

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**
- **Look at some of the adjustments that are needed with Extreme Computing**

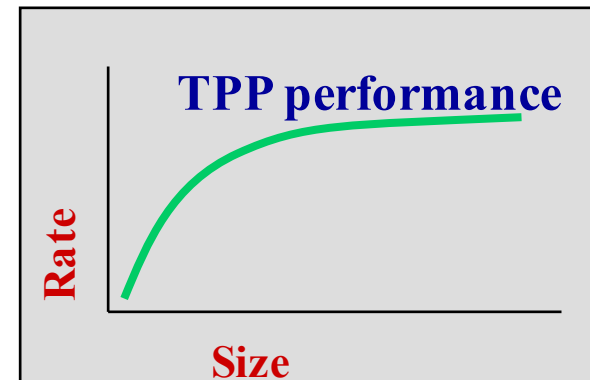
State of Supercomputing Today

- Pflops ($> 10^{15}$ Flop/s) computing fully established with 95 systems.
- Three technology architecture possibilities or “swim lanes” are thriving.
 - Commodity (e.g. Intel)
 - Commodity + accelerator (e.g. GPUs) (93 systems)
 - Lightweight cores (e.g. ShenWei, ARM, Intel’s Knights Landing)
- Interest in supercomputing is now worldwide, and growing in many new markets (around 50% of Top500 computers are used in industry).
- Exascale (10^{18} Flop/s) projects exist in many countries and regions.
- Intel processors have largest share, 91% followed by AMD, 3%.

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

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

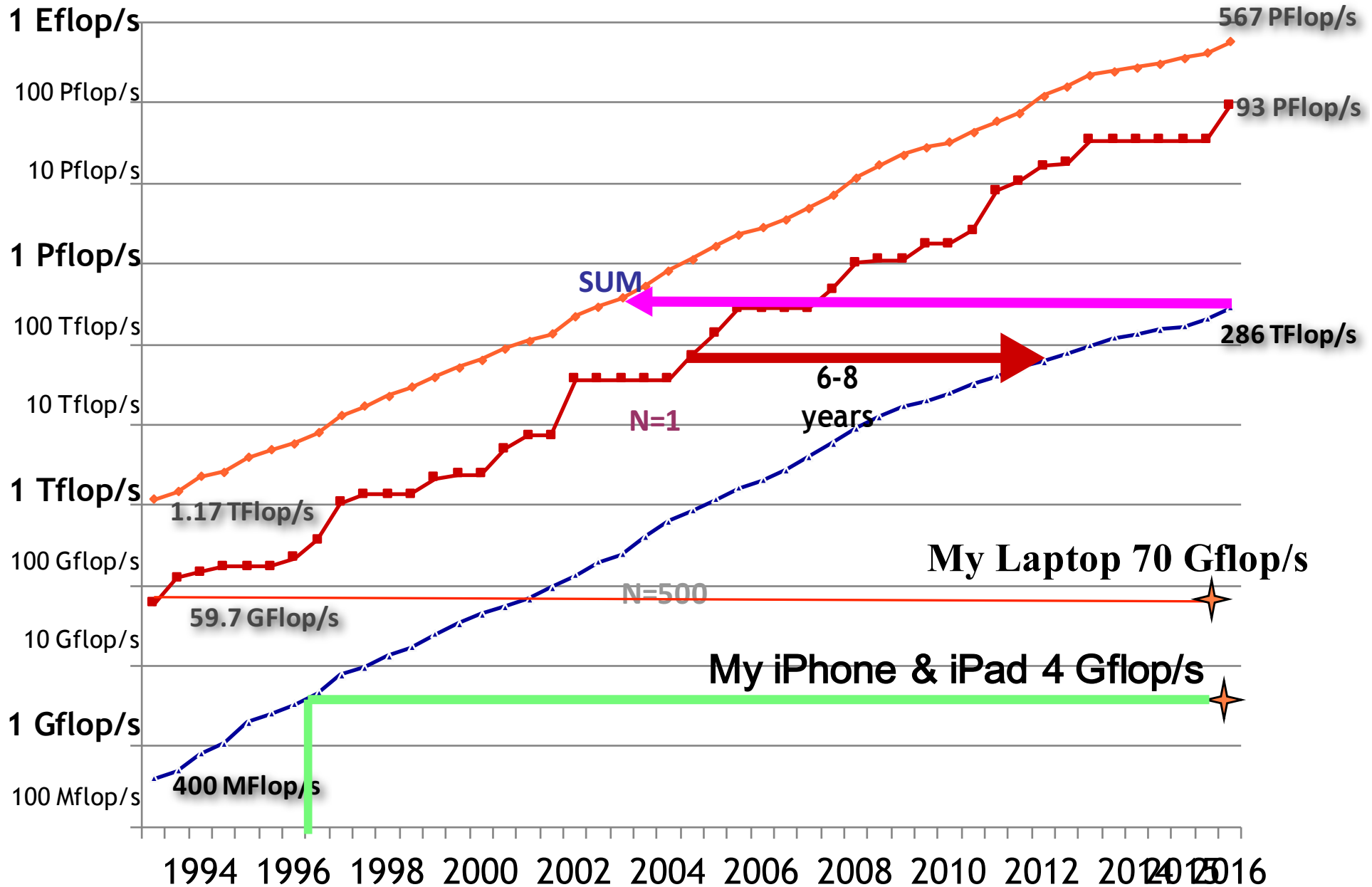
$$Ax=b, \text{ dense problem}$$



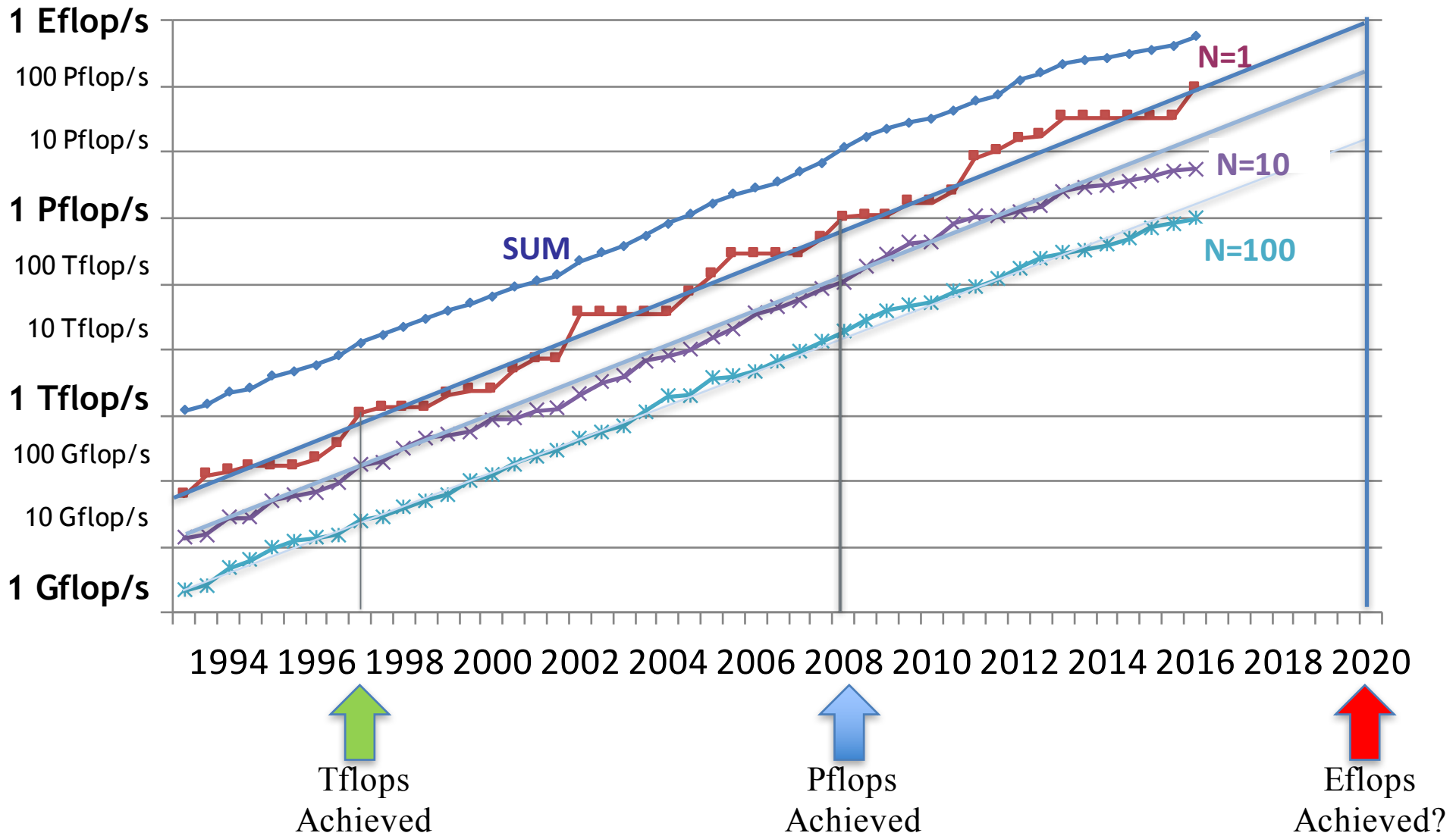
- Updated twice a year
 - SC'xy in the States in November
 - Meeting in Germany in June
- All data available from www.top500.org



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













PERFORMANCE DEVELOPMENT





June 2016: The TOP 10 Systems

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	GFlops/Watt
1	National Super Computer Center in Wuxi	Sunway TaihuLight, SW26010 (260C) + Custom	 China	10,649,000	93.0	74	15.4	6.04
2	National Super Computer Center in Guangzhou	Tianhe-2 NUDT, Xeon (12C) + Intel Xeon Phi (57c) + Custom	 China	3,120,000	33.9	62	17.8	1.91
3	DOE / OS Oak Ridge Nat Lab	Titan, Cray XK7, AMD (16C) + Nvidia Kepler GPU (14c) + Custom	 USA	560,640	17.6	65	8.21	2.14
4	DOE / NNSA L Livermore Nat Lab	Sequoia, BlueGene/Q (16C) + custom	 USA	1,572,864	17.2	85	7.89	2.18
5	RIKEN Advanced Inst for Comp Sci	K computer Fujitsu SPARC64 VIIIfx (8C) + Custom	 Japan	705,024	10.5	93	12.7	.827
6	DOE / OS Argonne Nat Lab	Mira, BlueGene/Q (16C) + Custom	 USA	786,432	8.16	85	3.95	2.07
7	DOE / NNSA / Los Alamos & Sandia	Trinity, Cray XC40, Xeon (16C) + Custom	 USA	301,056	8.10	80	4.23	1.92
8	Swiss CSCS	Piz Daint, Cray XC30, Xeon (8C) + Nvidia Kepler (14c) + Custom	 Swiss	115,984	6.27	81	2.33	2.69
9	HLRS Stuttgart	Hazel Hen, Cray XC40, Xeon (12C) + Custom	 Germany	185,088	5.64	76	3.62	1.56
10	KAUST	Shaheen II, Cray XC40, Xeon (16C) + Custom	 Saudi Arabia	196,608	5.54	77	2.83	1.96

500 Internet company

Inspur Intel (8C) + Nvidia

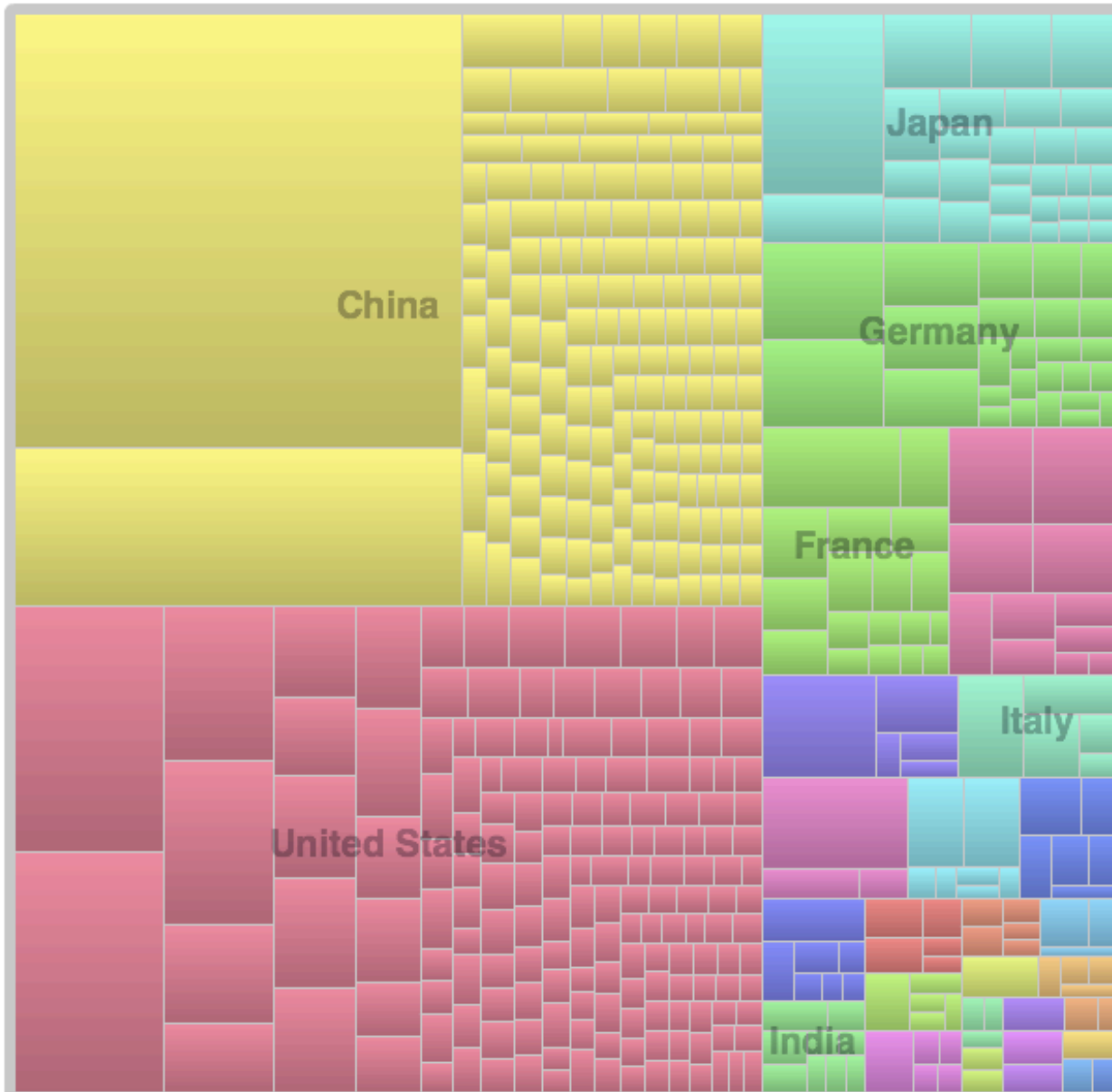
China

5440

.286

71

Countries Share



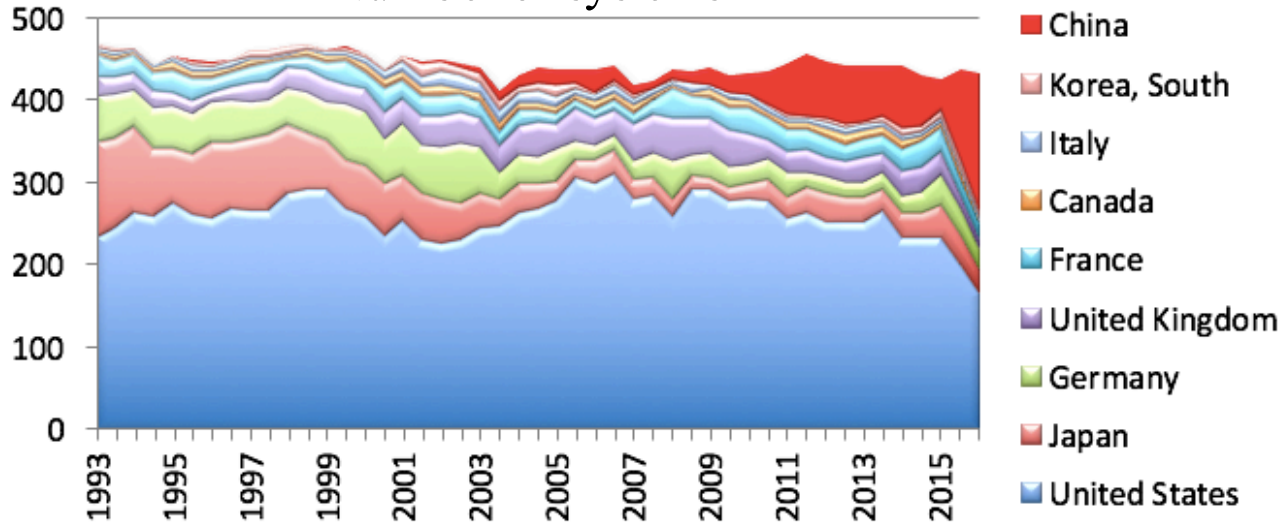
COUNTRY	NUMBER OF SUPERCOMPUTERS
China	167
United States	165
Japan	29
Germany	26
France	18
Britain	12
India	9
Russia	7
South Korea	7
Poland	6
other	54

China has 1/3 of the systems, while the number of systems in the US has fallen to the lowest point since the TOP500 list was created.

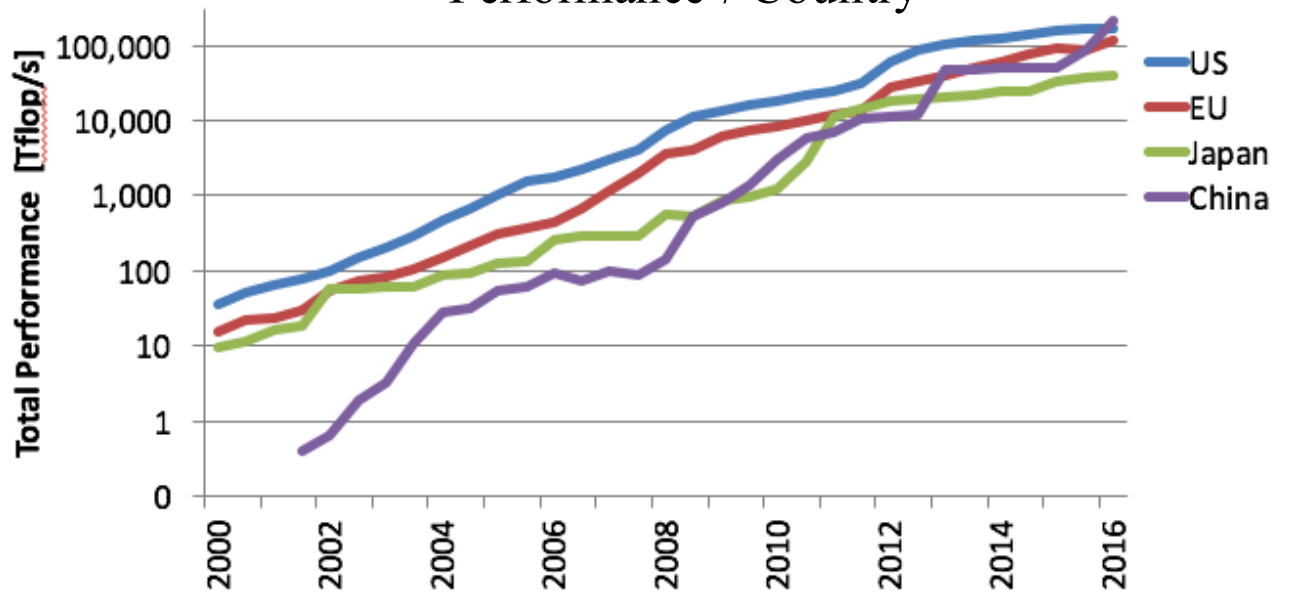
Rank	Name	Computer	Site	Total Cores	Rmax
9	Hazel Hen	Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect	HLRS - Hochleistungsrechenzentrum Stuttgart	185088	5640170
13	JUQUEEN	BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect	Forschungszentrum Juelich (FZJ)	458752	5008857
27	SuperMUC	iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR	Leibniz Rechenzentrum	147456	2897000
28	SuperMUC Phase 2	NeXtScale nx360M5, Xeon E5-2697v3 14C 2.6GHz, Infiniband FDR14	Leibniz Rechenzentrum	86016	2813620
33	Mistral	bullx DLC 720, Xeon E5-2680v3 12C 2.5GHz/E5-2695V4 18C 2.1Ghz, Infiniband FDR	DKRZ - Deutsches Klimarechenzentrum	88992	2542150
57	JURECA	T-Platforms V-Class, Xeon E5-2680v3 12C 2.5GHz, Infiniband EDR/ParTec ParaStation ClusterSuite, NVIDIA Tesla K80/K40	Forschungszentrum Juelich (FZJ)	49476	1424720
66		iDataPlex DX360M4, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband FDR	Max-Planck-Gesellschaft MPI/IPP	65320	1283311.9
87	Taurus	bullx DLC 720, Xeon E5-2680v3 12C 2.5GHz, Infiniband FDR	TU Dresden, ZIH	34656	1029940
96	Konrad	Cray XC40, Intel Xeon E5-2695v2/E5-2680v3 12C 2.4/2.5GHz, Aries interconnect	HLRN at ZIB/Konrad Zuse- Zentrum Berlin	44928	991525
114	Gottfried	Cray XC40, Intel Xeon E5-2695v2 12C 2.4GHz/E5-2680v3 12C 2.5GHz, Aries interconnect	HLRN at Universitaet Hannover / RRZN	40320	829805
126	ForHLR II	Lenovo NeXtScale nx360M5, Xeon E5-2660v3 10C 2.6GHz, Infiniband EDR/FDR	Karlsruher Institut für Technologie (KIT)	22960	768336
145		iDataPlex DX360M4, Intel Xeon E5-2680v2 10C 2.800GHz, Infiniband, NVIDIA K20x	Max-Planck-Gesellschaft MPI/IPP	15840	709700
214	NEMO bwForCluster	Dalco H88 Cluster, Xeon E5-2630v4 10C 2.2GHz, Intel Omni-Path	Universitaet Freiburg	15120	525714
279	magnitUDE	NEC Cluster, Xeon E5-2650v4 12C 2.2GHz, Intel Omni-Path	University of Duisburg-Essen	13536	437705
327	HPC4	HP POD - Cluster Platform BL460c, Intel Xeon E5-2697v2 12C 2.7GHz, Infiniband FDR	Airbus	21120	400413
334		Cray XC40, Intel Xeon E5-2670v2 10C 2.5GHz/E5-2680v3 12C 2.5Ghz, Aries interconnect	Deutscher Wetterdienst	17648	390568
335		Cray XC40, Intel Xeon E5-2670v2 10C 2.5GHz/E5-2680v3 12C 2.5Ghz, Aries interconnect	Deutscher Wetterdienst	17648	390568
336		Cluster Platform 3000 BL460c Gen8, Intel Xeon E5-2697v2 12C 2.7GHz, Infiniband FDR	Aerospace Company (E)	21240	389507.6
356	CoolMUC 2	NeXtScale nx360M5, Xeon E5-2697v3 14C 2.6GHz, Infiniband FDR14	Leibniz Rechenzentrum	11200	366357
361	Ollie	Cray CS400, Xeon E5-2697v4 18C 2.3GHz, Omni-Path	Alfred Wegener Institute, Helmholtz Centre for Polar and Marine Research	11232	364165
362	EOS	Cray CS400, Xeon E5-2698v3 16C 2.3GHz, Infiniband FDR	Max-Planck-Gesellschaft MPI/IPP	12800	363951
413	BinAC GPU	MEGWARE MiriQuid, Xeon E5-2680v4 14C 2.4GHz, Infiniband FDR, NVIDIA Tesla K80	Universitaet Tuebingen	11184	334800
440		ASUS ESC4000 FDR/G2S, Intel Xeon E5-2690v2 10C 3GHz, Infiniband FDR, AMD FirePro S9150	GSI Helmholtz Center	10976	316700
463	Minerva	Clustervision MD30-RS0, Xeon E5-2630v3 8C 2.4GHz, Intel Omni-Path	Max-Planck-Gesellschaft MPI/Albert-Einstein-Institut	9504	302416
467	LOEWE-CSC	SuperServer 2022TG-GIBQRF, Opteron 6172 12C 2.1GHz, Infiniband QDR, ATI HD 5870	Universitaet Frankfurt	44928	299300

Countries Share

Number of systems

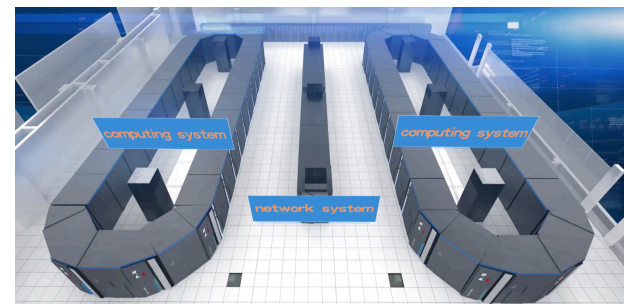
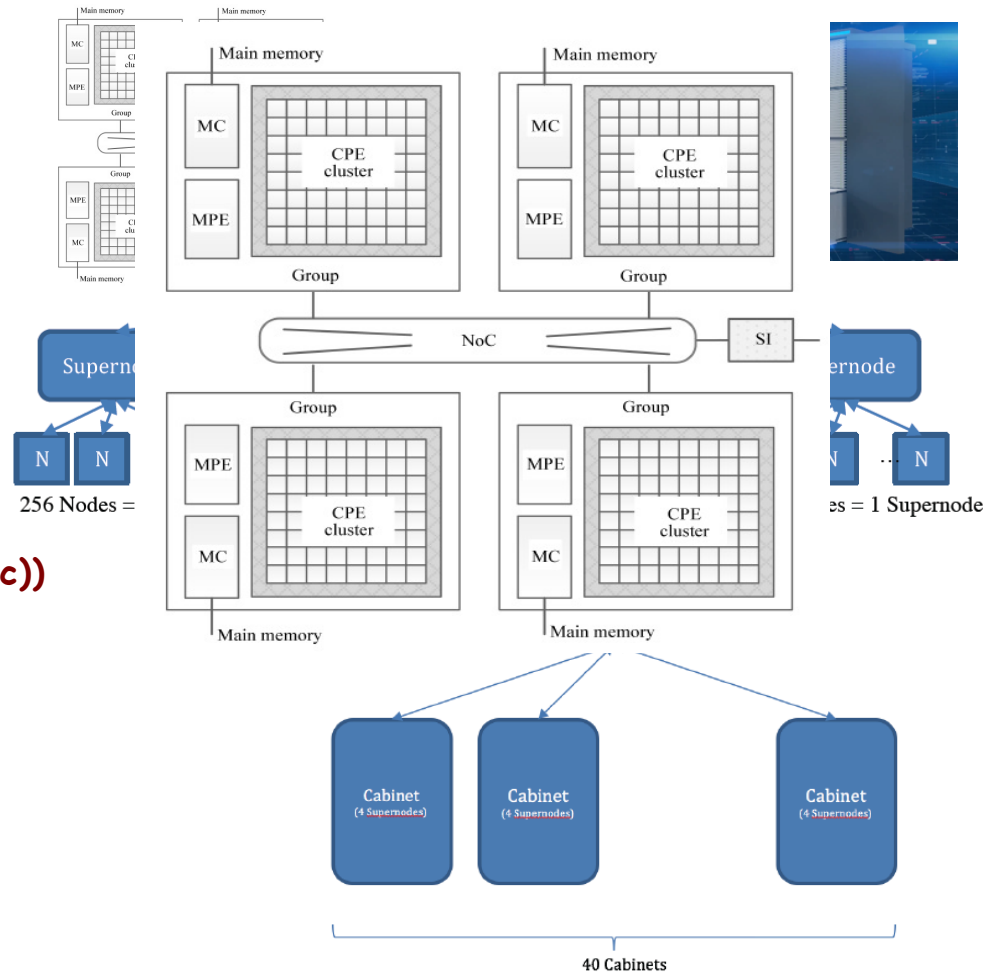


Performance / Country



Sunway TaihuLight <http://bit.ly/sunway-2016>

- SW26010 processor
- Chinese design, fab, and ISA
- 1.45 GHz
- Node = 260 Cores (1 socket)
 - **4 - core groups**
 - 64 CPE, No cache, 64 KB scratchpad/CG
 - Each core of CPE independent w/own inst stream
 - 1 MPE w/32 KB L1 dcache & 256KB L2 cache
 - **32 GB memory total, 136.5 GB/s**
 - **~3 Tflop/s, (22 flops/byte)**
- **Cabinet = 1024 nodes**
 - **4 supernodes=32 boards(4 cards/b(2 node/c))**
 - **~3.14 Pflop/s**
- **40 Cabinets in system**
 - **40,960 nodes total**
 - **125 Pflop/s total peak**
- **10,649,600 cores total**
- **1.31 PB of primary memory (DDR3)**
- **93 Pflop/s HPL, 74% peak**
- **0.32 Pflop/s HPCG, 0.3% peak**
- **15.3 MW, water cooled**
 - **6.07 Gflop/s per Watt**
- **3 of the 6 finalists Gordon Bell Award@SC16**
- **1.8B RMBs ~ \$280M, (building, hw, apps, sw, ...)**



Apps Running on Sunway TaihuLight

Table 4 Summary of the major applications on the Sunway TaihuLight, compared with similar applications on other large-scale systems

Category	System	Application summary	Scale of run	Performance
Non-linear solver	Sunway TaihuLight	A fully-implicit nonhydrostatic dynamic for cloud-resolving atmospheric simulation	131072 MPEs and 8388608 CPEs	1.5 PFlops
	Sequoia	An implicit solver for complex PDEs in highly heterogeneous flow in Earth's mantle [3]	1572864 cores	687 TFlops
Molecular dynamics	Sunway TaihuLight	Atomic simulation of silicon nanowires	131072 MPEs and 8388608 CPEs	14.7 PFlops
	Tianhe-1A	Molecular dynamics simulation of crystalline silicon [20]	7168 GPUs (3211264 CUDA cores)	1.87 PFlops
Phase-field simulation	Sunway TaihuLight	Coarsening dynamics based on Cahn-Hilliard equation with degenerated mobility	131072 MPEs and 8388608 CPEs	39.678 PFlops
	Tsubame 2.0	Dendritic solidification [6]	16000 CPU cores and 4000 GPUs (1792000 CUDA cores)	1.017 PFlops

hpcg-benchmark.org

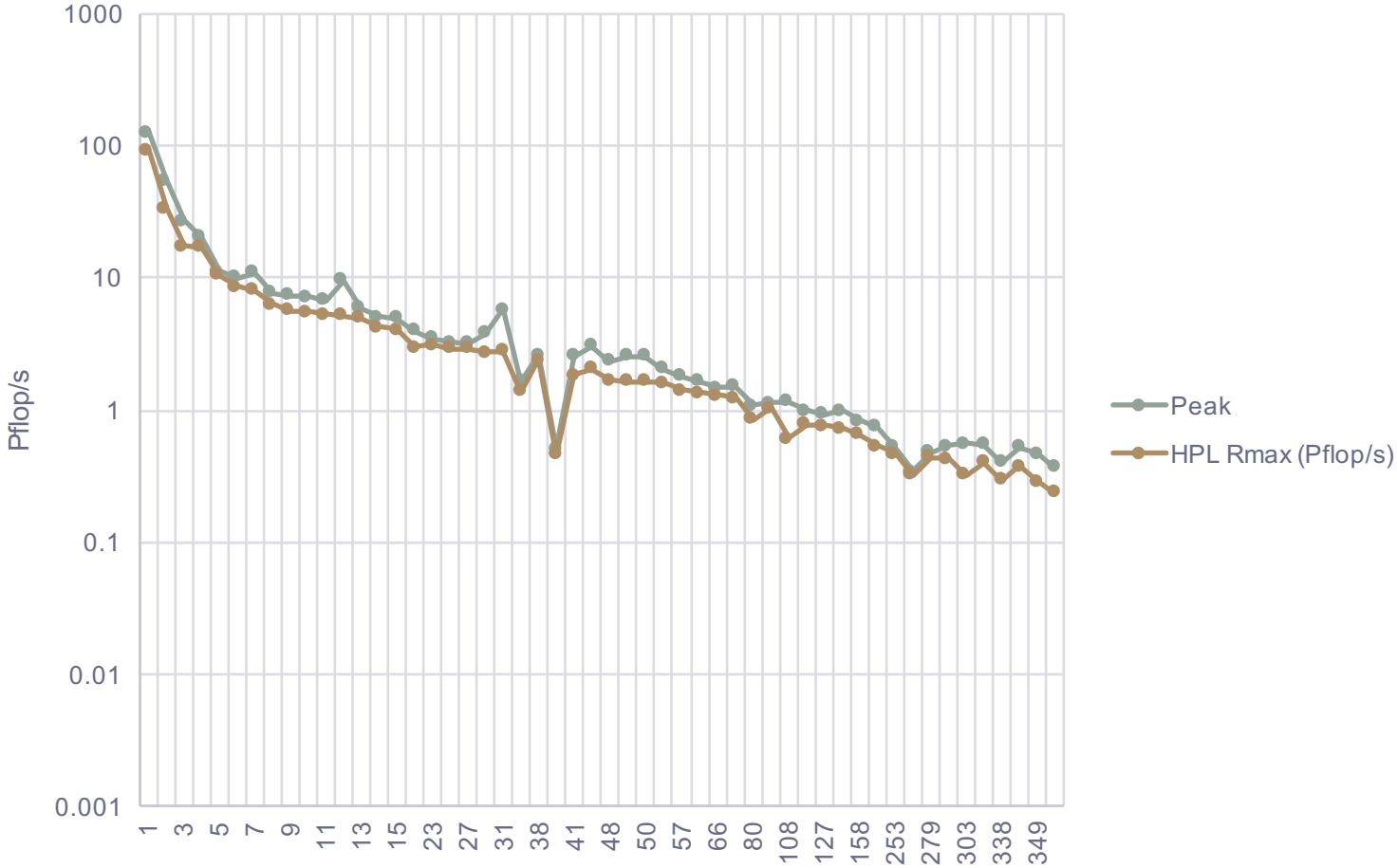
HPCG Snapshot

- High Performance Conjugate Gradients (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 collectives.
 - Multi-scale execution of kernels via MG (truncated) V cycle.
 - Data-driven parallelism (unstructured sparse triangular solves).
- Strong verification (via spectral properties of PCG).

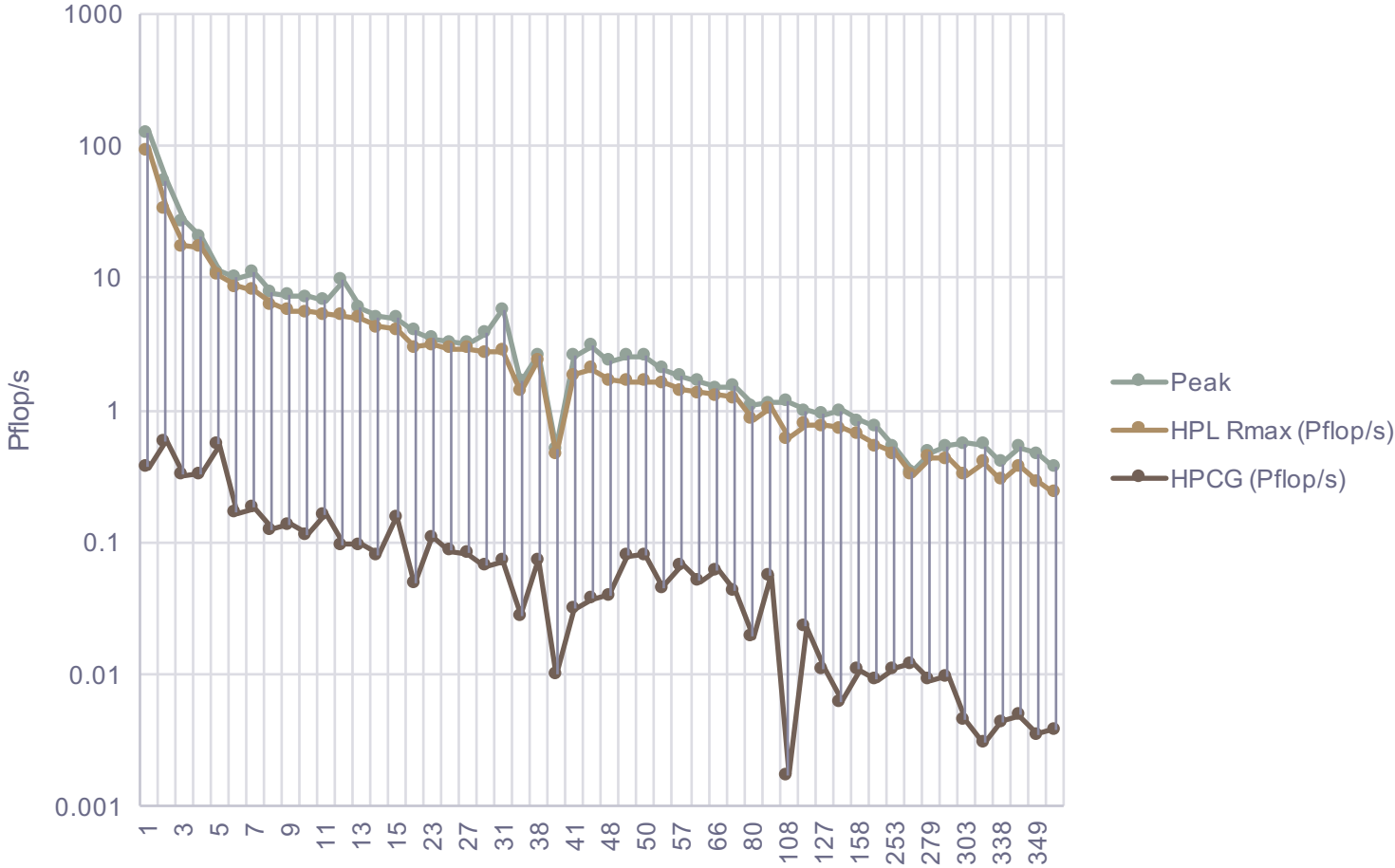
HPCG with 80 Entries

Rank (HPL)	Site	Computer	Cores	Rmax	HPCG	HPCG / HPL	% of Peak
1 (2)	NSCC / Guangzhou	Tianhe-2 NUDT, Xeon 12C 2.2GHz + Intel Xeon Phi 57C + Custom	3,120,000	33.86	0.580	1.7%	1.1%
2 (5)	RIKEN AICS	K computer, SPARC64 VIIIfx 2.0GHz, custom	705,024	10.51	0.554	5.3%	4.9%
3 (1)	NCSS / Wuxi	Sunway TaihuLight -- SW26010, Sunway	10,649,600	93.01	0.371	0.4%	0.3%
4 (4)	DOE NNSA / LLNL	Sequoia - IBM BlueGene/Q + custom	1,572,864	17.17	0.330	1.9%	1.6%
5 (3)	DOE SC / ORNL	Titan - Cray XK7 , Opteron 6274 16C 2.200GHz, custom, NVIDIA K20x	560,640	17.59	0.322	1.8%	1.2%
6 (7)	DOE NNSA / LANL& SNL	Trinity - Cray XC40, Intel E5-2698v3, + custom	301,056	8.10	0.182	2.3%	1.6%
7 (6)	DOE SC / ANL	Mira - BlueGene/Q, Power BQC 16C 1.60GHz, + Custom	786,432	8.58	0.167	1.9%	1.7%
8 (11)	TOTAL	Pangea -- Intel Xeon E5-2670, lfb FDR	218592	5.28	0.162	3.1%	2.4%
9 (15)	NASA / Mountain View	Pleiades - SGI ICE X, Intel E5-2680, E5-2680V2, E5-2680V3 + lfb	185,344	4.08	0.155	3.8%	3.1%
10 (9)	HLRS / U of Stuttgart	Hazel Hen - Cray XC40, Intel E5-2680v3, + custom	185,088	5.64	0.138	2.4%	1.9%

Bookends: Peak, HPL, and HPCG



Bookends: Peak, HPL, and HPCG



Peak Performance - Per Core

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

Floating point operations per cycle per core

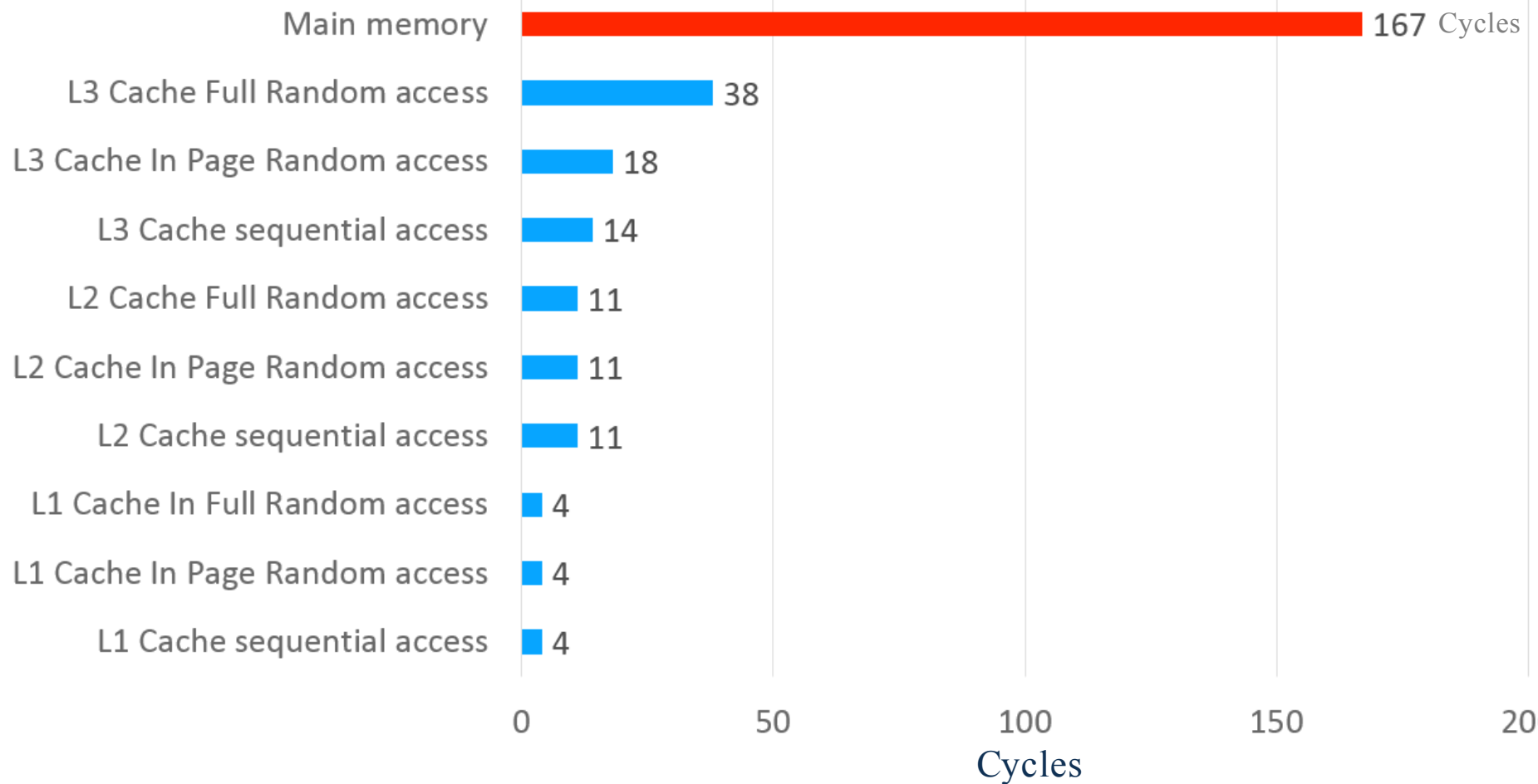
- + Most of the recent computers have FMA (Fused multiple add): (i.e. $x \leftarrow x + y * z$ in one cycle)
- + Intel Xeon earlier models and AMD Opteron have SSE2
 - + 2 flops/cycle DP & 4 flops/cycle SP
- + Intel Xeon Nehalem ('09) & Westmere ('10) have SSE4
 - + 4 flops/cycle DP & 8 flops/cycle SP
- + 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) AVX 512
 - + 32 flops/cycle DP & 64 flops/cycle SP
 - + Knight's Landing



We
are
here
(almost)

CPU Access Latencies in Clock Cycles

In 167 cycles can do 2672 DP Flops

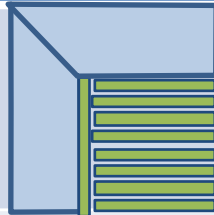


40 Years Evolving SW and Algo

Tracking Hardware Developments

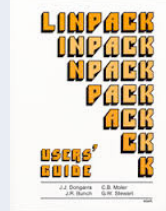
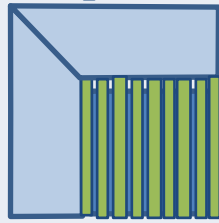
Software/Algorithms follow hardware evolution in time

EISPACK (70's)
(Translation of Algo)



Rely on
- Fortran, but row oriented

LINPACK (80's)
(Vector operations)



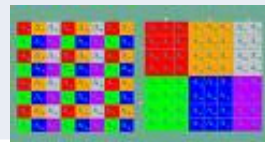
Rely on
- Level-1 BLAS operations
- Column oriented

LAPACK (90's)
(Blocking, cache friendly)



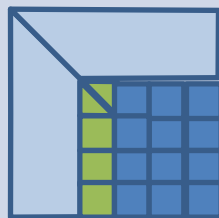
Rely on
- Level-3 BLAS operations

ScaLAPACK (00's)
(Distributed Memory)



Rely on
- PBLAS Message Passing

PLASMA, DPLASMA &
MAGMA(10's)
New Algorithms
(many-core
heterogeneous friendly)



Rely on
- DAG/scheduler
- block data layout
- some extra kernels

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.

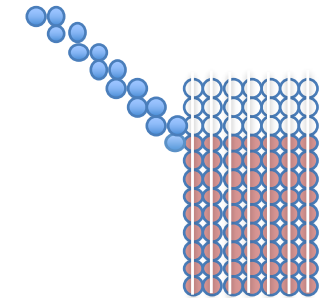
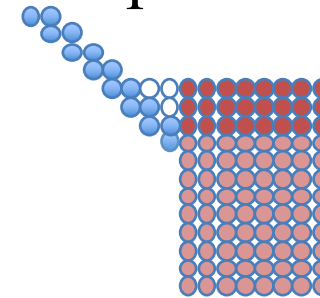
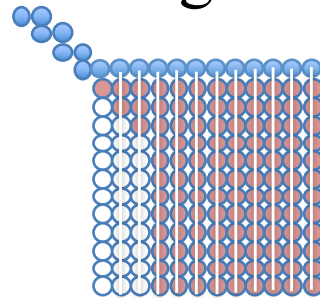
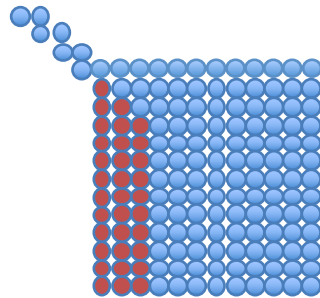
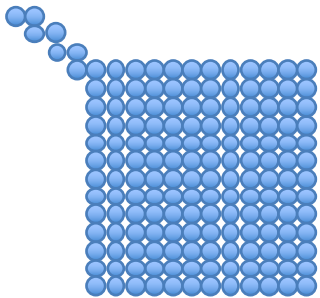
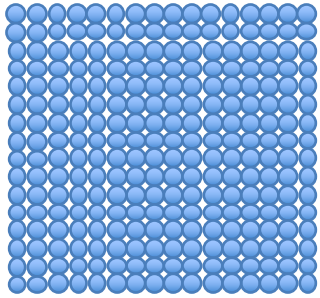


Singular Value Decomposition

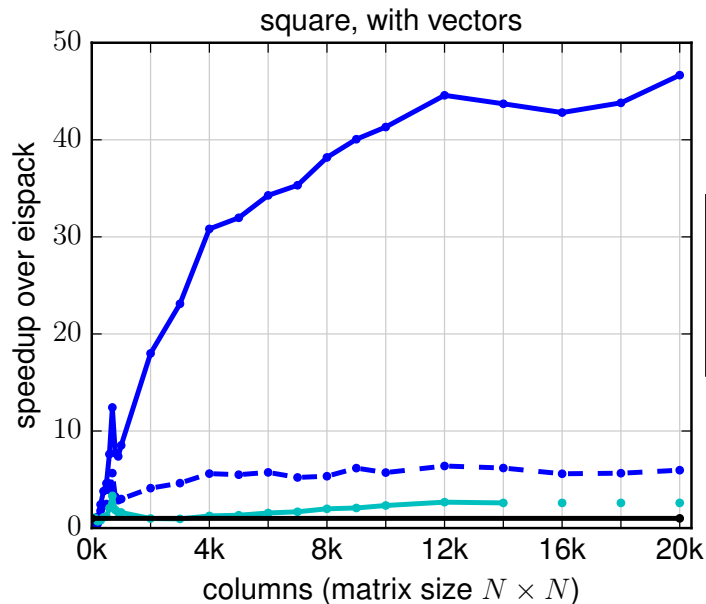
LAPACK Version 1991

Level 1, 2, & 3 BLAS

First Stage $\frac{8}{3} n^3$ Ops



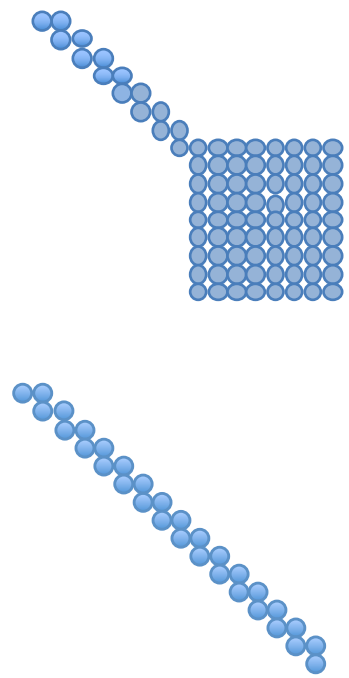
3 Generations of software compared



- LAPACK QR (BLAS in ||, 16 cores)
- LAPACK QR (using 1 core)(1991)
- LINPACK QR (1979)
- EISPACK QR (1975)

QR refers to the QR algorithm for computing the eigenvalues

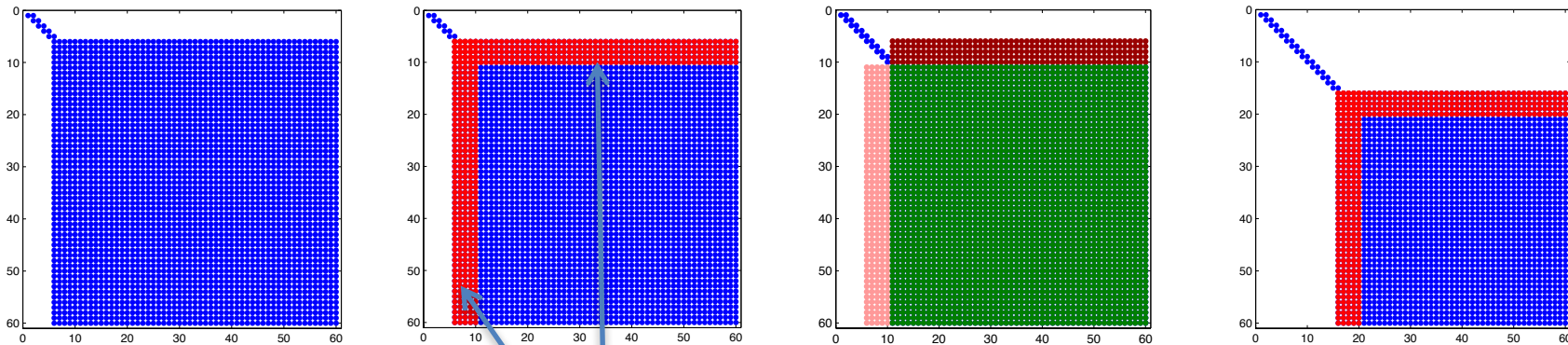
Dual socket – 8 core
Intel Sandy Bridge 2.6 GHz
(8 Flops per core per cycle)



Bottleneck in the Bidiagonalization

The Standard Bidiagonal Reduction: xGEBRD

Two Steps: Factor Panel & Update Tailing Matrix



factor panel k

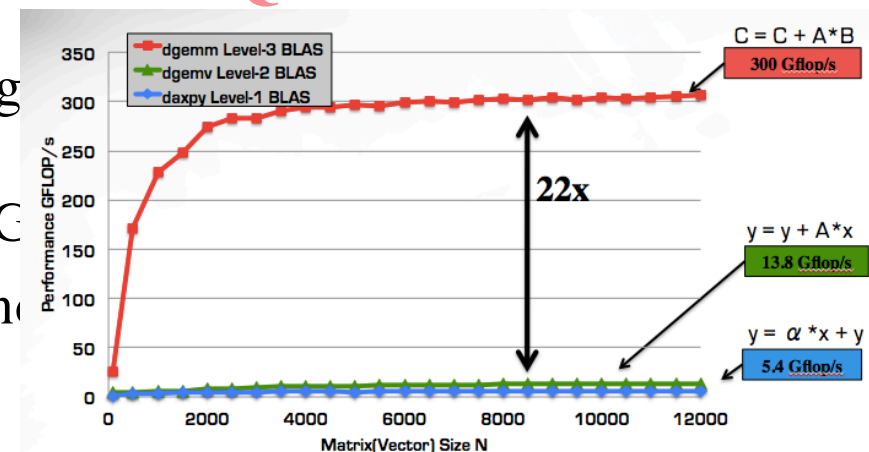
Requires 2 GEMVs

then update → factor panel k+1

★ Characteristics

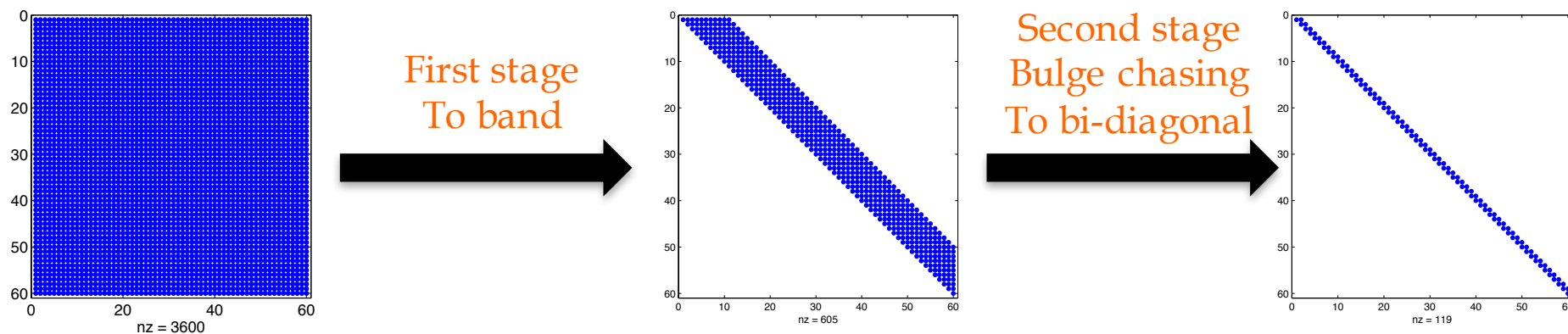
- Total cost $8n^3/3$, (reduction to bi-diag)
- Too many Level 2 BLAS operations
- $4/3 n^3$ from GEMV and $4/3 n^3$ from C
- Performance limited to 2* performance
- → **Memory bound algorithm.**

$$Q * A * P^H$$



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

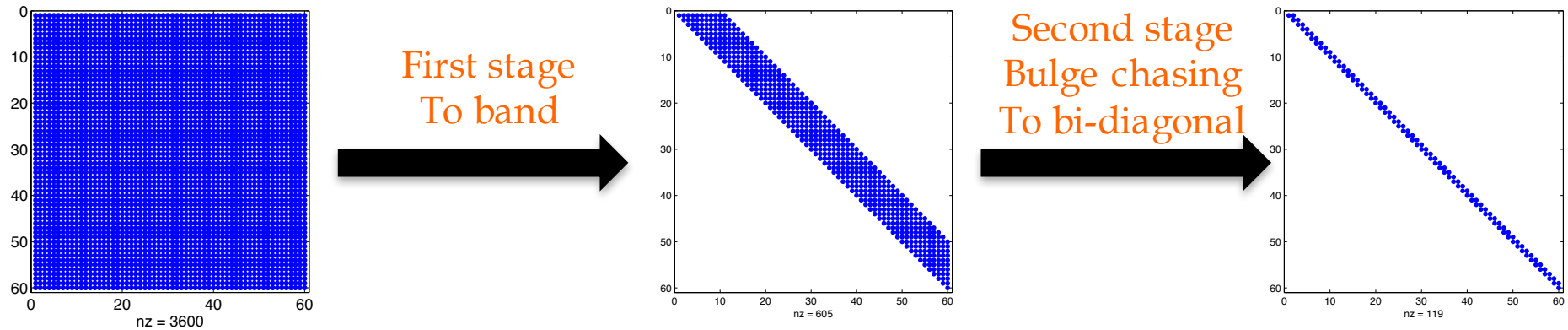
Recent Work on 2-Stage Algorithm



★ Characteristics

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

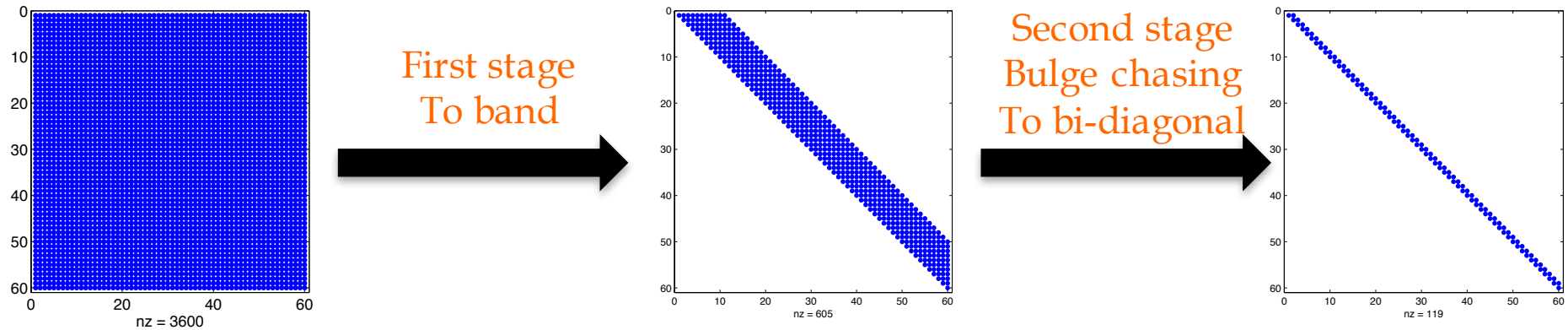
Recent work on developing new 2-stage algorithm



$$\begin{aligned}
 \text{flops} &\approx \sum_{s=1}^{\frac{n-n_b}{n_b}} 2n_b^3 + (nt-s)3n_b^3 + (nt-s)\frac{10}{3}n_b^3 + (nt-s) \times (nt-s)5n_b^3 \\
 &+ \sum_{s=1}^{\frac{n-n_b}{n_b}} 2n_b^3 + (nt-s-1)3n_b^3 + (nt-s-1)\frac{10}{3}n_b^3 + (nt-s) \times (nt-s-1)5n_b^3 \\
 &\approx \frac{10}{3}n^3 + \frac{10n_b}{3}n^2 + \frac{2n_b}{3}n^3 \\
 &\approx \frac{10}{3}n^3 (\text{gemm})_{\text{first stage}} \qquad \text{flops} = 6 \times n_b \times n^2 (\text{gemv})_{\text{second stage}}
 \end{aligned}$$

More Flops, original did $\frac{8}{3} n^3$
25% More flops

Recent work on developing new 2-stage algorithm

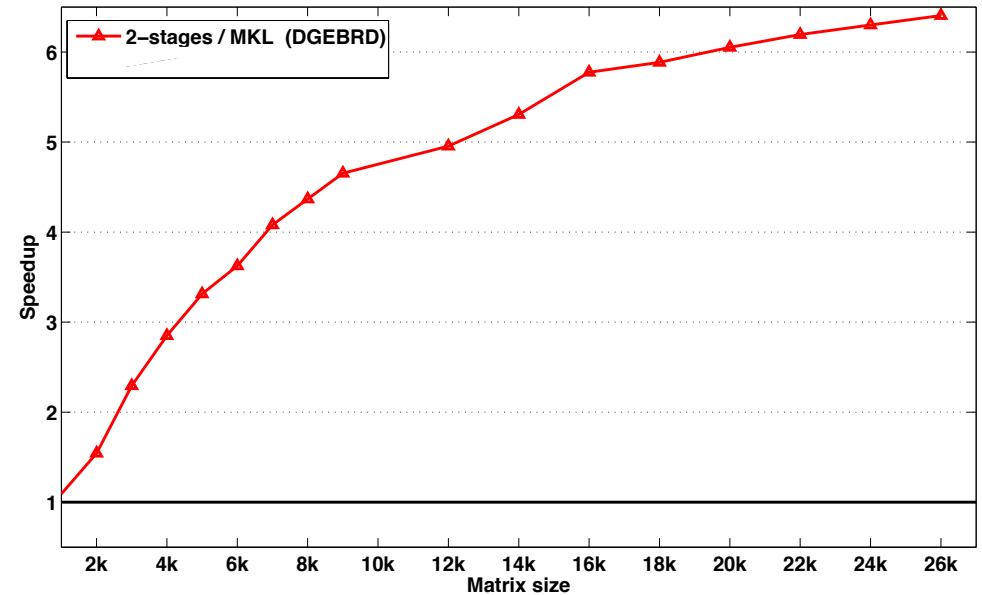


$$\text{speedup} = \frac{\text{time of one-stage}}{\text{time of two-stage}}$$

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

$$\implies \frac{84}{70} \leq \text{Speedup} \leq \frac{84}{15}$$

$$\implies 1.8 \leq \text{Speedup} \leq 7$$



16 Sandy Bridge cores 2.6 GHz

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

25% More flops and 1.8 – 7 times faster



≠



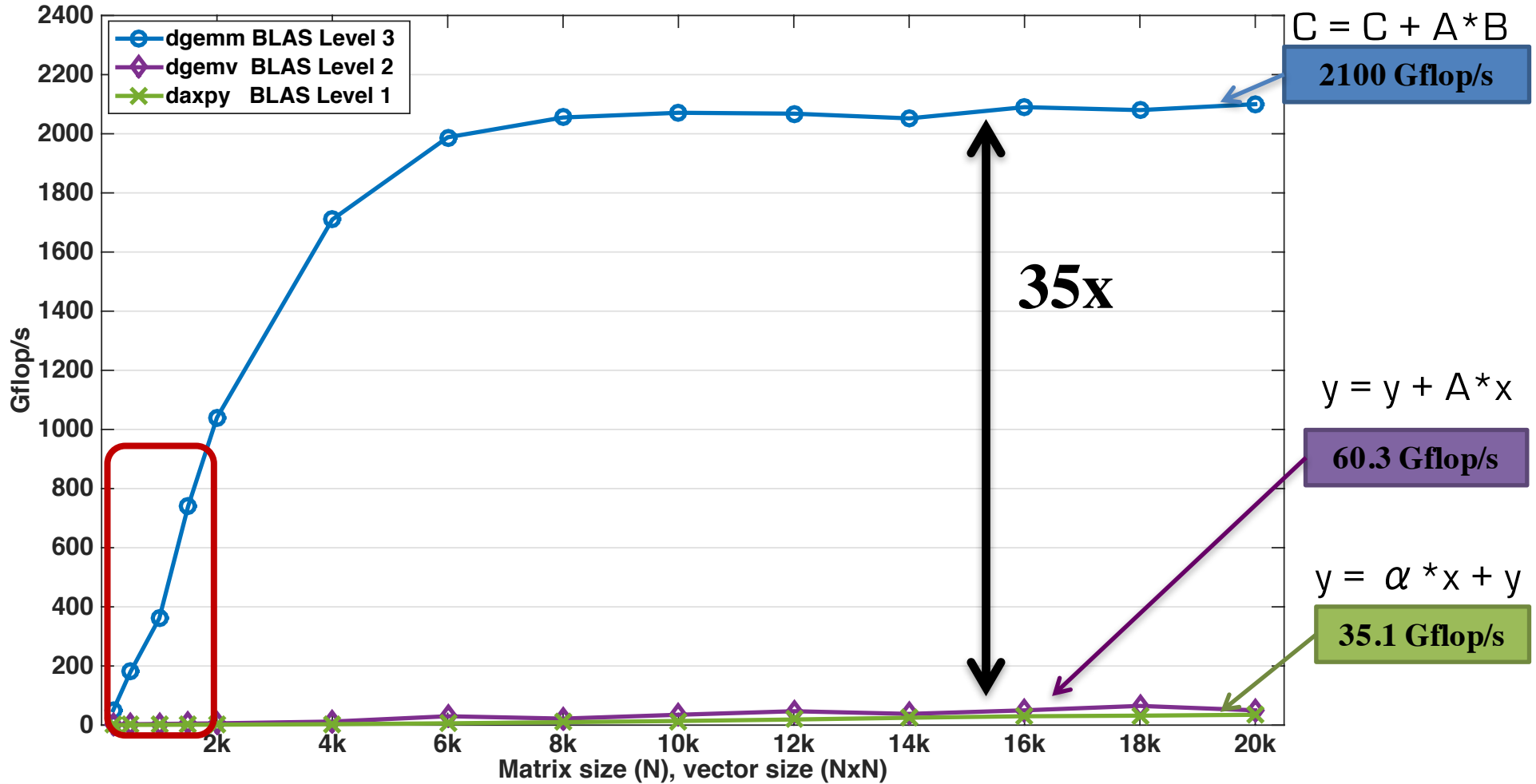
API for Batching BLAS Operations

- We are proposing, as a community standard, an API for Batched Basic Linear Algebra Operations
- The focus is on multiple independent BLAS operations
 - Think “small” matrices ($n < 500$) that are operated on in a single routine.
- Goal to be more efficient and portable for multi/manycore & accelerator systems.
- We can show 2x speedup and 3x better energy efficiency.

Level 1, 2 and 3 BLAS



64 cores Intel Xeon Phi KNL, 1.3 GHz, Peak DP = 2662 Gflop/s



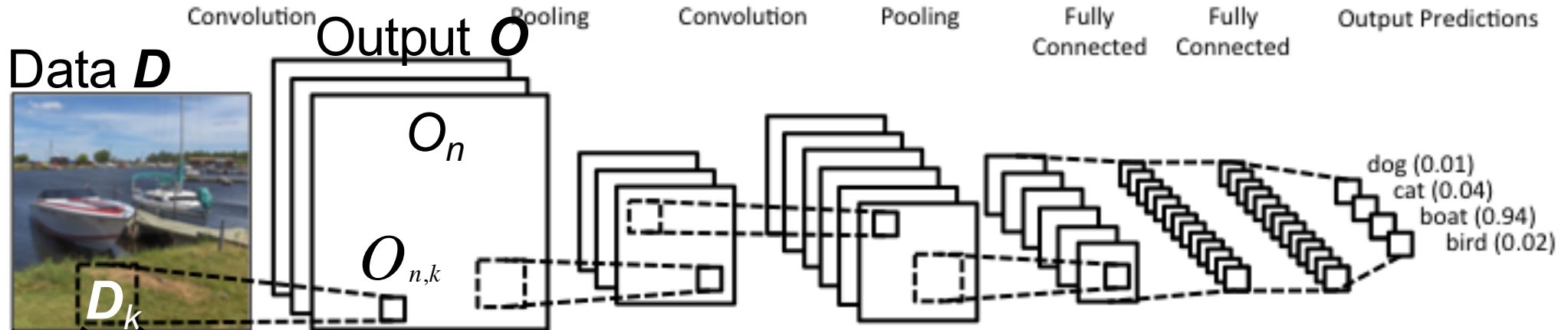
64 cores Intel Xeon Phi KNL, 1.3 GHz
 The theoretical peak double precision is 2662 Gflop/s
 Compiled with icc and using Intel MKL 2017b1 20160506

Machine Learning

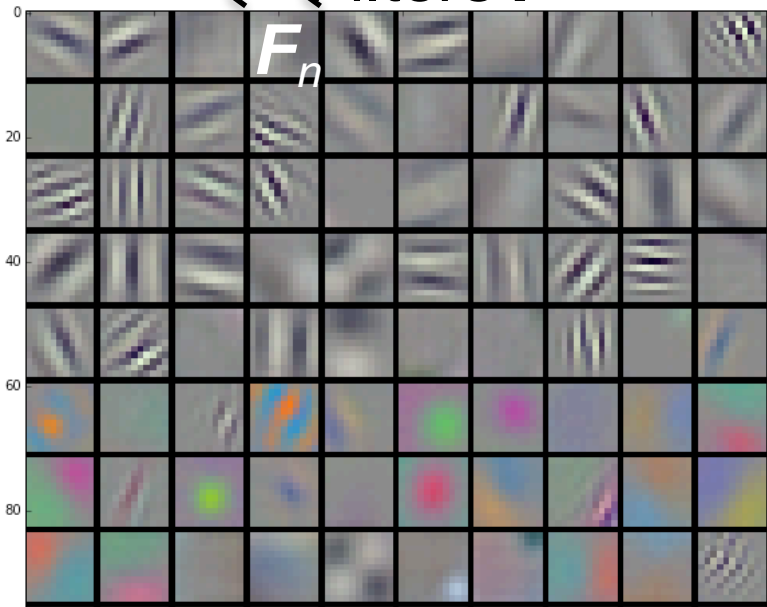
Need of Batched and/or Tensor contraction routines in machine learning

e.g., Convolutional Neural Networks (CNNs) used in computer vision

Key computation is convolution of Filter F_i (feature detector) and input image D (data):



Filters F



Convolution operation:

- For every filter F_n and every channel, the computation for every pixel value $O_{n,k}$ is a **tensor contraction**:

$$O_{n,k} = \sum D_{k,i} F_{n,i}$$

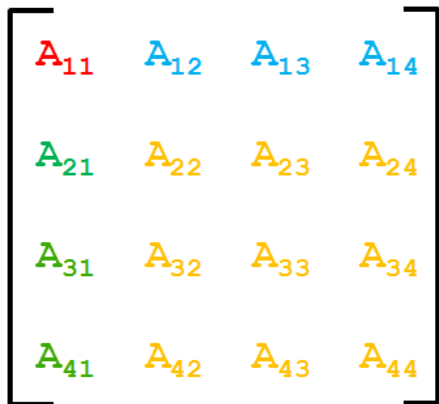
This problem can get away with 16 bit floating point
 => Some architectures are now implementing this

Examples

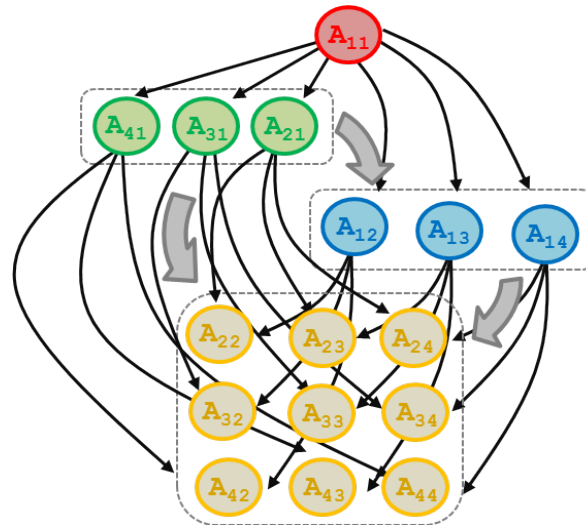
Need of **Batched** routines for **Numerical LA**

[e.g., sparse direct multifrontal methods, preconditioners for sparse iterative methods, tiled algorithms in dense linear algebra, etc.;]
 [collaboration with Tim Davis at al., Texas A&M University]

Sparse / Dense Matrix System



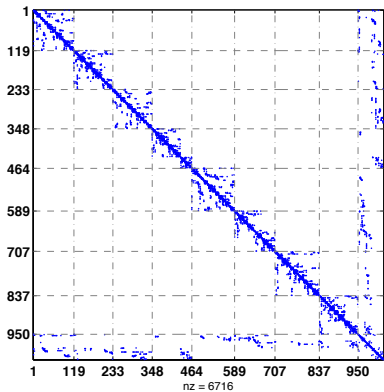
DAG-based factorization



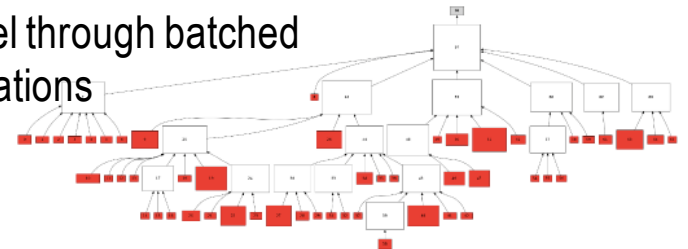
To capture main LA patterns needed in a **numerical library for Batched LA**



- **LU, QR, or Cholesky** on small diagonal matrices
- **TRSMs, QRs, or LUs**
- **TRSMs, TRMMs**
- **Updates (Schur complement) GEMMs, SYRKs, TRMMs**



- Example matrix from Quantum chromodynamics
- Reordered and ready for sparse direct multifrontal solver
- Diagonal blocks can be handled in parallel through batched LU, QR, or Cholesky factorizations

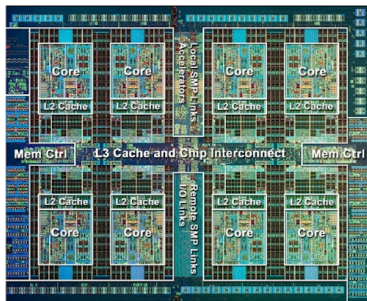


MAGMA Batched Computations CPU

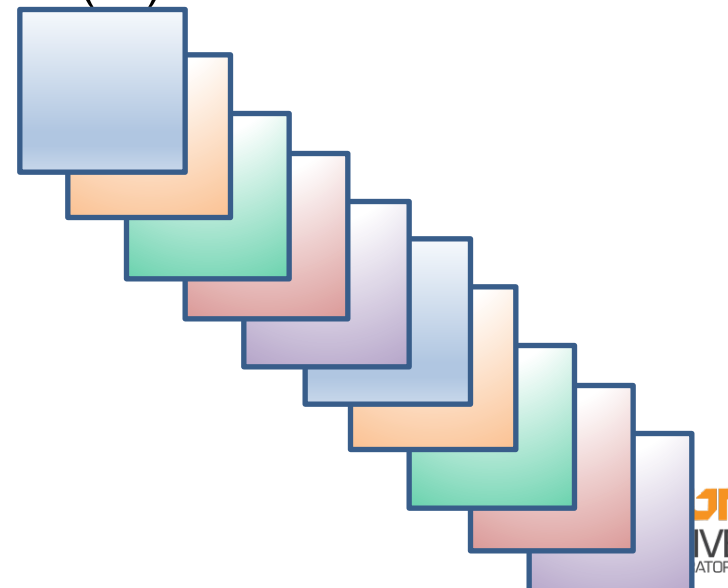
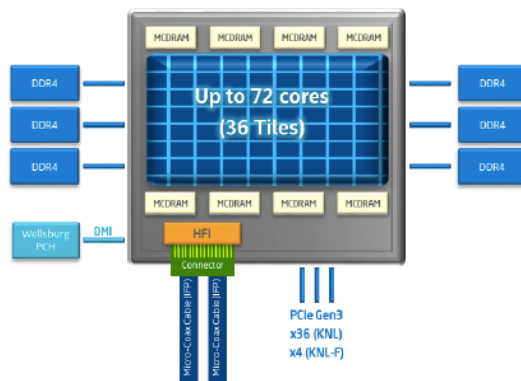
1. Non-batched computation

loop over the matrices one by one and compute either:

- One call for each matrix.
- Sequentially wasting all the other cores, and attaining very poor performance
- Or using multithread (note that for small matrices there is not enough work for all cores so expect low efficiency as well as threads contention can affect the performance)



```
for (i=0; i<batchout; i++)  
  dgemm(...)
```

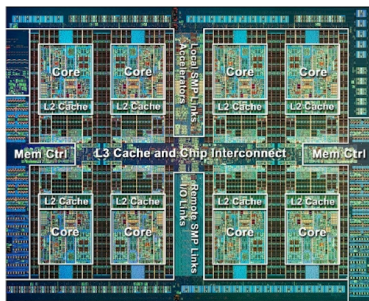


MAGMA Batched Computations CPU

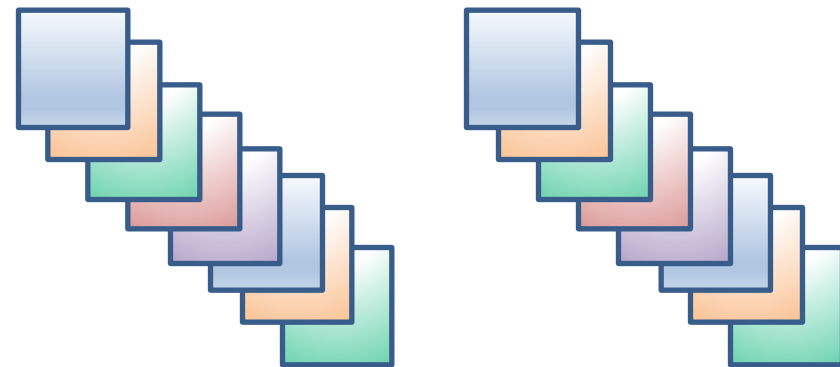
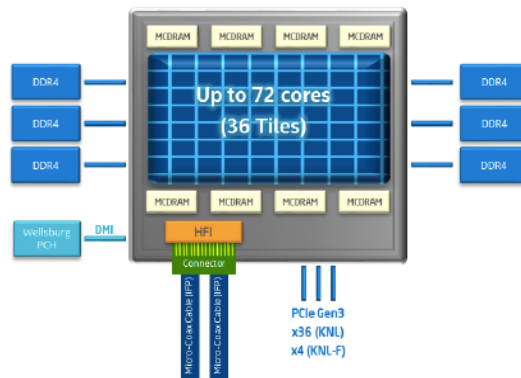
2. Batched computation

loop over the matrices and assign a matrix to each core working on it sequentially and independently

- Since matrices are very small, all the n_cores matrices will fit into L2 cache thus we do not increase L2 cache misses while performing in parallel n_cores computations reaching the best of each core



```
for (i=cpu_id; i<batchcount; i+=n_cpu)  
  batched_dgemm(...)
```

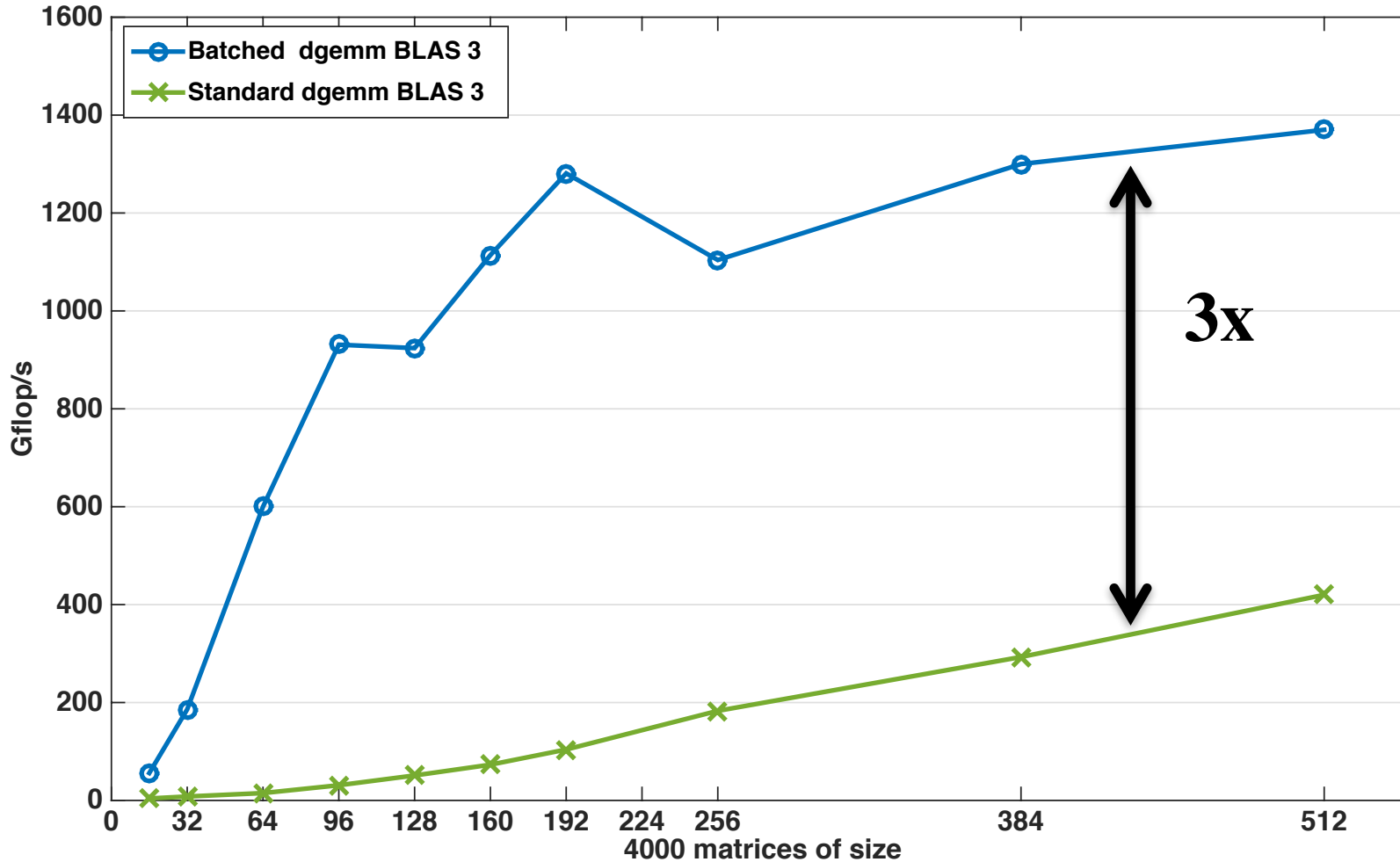




Level 1, 2 and 3 BLAS



64 cores Intel Xeon Phi KNL, 1.3 GHz, Peak DP = 2662 Gflop/s



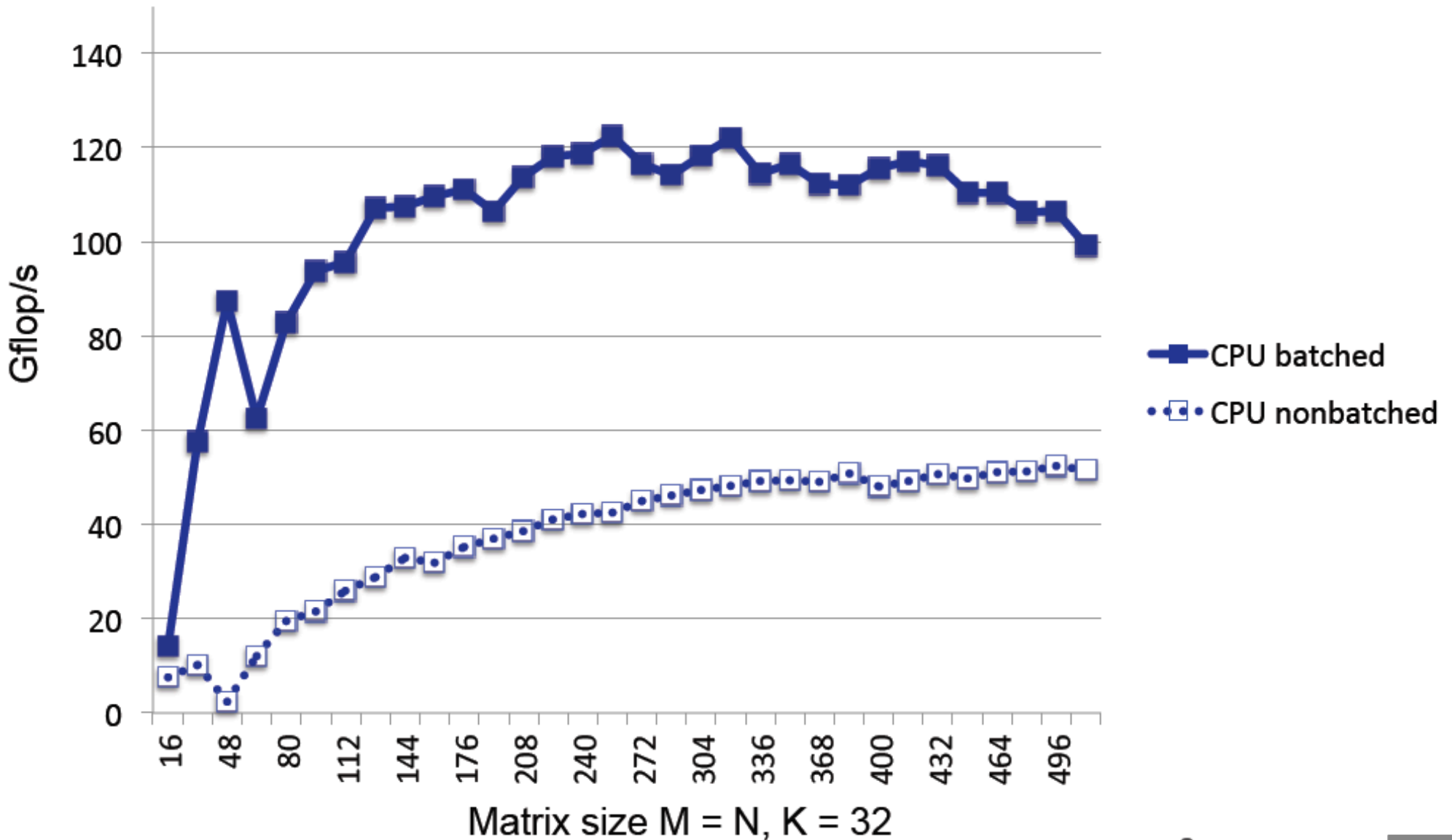
$$C = C + A * B$$

3x

64 cores Intel Xeon Phi KNL, 1.3 GHz
The theoretical peak double precision is 2662 Gflop/s
Compiled with icc and using Intel MKL 2017b1 20160506

Batched Level 3 BLAS DGEMM Example

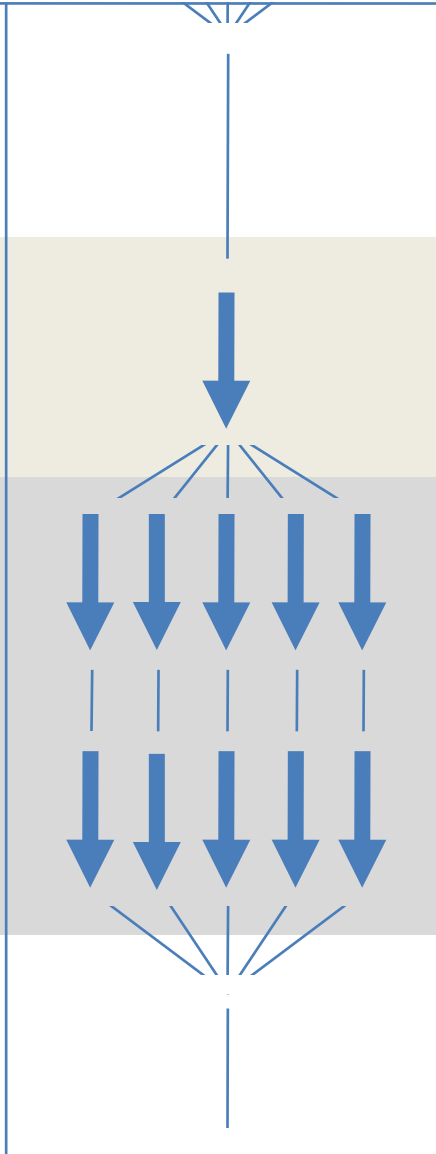
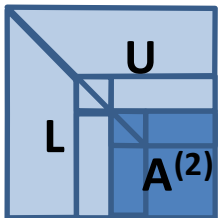
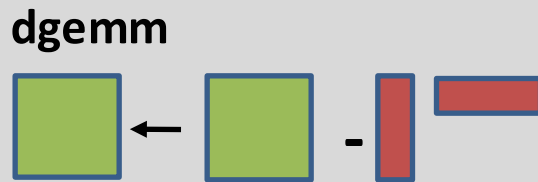
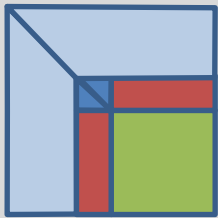
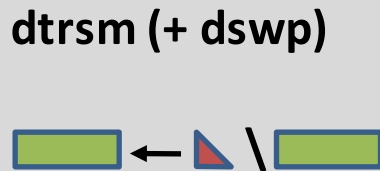
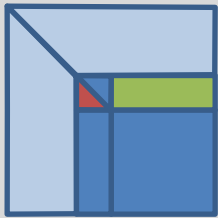
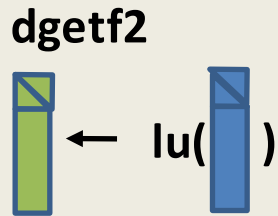
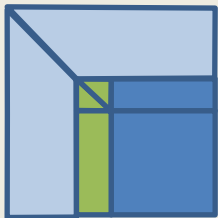
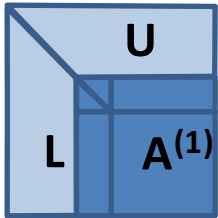
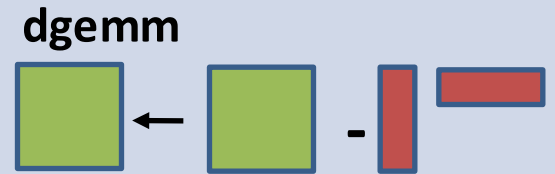
DGEMM (NN), batch_count = 500, 16-core Intel Xeon E5-2670 CPU



Parallelization of LU and QR.

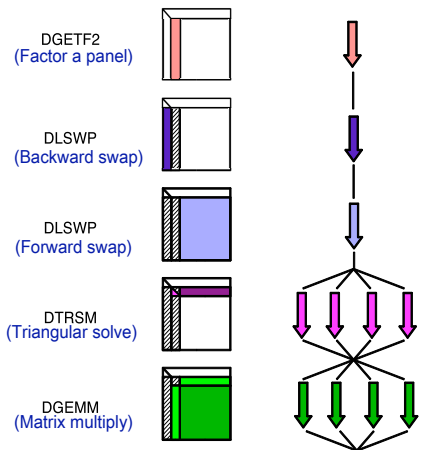
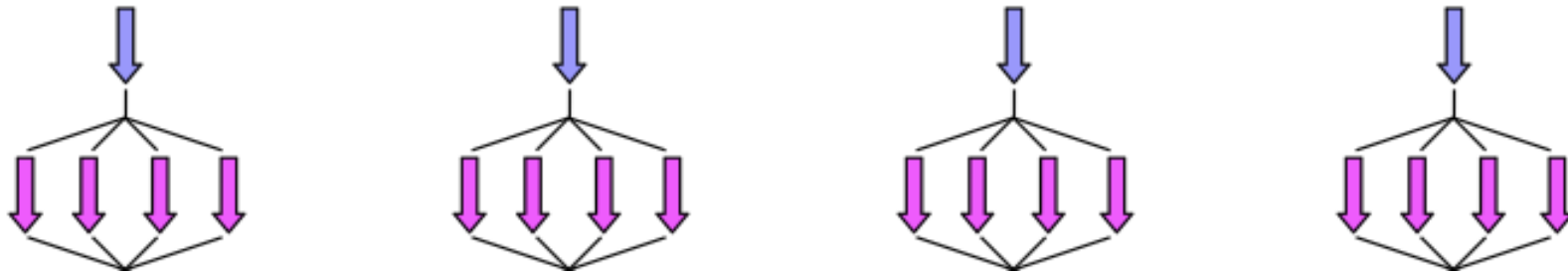
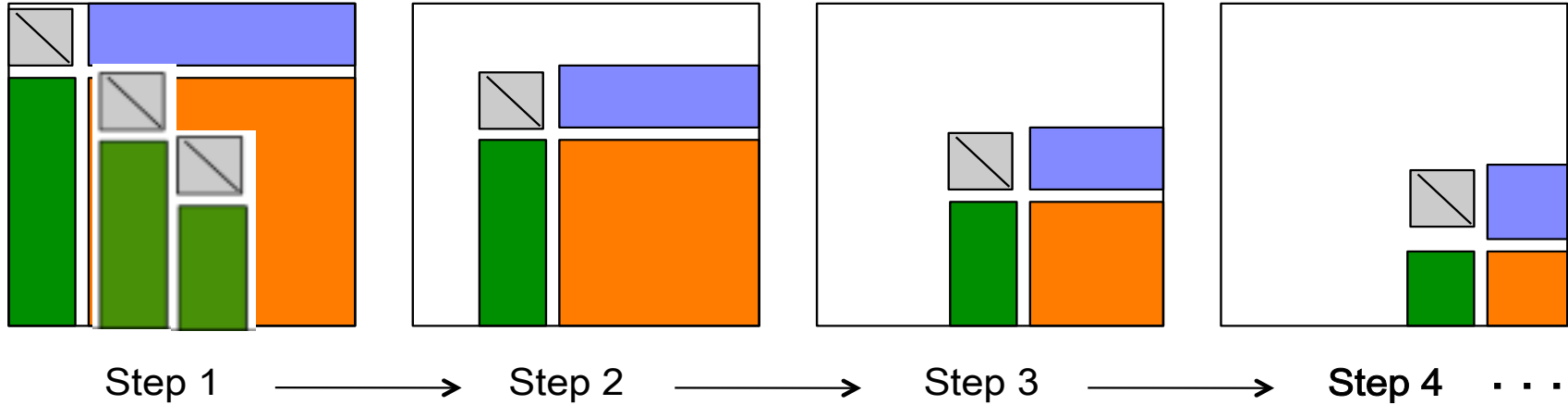
Parallelize the update:

- 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

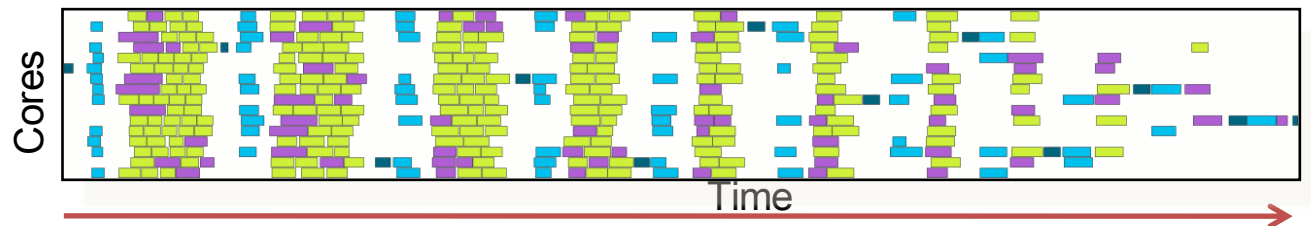


Fork - Join parallelism
Bulk Sync Processing

Synchronization (in LAPACK LU)



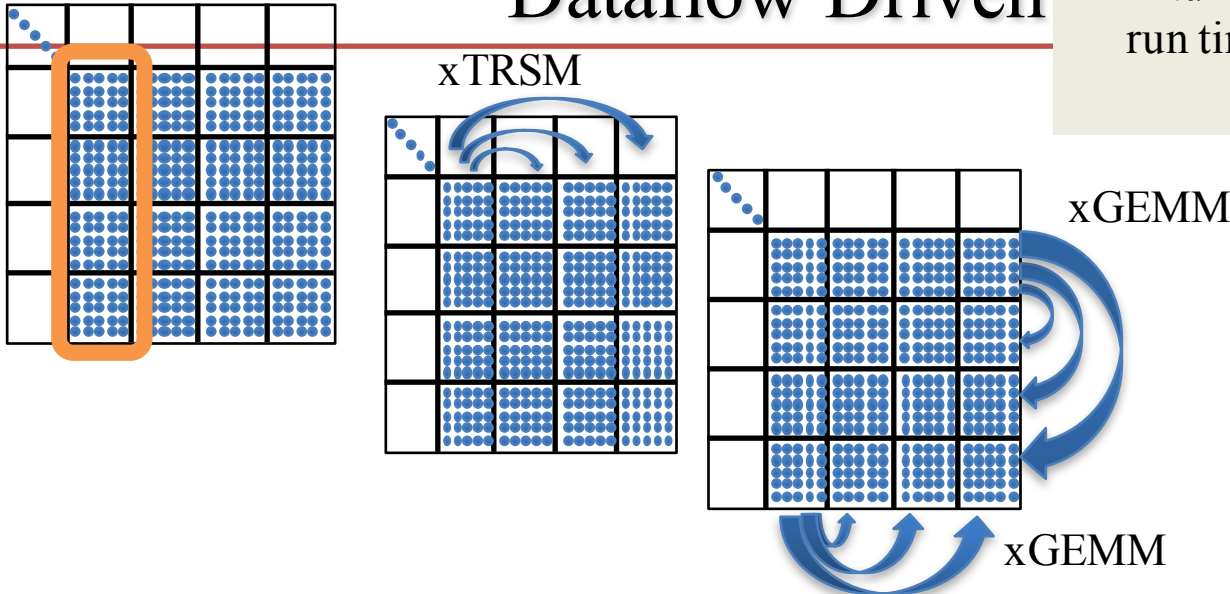
➤ fork join
➤ bulk synchronous processing



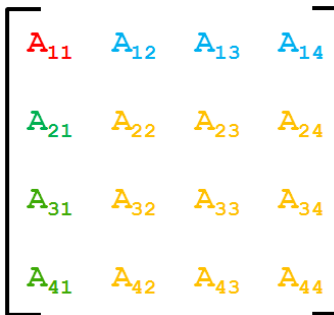
PLASMA LU Factorization

Dataflow Driven

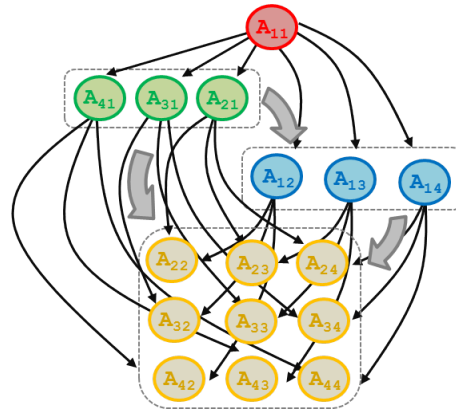
Numerical program generates tasks and run time system executes tasks respecting data dependences.



Sparse / Dense Matrix System



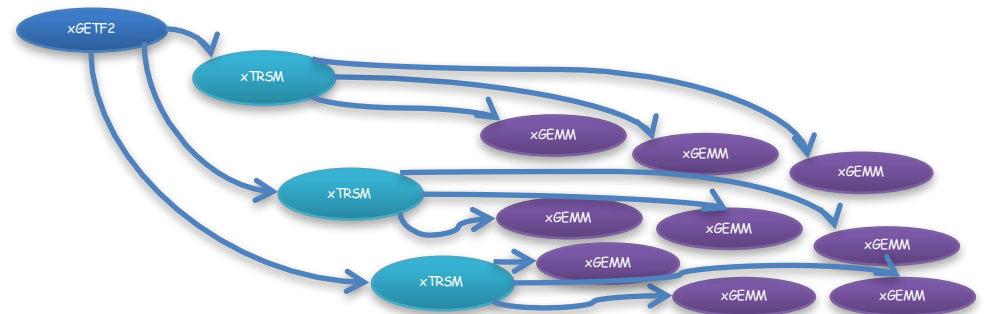
DAG-based factorization



Batched LA

- ➔ • LU, QR, or Cholesky on small diagonal matrices
- ➔ • TRSMs, QRs, or LUs
- ➔ • TRSMs, TRMMs
- ➔ • Updates (Schur complement) GEMMs, SYRKs, TRMMs

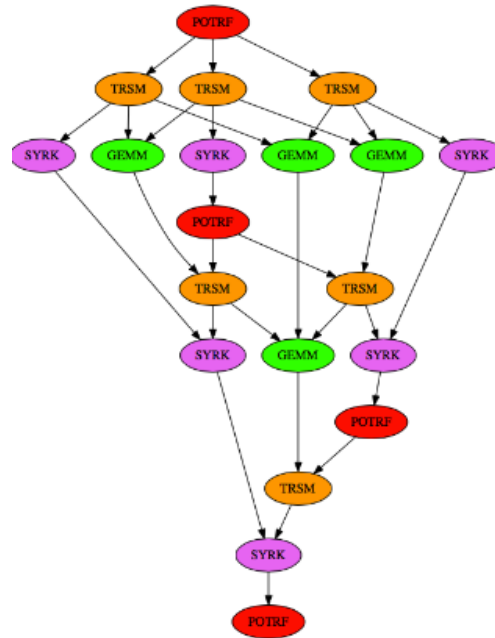
And many other BLAS/LAPACK, e.g., for application specific solvers, preconditioners, and matrices



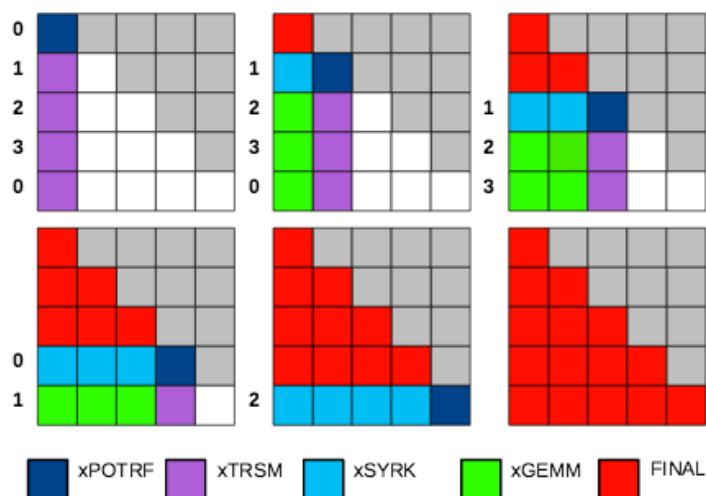
OpenMP tasking

- Added with OpenMP 3.0 (2009)
- Allows parallelization of irregular problems
- OpenMP 4.0 (2013) - Tasks can have dependencies

- **DAGs**



Tiled Cholesky Decomposition



```

#pragma omp parallel
#pragma omp master
{ CHOLESKY( A ); }
CHOLESKY( A ) {
  for (k = 0; k < M; k++) {
    #pragma omp task depend(inout:A(k,k)[0:tilesizel
    { POTRF( A(k,k) ); }
    for (m = k+1; m < M; m++) {
      #pragma omp task \
        depend(in:A(k,k)[0:tilesizel) \
        depend(inout:A(m,k)[0:tilesizel)
      { TRSM( A(k,k), A(m,k) ); }
    }
    for (m = k+1; m < M; m++) {
      #pragma omp task \
        depend(in:A(m,k)[0:tilesizel) \
        depend(inout:A(m,m)[0:tilesizel)
      { SYRK( A(m,k), A(m,m) ); }
      for (n = k+1; n < m; n++) {
        #pragma omp task \
          depend(in:A(m,k)[0:tilesizel, \
            A(n,k)[0:tilesizel) \
          depend(inout:A(m,n)[0:tilesizel)
        { GEMM( A(m,k), A(n,k), A(m,n) ); }
      }
    }
  }
}

```

Dataflow Based Design

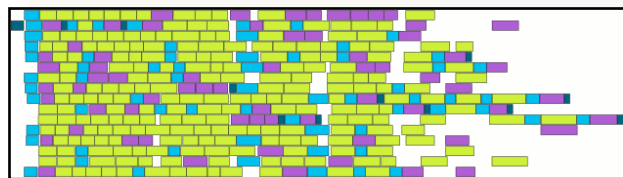
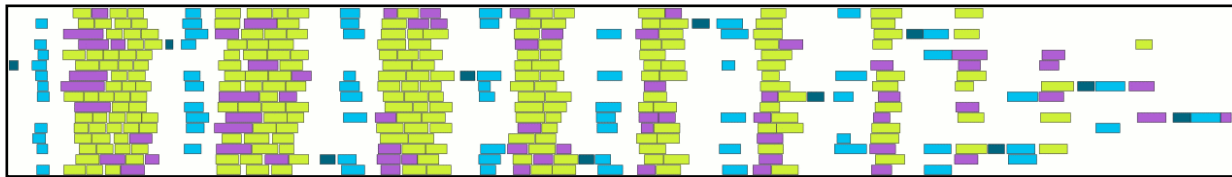
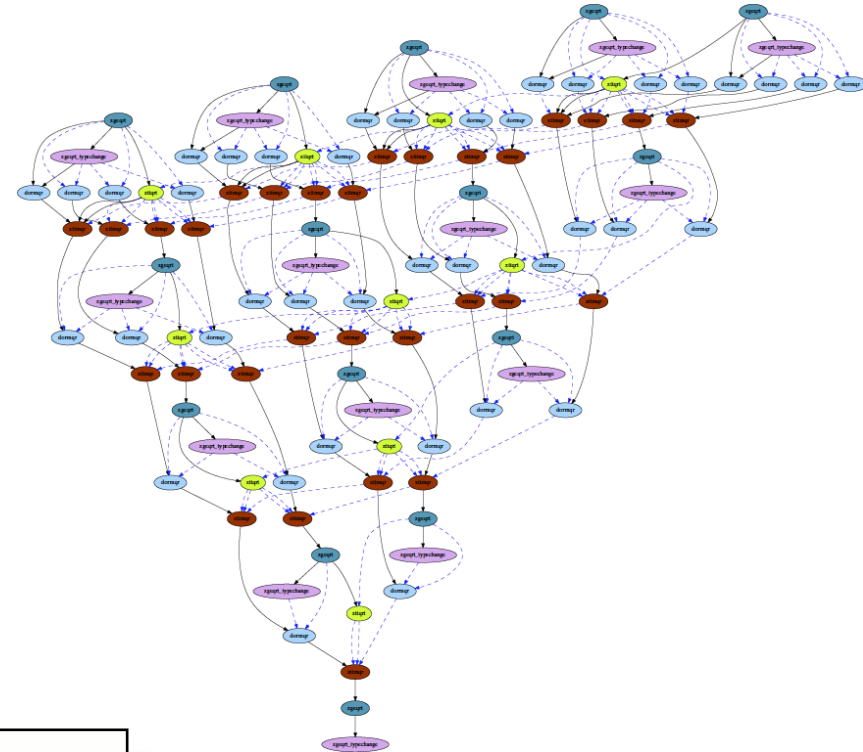
Objectives

- 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.

Time

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 .**

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

$$\boxed{x_{i+1} - x_i} = -\frac{f(x_i)}{f'(x_i)} \quad 40$$

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.

$L U = \text{lu}(A)$	$O(n^3)$
$x = L \setminus (U \setminus b)$	$O(n^2)$
$r = b - Ax$	$O(n^2)$
WHILE $\ r \ $ not small enough	
$z = L \setminus (U \setminus r)$	$O(n^2)$
$x = x + z$	$O(n^1)$
$r = b - Ax$	$O(n^2)$
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.

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

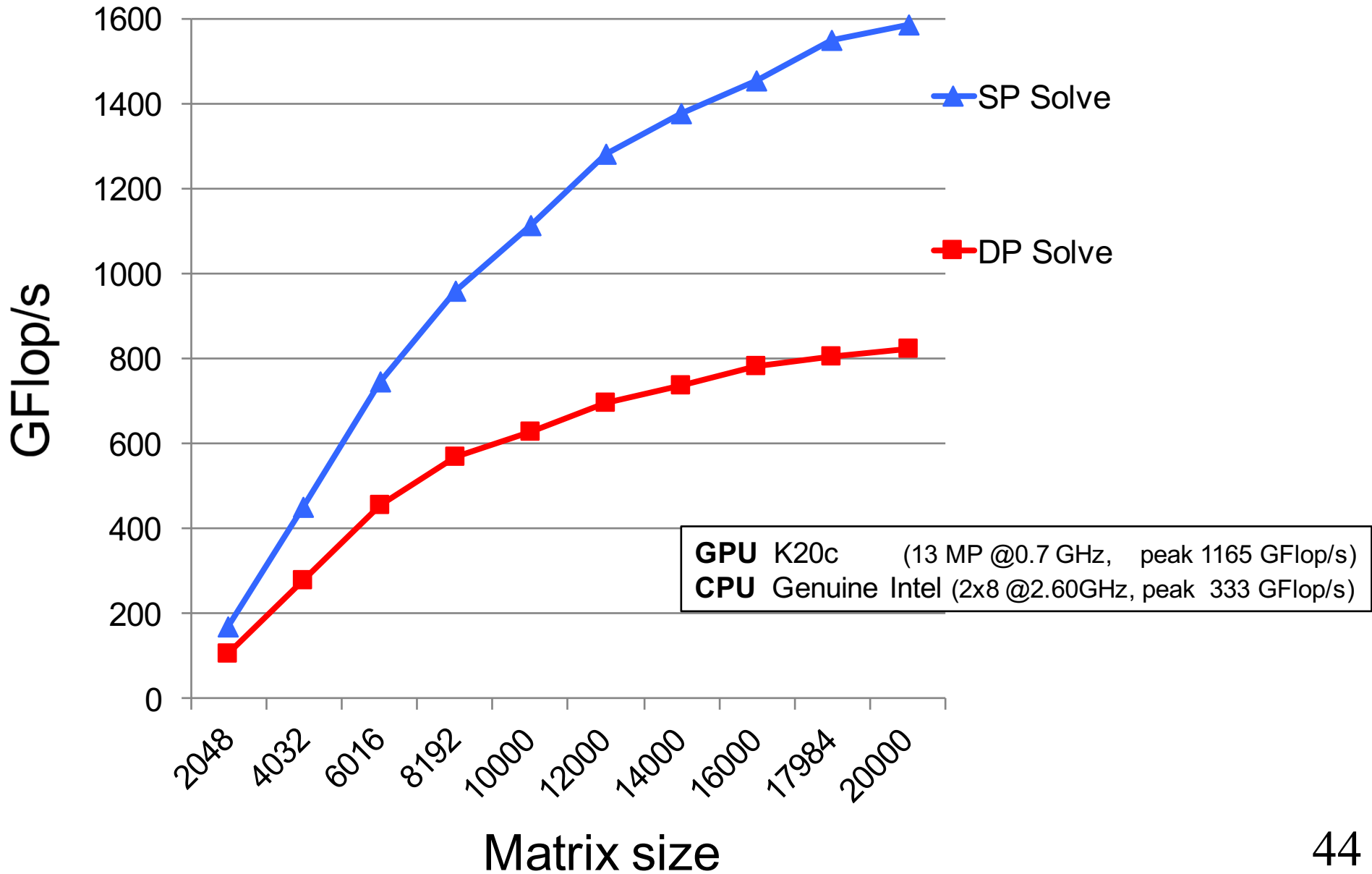
- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

- 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; $O(10^8)$



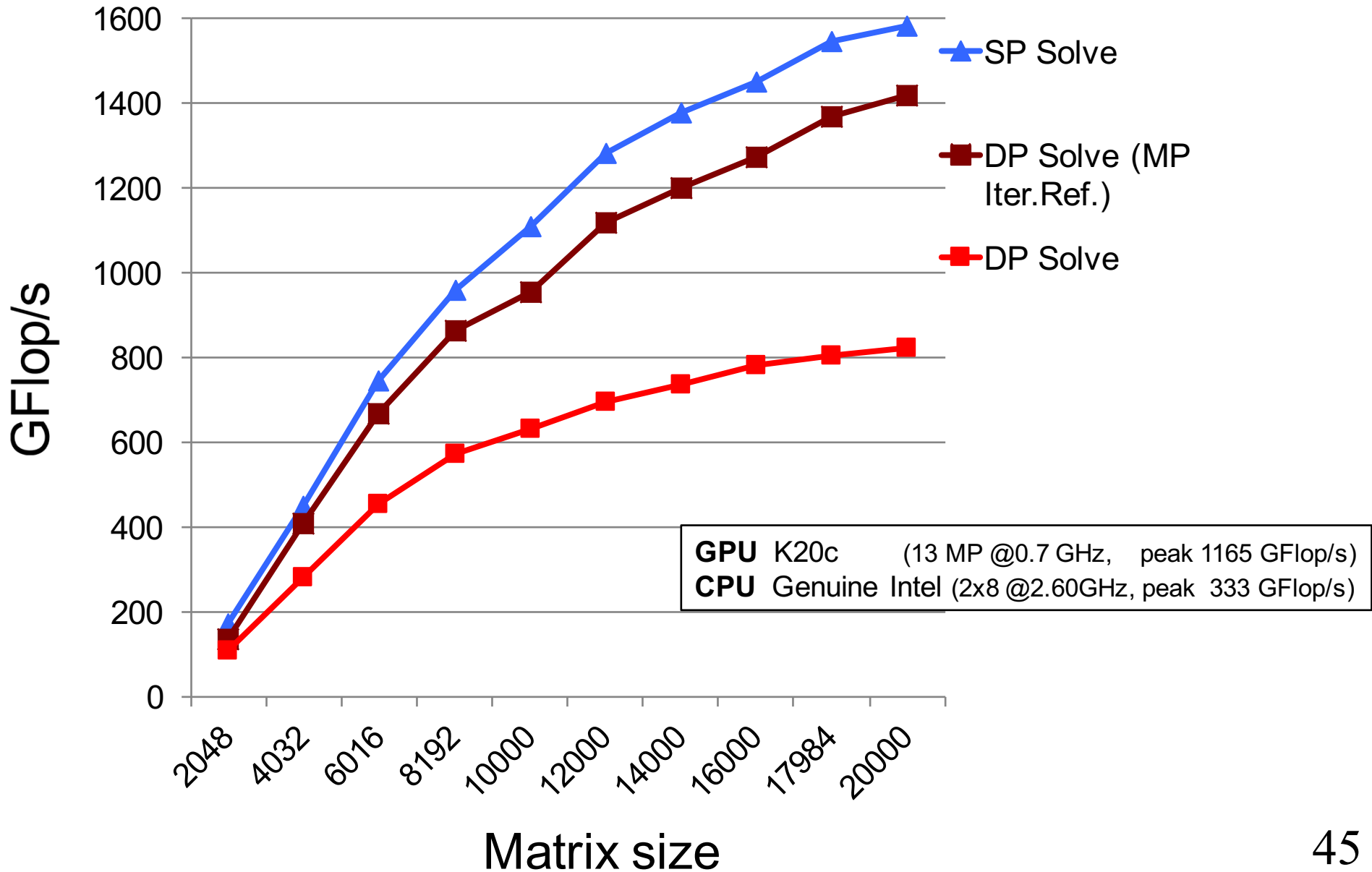
Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement



Avoiding Synchronization

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



Introduction

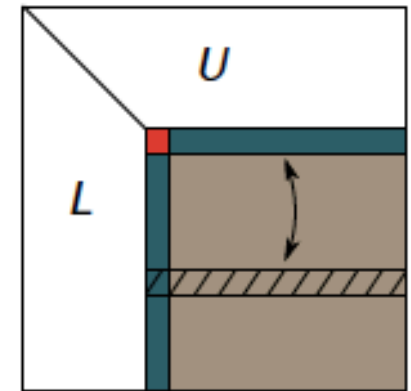
LU decomposition (Gaussian Elimination) for the solution of $Ax = b$

for $k = 1$ **to** n **do**

$$a_{k+1:n,k} \leftarrow \frac{a_{k+1:n,k}}{a_{kk}}$$

$$a_{k+1:n,k+1:n} \leftarrow a_{k+1:n,k+1:n} - a_{k+1:n,k} \times a_{k,k+1:n}$$

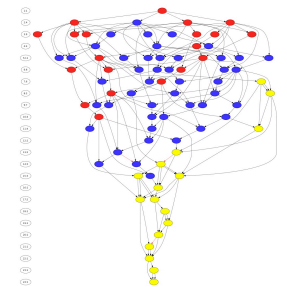
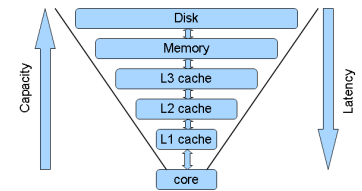
end for



- Stability issue: a_{kk} may be small or zero \Rightarrow large element growth \Rightarrow elements of normal size lost in summation.
- **Partial pivoting (GEPP)**: swap rows so that each a_{kk} is large.
row k is exchanged with row p such that $|a_{pk}| = \max_{j \geq k} |a_{jk}|$
Eventually, $PA = LU$ (P permutation matrix).

Software and Algorithm Must Keep Pace with the Changes in Hardware

- Classical analysis of algorithms may not be valid,
 - # of floating point ops \neq 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
- Many existing algorithms can't fully exploit the features of modern architecture
- Time to rethink



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**
 - With 10M cores things will fail.
 - **Power**
 - 50 Gflops/w (today at 6 Gflops/w)
- **We will need completely new approaches and technologies to reach the Exascale level**



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.

Collaborators and Support

MAGMA team

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

PLASMA team

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

Collaborating partners

University of Tennessee, Knoxville

Lawrence Livermore National Laboratory, Livermore, CA

University of California, Berkeley

University of Colorado, Denver

INRIA, France (StarPU team)

KAUST, Saudi Arabia



U.S. DEPARTMENT OF
ENERGY



Umeå
University



INRIA



Science & Technology
Facilities Council

Rutherford Appleton
Laboratory

MANCHESTER
1824

The University of Manchester

University of
Manchester