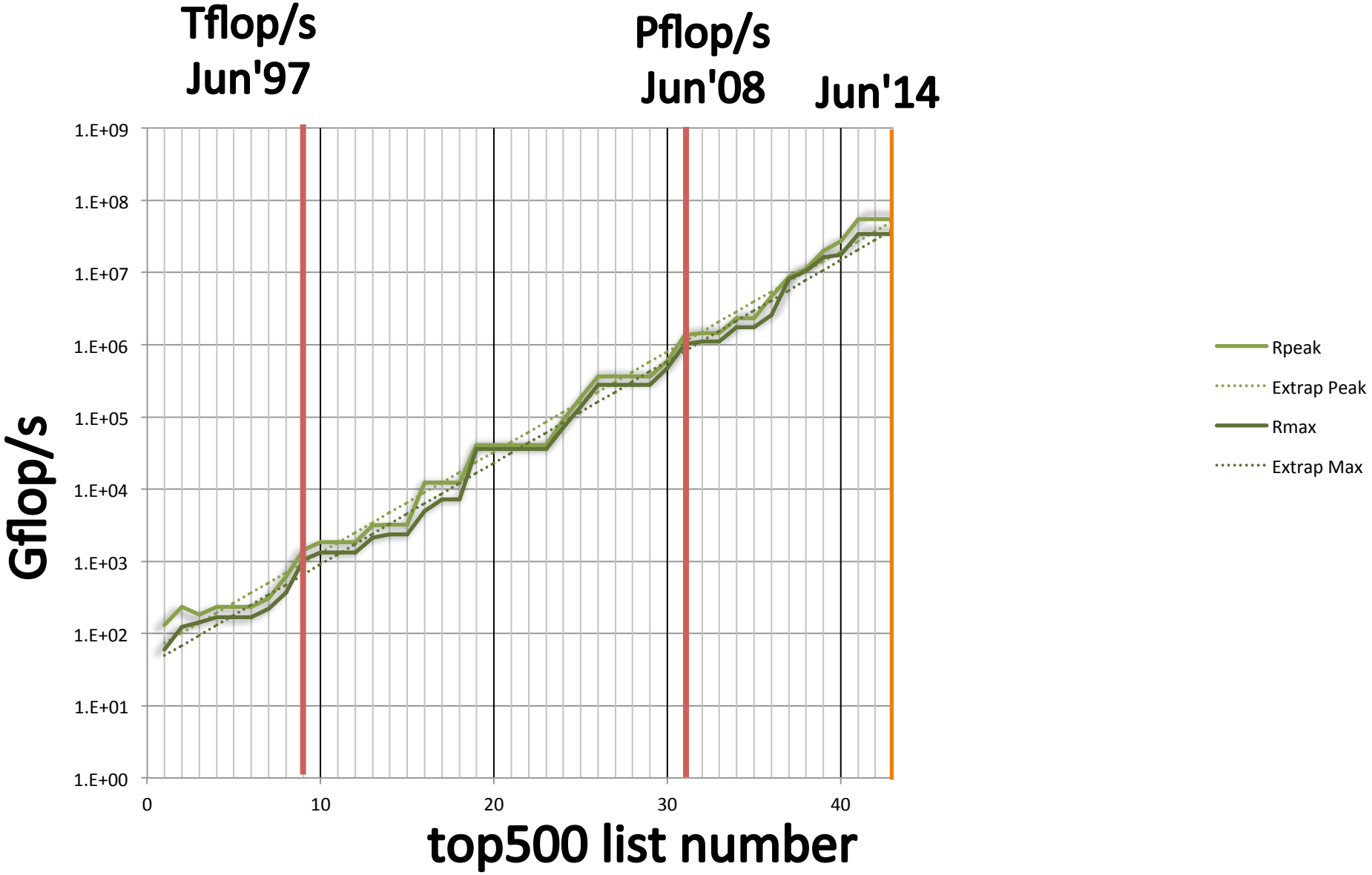
  - orthogonality error depends linearly on κ(V) (more details in SISC paper, submitted, underreview)
    $\| I - Q^T Q \| \leq O(\varepsilon\kappa(V) + (\varepsilon\kappa(V))^2)$ and $\| Q \| \leq 1 + O(\varepsilon\kappa(V))$
  - arithmetic instruction count increases by 8.5×

CA-GMRES with dd-CholQR

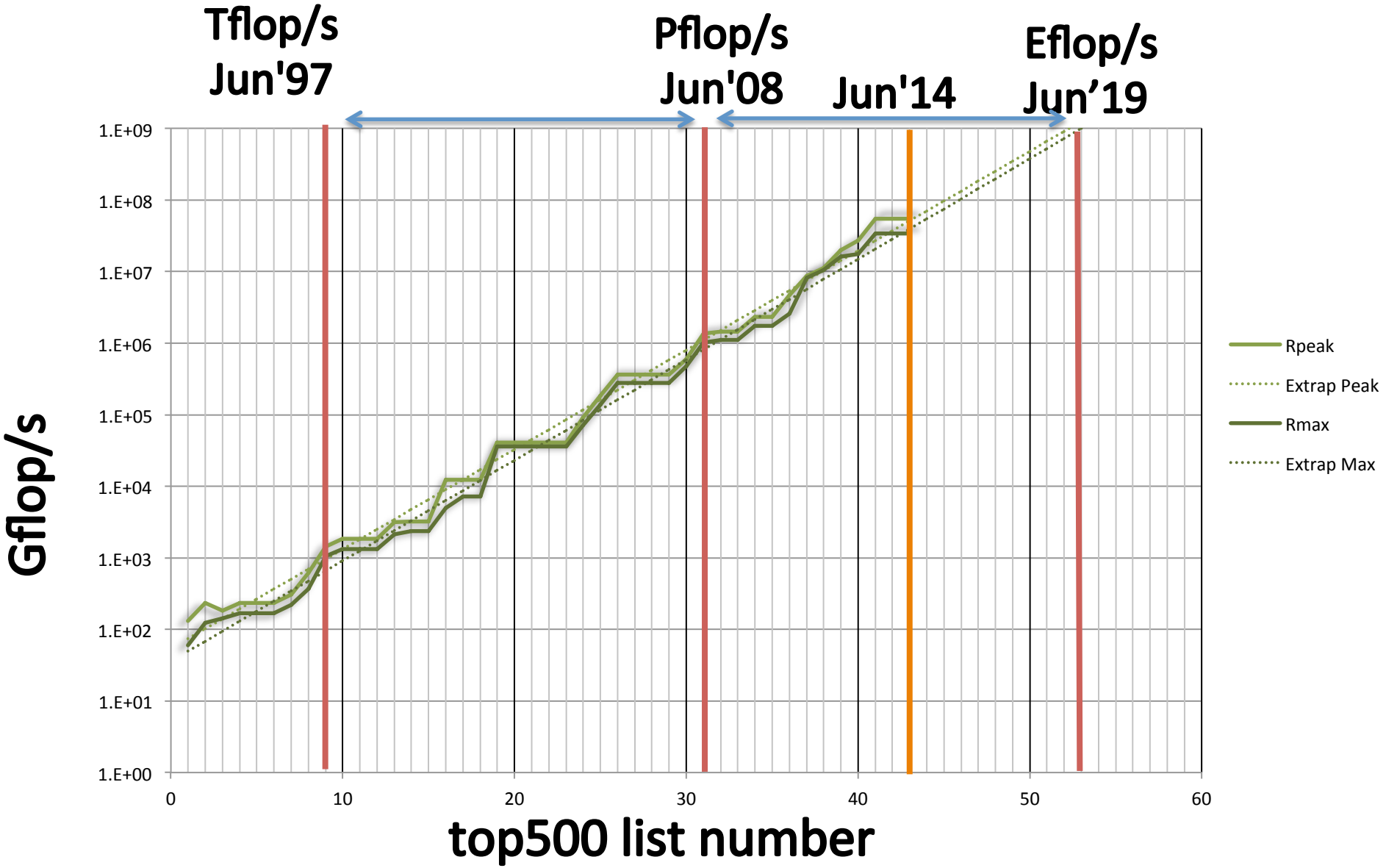


- **Total algorithm time to solution is reduced by > 40%**
- **Reorthogonalization is not needed (error depends linearly on κ(V)).**
  - **latency is reduced in half**
  - **total TSQR time may not increase, or may be reduced (dd 1.7× slower).**
- **CA-GMERS may converge quicker**
  - **solution time may be reduced.**

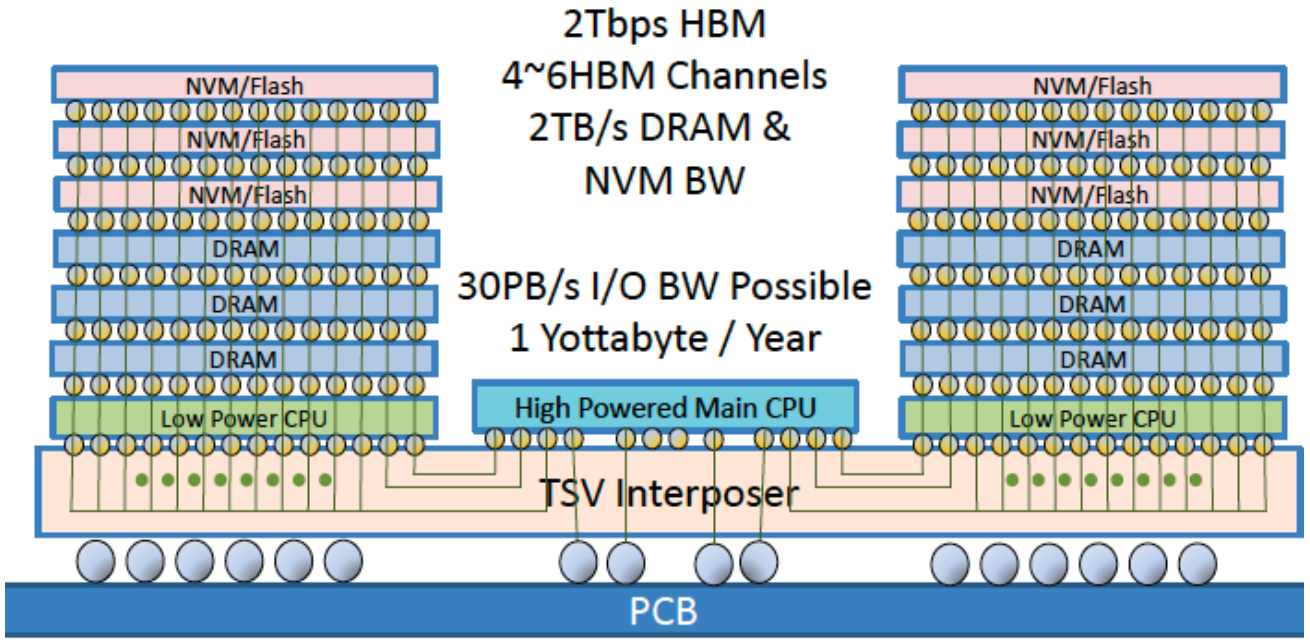
Top500 List 43 Edition



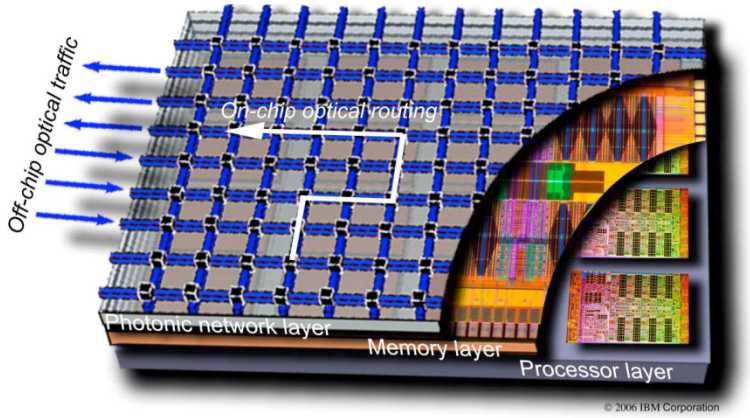
Top500 List 53 Edition



Problem with Multicore



- **Next generation will be more integrated, 3D design with a photonic network**



Today's #1 System

Systems	2014 Tianhe-2
System peak	**55 Pflop/s**
Power	**18 MW** **(3 Gflops/W)**
System memory	1.4 PB (1.024 PB CPU + .384 PB CoP)
Node performance	3.43 TF/s (.4 CPU +3 CoP)
Node concurrency	24 cores CPU + 171 cores CoP
Node Interconnect BW	6.36 GB/s
System size (nodes)	16,000
Total concurrency	3.12 M 12.48M threads (4/core)
MTTF	Few / day

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

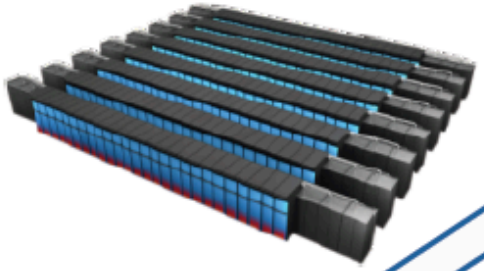
Systems	2014 Tianhe-2
System peak	**55 Pflop/s**
Power	**18 MW** **(3 Gflops/W)**
System memory	1.4 PB (1.024 PB CPU + .384 PB CoP)
Node performance	3.43 TF/s (.4 CPU +3 CoP)
Node concurrency	24 cores CPU + 171 cores CoP
Node Interconnect BW	6.36 GB/s
System size (nodes)	16,000
Total concurrency	3.12 M 12.48M threads (4/core)
MTTF	Few / day

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

Systems	2014 Tianhe-2	2020-2022	Difference Today & Exa
System peak	**55 Pflop/s**	**1 Eflop/s**	**~20x**
Power	**18 MW** **(3 Gflops/W)**	**~20 MW** **(50 Gflops/W)**	**O(1)** **~15x**
System memory	1.4 PB (1.024 PB CPU + .384 PB CoP)	32 - 64 PB	~50x
Node performance	3.43 TF/s (.4 CPU +3 CoP)	1.2 or 15TF/s	O(1)
Node concurrency	24 cores CPU + 171 cores CoP	O(1k) or 10k	~5x - ~50x
Node Interconnect BW	6.36 GB/s	200-400GB/s	~40x
System size (nodes)	16,000	O(100,000) or O(1M)	~6x - ~60x
Total concurrency	3.12 M 12.48M threads (4/core)	O(billion)	~100x
MTTF	Few / day	Many / day	O(?)

Exascale in the USA not until 2022

DOE Facilities have a fixed 4-5 year cadence
Present Roadmap for Largest US supercomputers 2012 - 2022



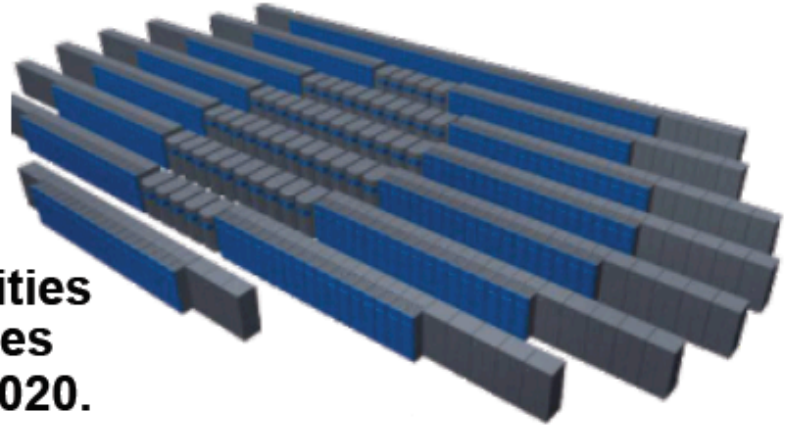
2022 CORAL-2 1000 PF

2019 Trinity-2 250-300 PF

2017 CORAL 100-200 PF

2015 Trinity 60 PF

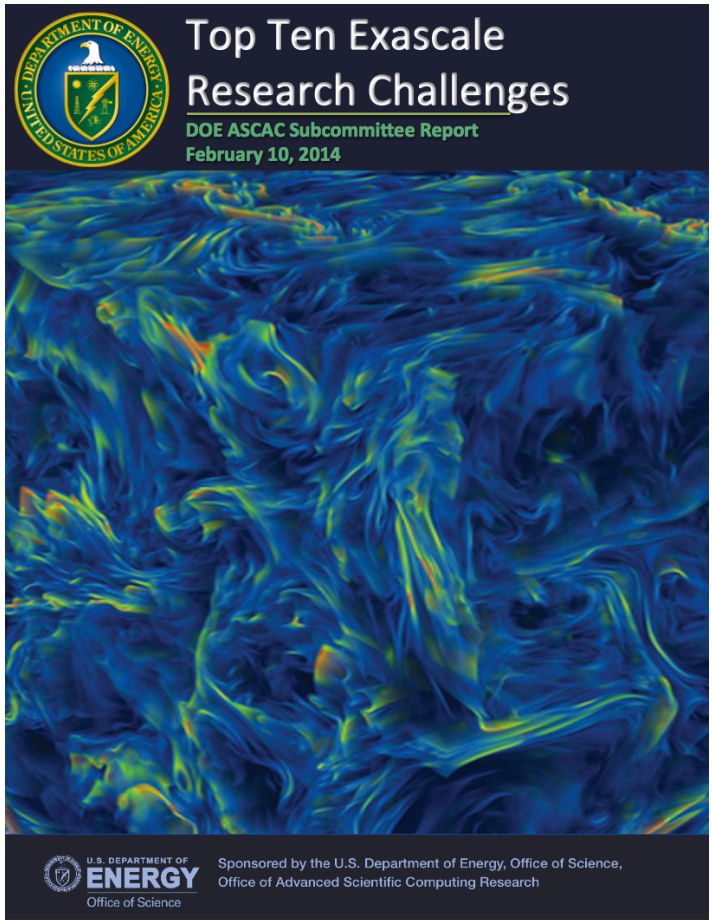
2012 Titan 26 PF and Sequoia 20PF



Power constraints of 20-30 MW facilities and pay-off schedules of 4 year leases limit accelerating this Roadmap to 2020.

Top 10 Challenges to Exascale

In a recent report U.S. Department of Energy identified ten research challenges (Google "Top 10 Challenges to Exascale")



ASCAC Subcommittee for the Top Ten Exascale Research Challenges

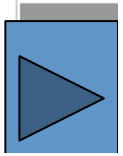
Subcommittee Chair
Robert Lucas (University of Southern California, Information Sciences Institute)

Subcommittee Members
James Ang (Sandia National Laboratories)
Keren Bergman (Columbia University)
Shekhar Borkar (Intel)
William Carlson (Institute for Defense Analyses)
Laura Carrington (UC, San Diego)
George Chiu (IBM)
Robert Colwell (DARPA)
William Dally (NVIDIA)
Jack Dongarra (U. Tennessee)
Al Geist (ORNL)
Gary Grider (LANL)
Rud Haring (IBM)
Jeffrey Hittinger (LLNL)
Adolfy Hoisie (PNLL)
Dean Klein (Micron)
Peter Kogge (U. Notre Dame)
Richard Lethin (Reservoir Labs)
Vivek Sarkar (Rice U.)
Robert Schreiber (Hewlett Packard)
John Shalf (LBNL)
Thomas Sterling (Indiana U.)
Rick Stevens (ANL)

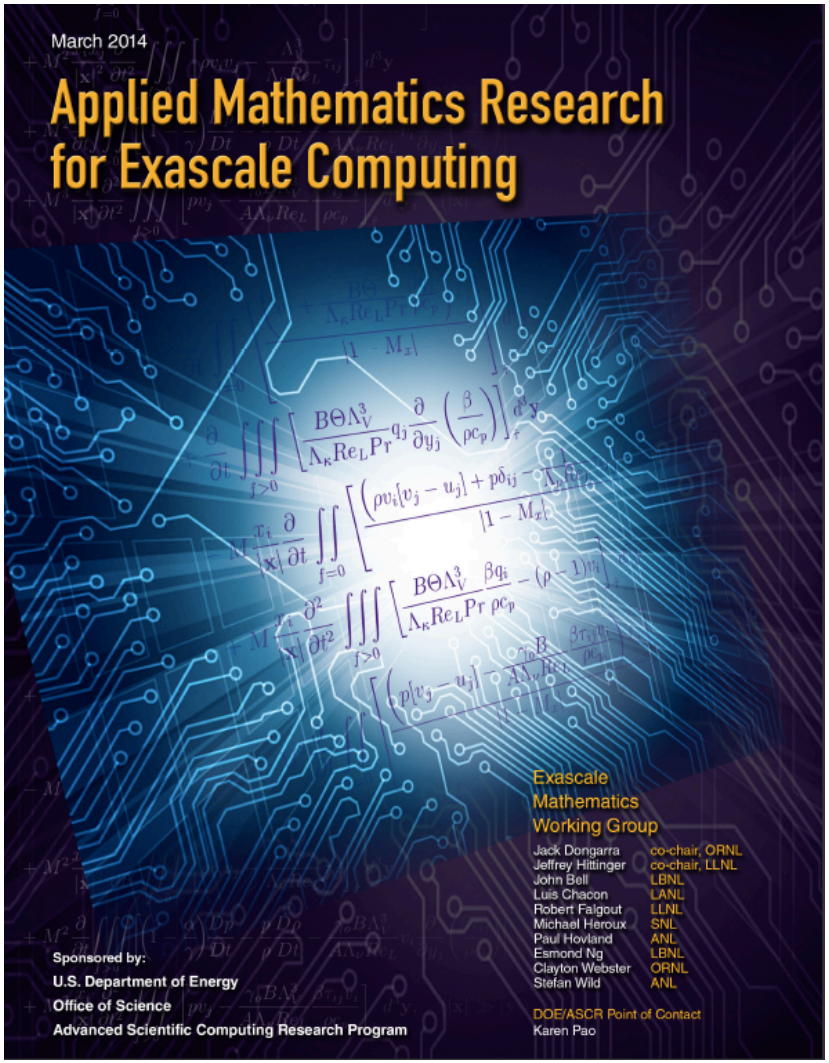
Top 10 Challenges to Exascale

3 Hardware, 4 Software, 3 Algorithms/Math Related

- **Energy efficiency:**
  - Creating more energy efficient circuit, power, and cooling technologies.
- **Interconnect technology:**
  - Increasing the performance and energy efficiency of data movement.
- **Memory Technology:**
  - Integrating advanced memory technologies to improve both capacity and bandwidth.
- **Scalable System Software:**
  - Developing scalable system software that is power and resilience aware.
- **Programming systems:**
  - Inventing new programming environments that express massive parallelism, data locality, and resilience
- **Data management:**
  - Creating data management software that can handle the volume, velocity and diversity of data that is anticipated.
- **Scientific productivity:**
  - Increasing the productivity of computational scientists with new software engineering tools and environments.
- **Exascale Algorithms:**
  - Reformulating science problems and refactoring their solution algorithms for exascale systems.
- **Algorithms for discovery, design, and decision:**
  - Facilitating mathematical optimization and uncertainty quantification for exascale discovery, design, and decision making.
- **Resilience and correctness:**
  - Ensuring correct scientific computation in face of faults, reproducibility, and algorithm verification challenges.



Applied Math @ Exascale

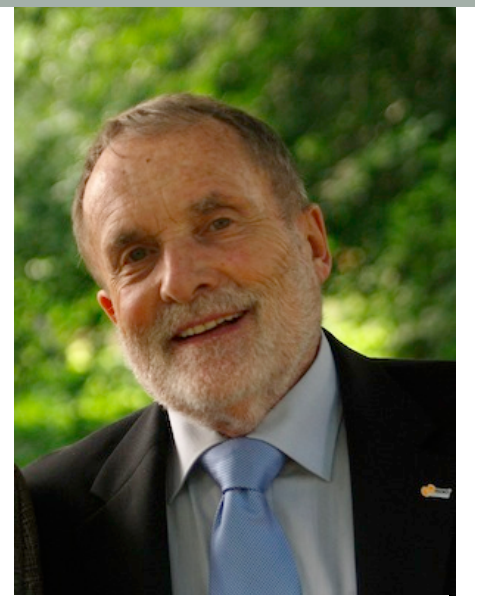


Google "doe applied math exascale"

Report Recommendations

- **DOE should proceed expeditiously and with high priority with an exascale math initiative.**
- **A significant new investment in research and development of new models, discretizations, and algorithms implemented in new science application codes is required at exascale.**
- **Not all problems require exascale computation, and yet these problems will continue to require applied mathematics research.**
- **An intensive co-design effort is essential for success, where computer scientists, applied mathematicians, and application scientists work closely together to produce a computational science discovery environment.**
- **DOE must make investments to increase the pool of computational scientists and mathematicians trained in both applied mathematics and high-performance computing.**

TOP500



- In 1986 Hans Meuer started a list of supercomputer around the world, they were ranked by peak performance.
- Hans approached me in 1992 to put together our lists into the “TOP500”.
- The first TOP500 list was in June 1993.

TOP500® SUPERCOMPUTER SITES

Rank	Site	System	Cores	Rmax (GFlop/s)	Rpeak (GFlop/s)	Power (kW)
1	Los Alamos National Laboratory United States	CM-5/1024 Thinking Machines Corporation	1,024	59.7	131.0	
2	Minnesota Supercomputer Center United States	CM-5/544 Thinking Machines Corporation	544	30.4	69.6	
3	National Security Agency United States	CM-5/512 Thinking Machines Corporation	512	30.4	65.5	
4	NCSA United States	CM-5/512 Thinking Machines Corporation	512	30.4	65.5	
5	NEC Japan	SX-3/44R NEC	4	23.2	25.6	
6	Atmospheric Environment Service (AES)	SX-3/44	4	20.0	22.0	

HPL - Bad Things

- LINPACK Benchmark is 37 years old
  - TOP500 (HPL) is 22 years old
- Floating point-intensive performs $O(n^3)$ floating point operations and moves $O(n^2)$ data.
- No longer so strongly correlated to real apps.
- Reports Peak Flops (although hybrid systems see only 1/2 to 2/3 of Peak)
- Encourages poor choices in architectural features
- Overall usability of a system is not measured
- Used as a marketing tool
- Decisions on acquisition made on one number
- Benchmarking for days wastes a valuable resource

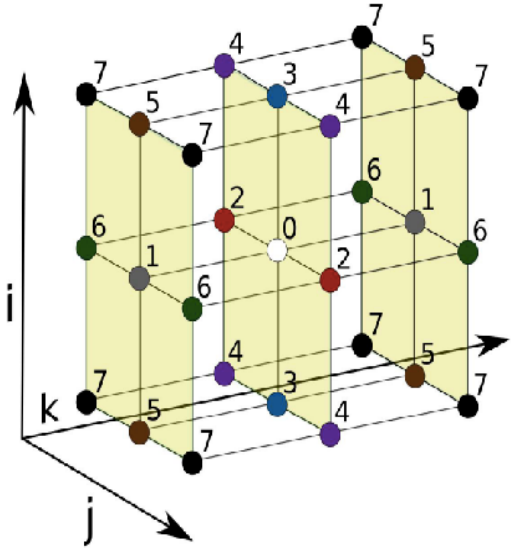
Proposal: HPCG

- High Performance Conjugate Gradient (HPCG).
- Solves $Ax=b$, A large, sparse, b known, x computed.
- An optimized implementation of PCG contains essential computational and communication patterns that are prevalent in a variety of methods for discretization and numerical solution of PDEs

- Patterns:
  - Dense and sparse computations.
  - Dense and sparse collective.
  - Multi-scale execution of kernels via MG (truncated) V cycle.
  - Data-driven parallelism (unstructured sparse triangular solves).
- Strong verification and validation properties (via spectral properties of PCG).

Model Problem Description

- Synthetic discretized 3D PDE (FEM, FVM, FDM).
- Single DOF heat diffusion model.
- Zero Dirichlet BCs, Synthetic RHS s.t. solution = 1.
- Local domain: $(n_x \times n_y \times n_z)$
- Process layout: $(np_x \times np_y \times np_z)$
- Global domain: $(n_x * np_x) \times (n_y * np_y) \times (n_z * np_z)$
- Sparse matrix:
  - 27 nonzeros/row interior.
  - 7 – 18 on boundary.
  - Symmetric positive definite.



27-point stencil operator

HPL vs. HPCG: Bookends

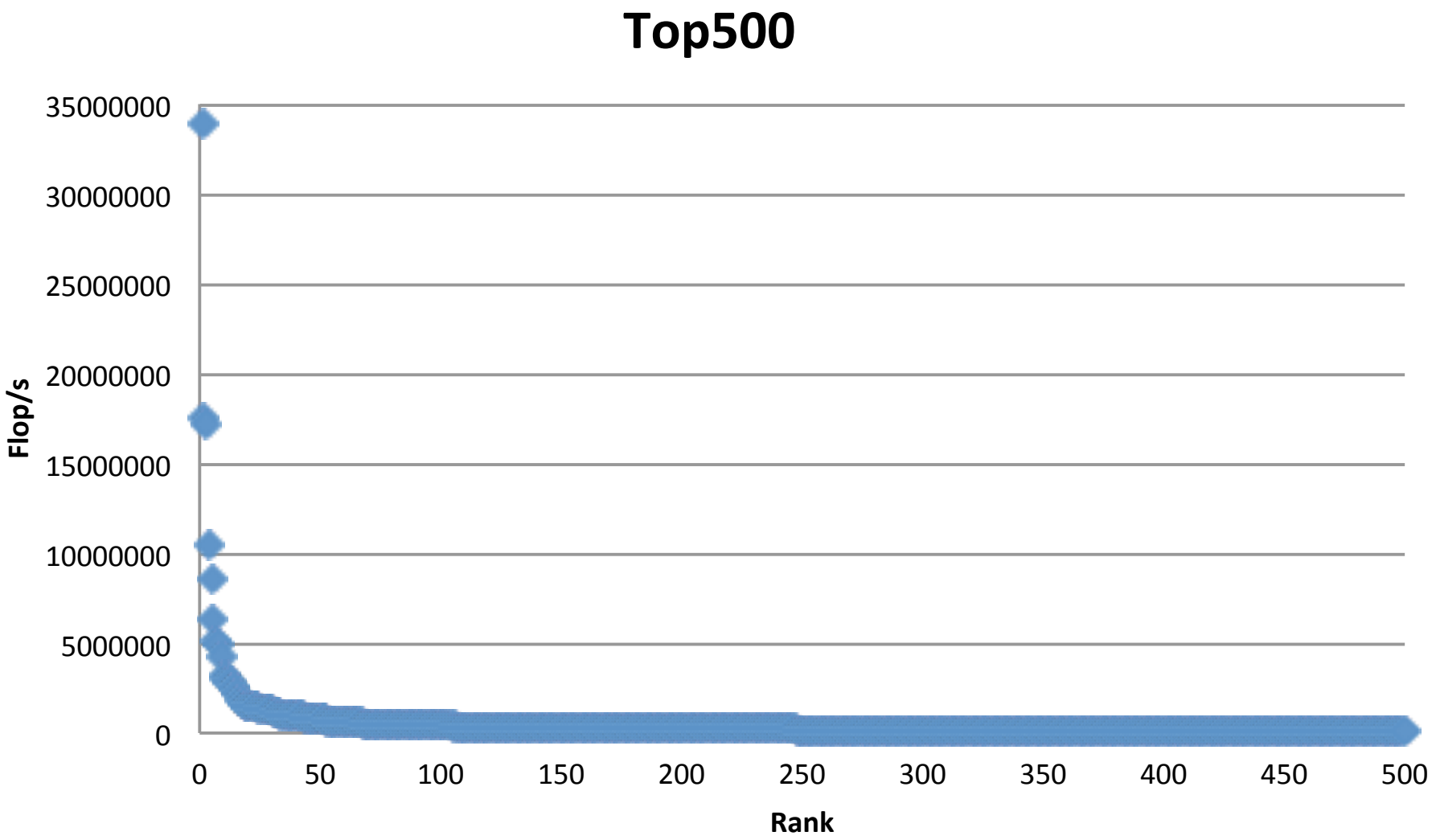
- Some see HPL and HPCG as “bookends” of a spectrum.
  - Applications teams know where their codes lie on the spectrum.
  - Can gauge performance on a system using both HPL and HPCG numbers.
- Problem of HPL execution time still an issue:
  - Need a lower cost option. End-to-end HPL runs are too expensive.
  - Work in progress.

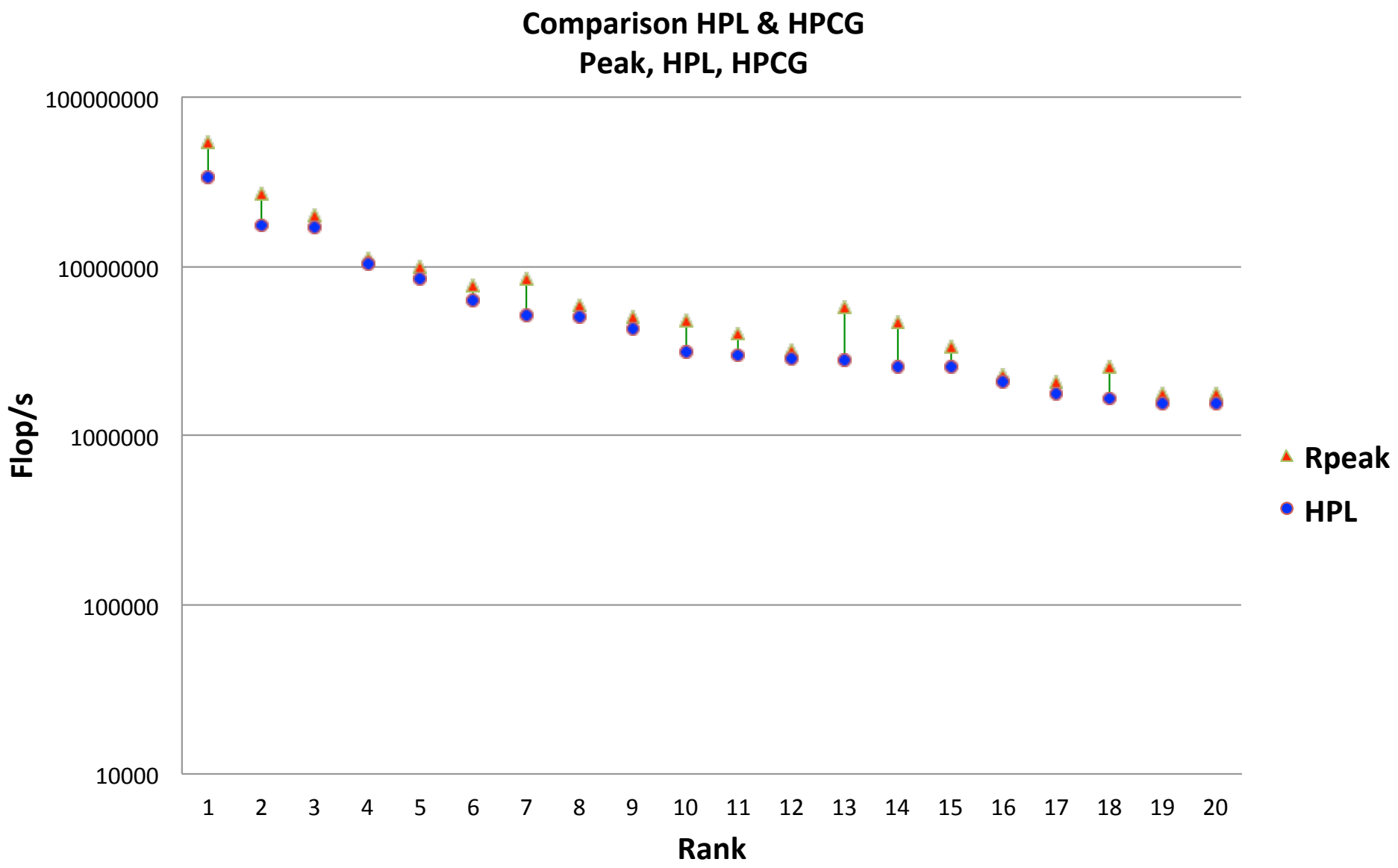
Site	Computer	Cores	HPL Rmax (Pflops)	HPL Rank	HPCG (Pflops)	HPCG/ HPL
NSCC / Guangzhou	Tianhe-2 NUDT, Xeon 12C 2.2GHz + Intel Xeon Phi 57C + Custom	3,120,000	33.9	1	.580	1.7%
RIKEN Advanced Inst for Comp Sci	K computer Fujitsu SPARC64 VIIIfx 8C + Custom	705,024	10.5	4	.427	4.1%
DOE/OS Oak Ridge Nat Lab	Titan, Cray XK7 AMD 16C + Nvidia Kepler GPU 14C + Custom	560,640	17.6	2	.322	1.8%
DOE/OS Argonne Nat Lab	Mira BlueGene/Q, Power BQC 16C 1.60GHz + Custom	786,432	8.59	5	.101#	1.2%
Swiss CSCS	Piz Daint, Cray XC30, Xeon 8C + Nvidia Kepler 14C + Custom	115,984	6.27	6	.099	1.6%
Leibniz Rechenzentrum	SuperMUC, Intel 8C + IB	147,456	2.90	12	.0833	2.9%
CEA/TGCC-GENCI	Curie tine nodes Bullx B510 Intel Xeon 8C 2.7 GHz + IB	79,504	1.36	26	.0491	3.6%
Exploration and Production Eni S.p.A.	HPC2, Intel Xeon 10C 2.8 GHz + Nvidia Kepler 14C + IB	62,640	3.00	11	.0489	1.6%
DOE/OS L Berkeley Nat Lab	Edison Cray XC30, Intel Xeon 12C 2.4GHz + Custom	132,840	1.65	18	.0439 #	2.7%
Texas Advanced Computing Center	Stampede, Dell Intel (8c) + Intel Xeon Phi (61c) + IB	78,848	.881*	7	.0161	1.8%
Meteo France	Beaufix Bullx B710 Intel Xeon 12C 2.7 GHz + IB	24,192	.469 (.467*)	79	.0110	2.4%
Meteo France	Prolix Bullx B710 Intel Xeon 2.7 GHz 12C + IB	23,760	.464 (.415*)	80	.00998	2.4%
U of Toulouse	CALMIP Bullx DLC Intel Xeon 10C 2.8 GHz + IB	12,240	.255	184	.00725	2.8%
Cambridge U	Wilkes, Intel Xeon 6C 2.6 GHz + Nvidia Kepler 14C + IB	3584	.240	201	.00385	1.6%
TiTech	TUSBAME-KFC Intel Xeon 6C 2.1 GHz + IB	2720	.150	436	.00370	2.5%

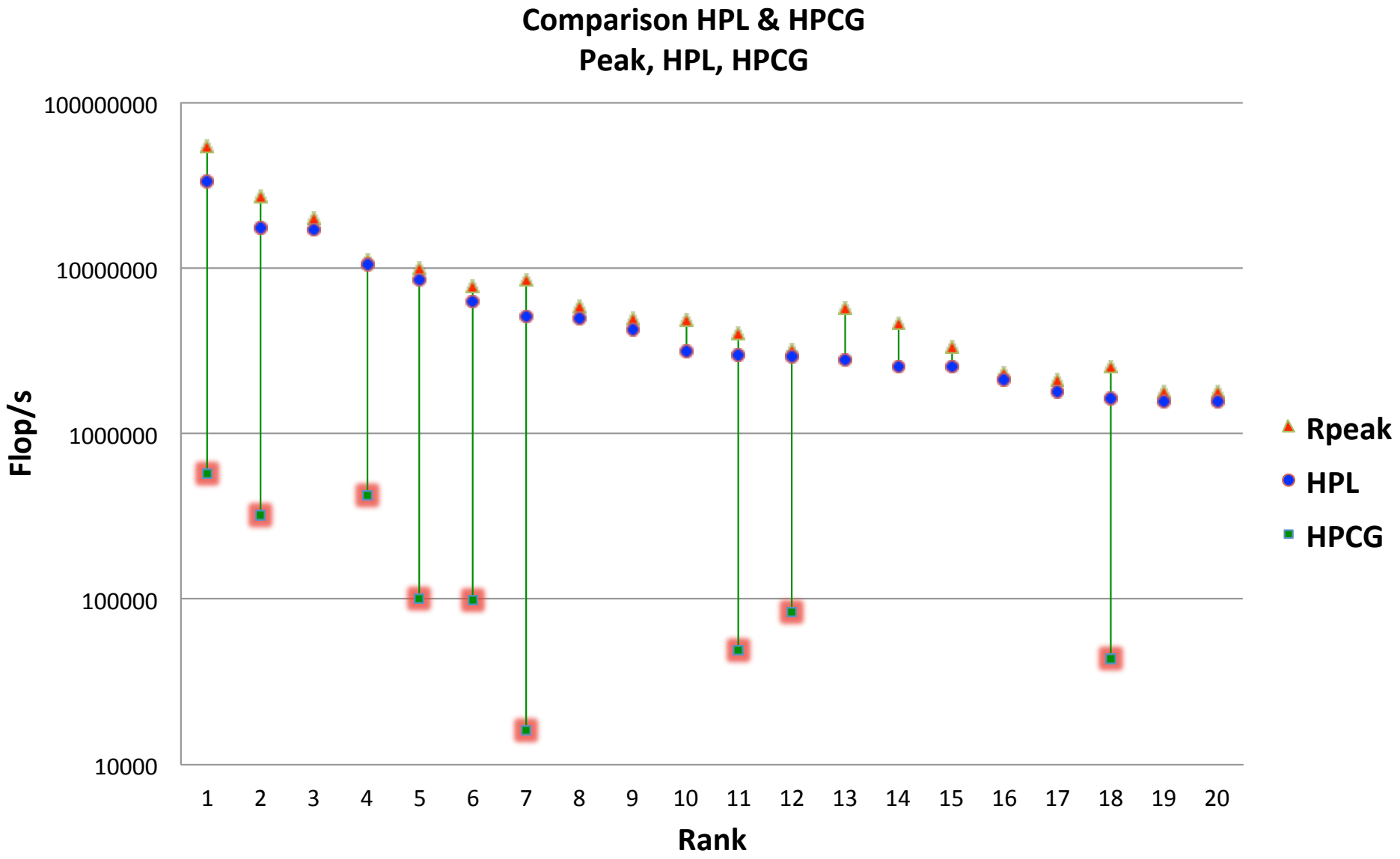
HPL

HPCG

* scaled to reflect the same number of cores
# unoptimized implementation


Top500
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
Flop/s
0
50
100
150
200
250
300
350
400
450
500
Rank


Comparison HPL & HPCG
Peak, HPL, HPCG
100000000
10000000
1000000
100000
10000
Flop/s
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Rank
Rpeak
HPL


Comparison HPL & HPCG
Peak, HPL, HPCG
100000000
10000000
1000000
100000
10000
Flop/s
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
Rank
Rpeak
HPL
HPCG

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 to adapt to the hardware
- 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.

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

Collaborators / Software / Support

- **PLASMA**
  http://icl.cs.utk.edu/plasma/
- **MAGMA**
  http://icl.cs.utk.edu/magma/
- **Quark (RT for Shared Memory)**
  http://icl.cs.utk.edu/quark/
- **PaRSEC**(Parallel Runtime Scheduling and Execution Control)
  http://icl.cs.utk.edu/parsec/

- Collaborating partners

University of Tennessee, Knoxville
University of California, Berkeley
University of Colorado, Denver

MAGMA **PLASMA**