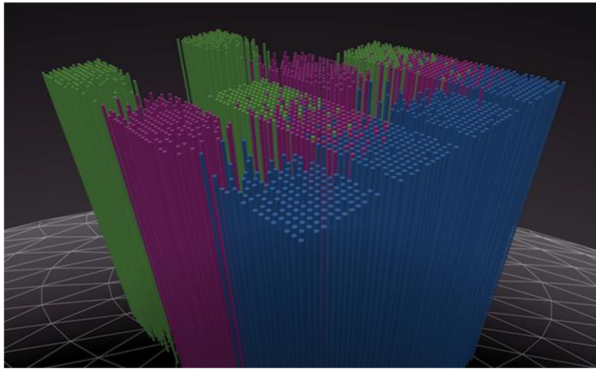
17th Workshop on High Performance Computing in Meteorology

24-28 October 2016



An Overview of HPC and the Changing Rules at Exascale

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
  - Using linear algebra algorithms and software as the example

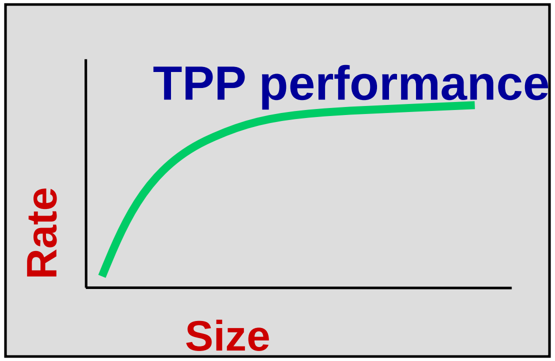
State of Supercomputing Today

- 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, KNC) (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

$Ax=b$, *dense problem*



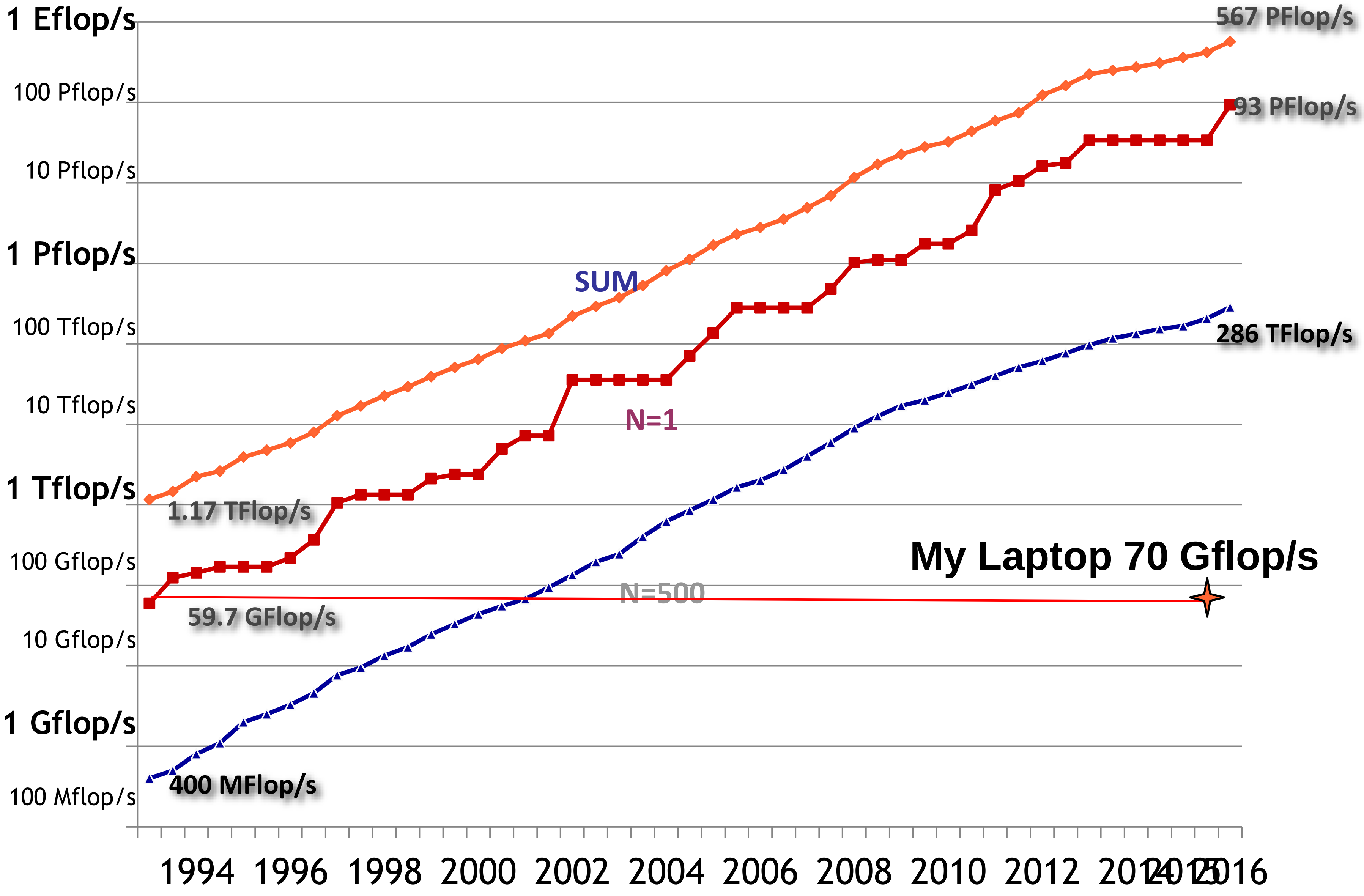
- Updated twice a year

  SC'xy in the States in November

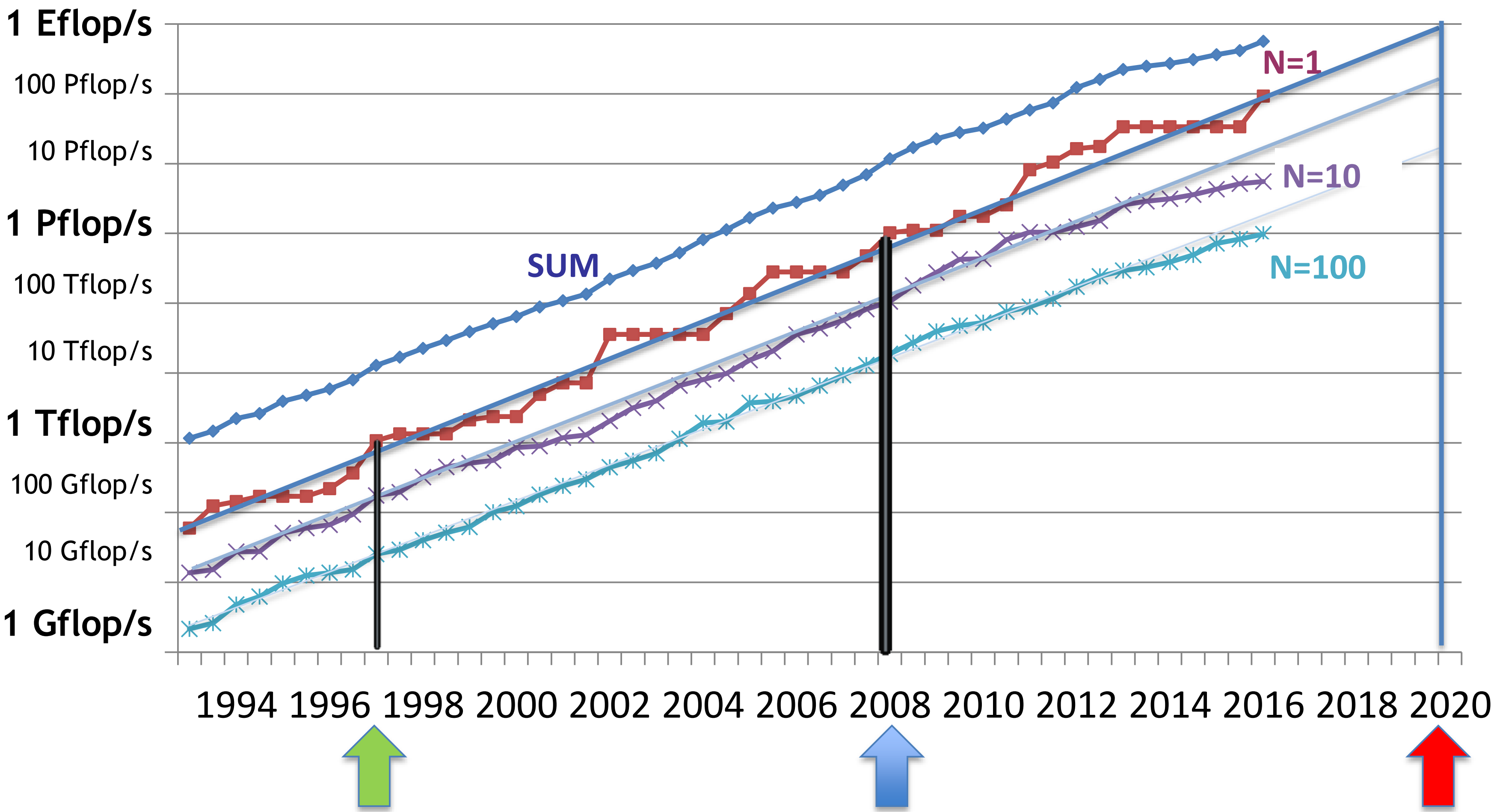
  Meeting in Germany in June

- All data available from **www.top500.org**

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



PERFORMANCE DEVELOPMENT



Tflops
Achieved

Pflops
Achieved

Eflops
Achieved?

June 2016: The TOP 10 Systems

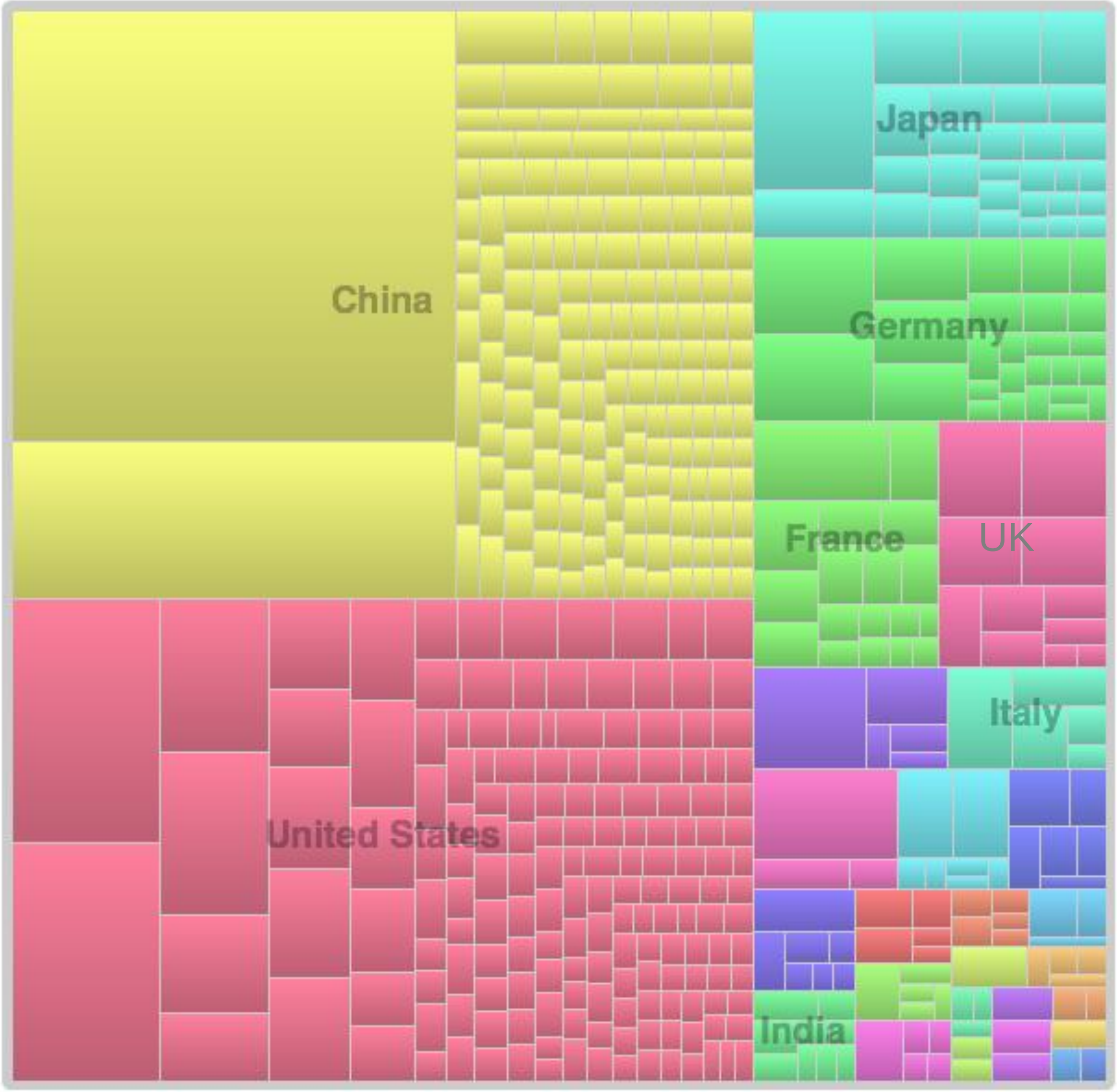
Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	GFlops/ Watt
1	National Super Computer Center in Wuxi	Sunway TaihuLight, SW26010 (260C) + Custom	China	10,649,000	93.0	74	15.4	6.04
2	National Super Computer Center in Guangzhou	Tianhe-2 NUDT, Xeon (12C) + IntelXeon Phi (57c) + Custom	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
					7.2	85	7.89	2.18
							12.7	.827
						85	3.95	2.07
7	Los Alamos & Sandia	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) + Nnvidia	China	5440	.286	71		

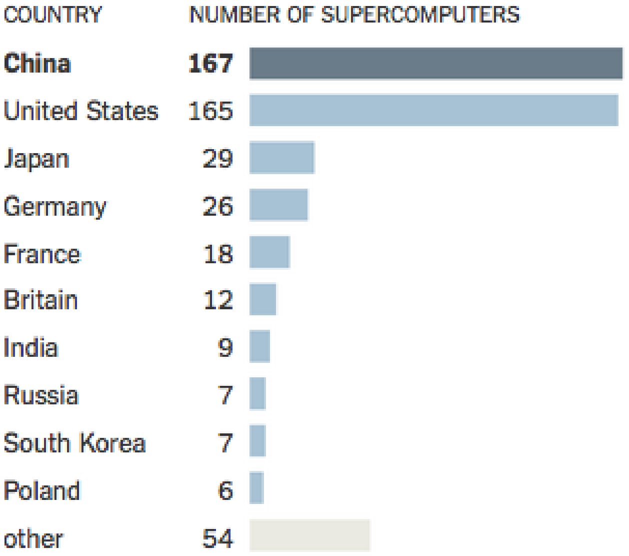
TaihuLight ~ 5 X Performance of Titan

TaihuLight ~ 1.4 X Sum of all DOE Systems

TaihuLight ~ .9 X Sum of all EU Systems

Countries Share

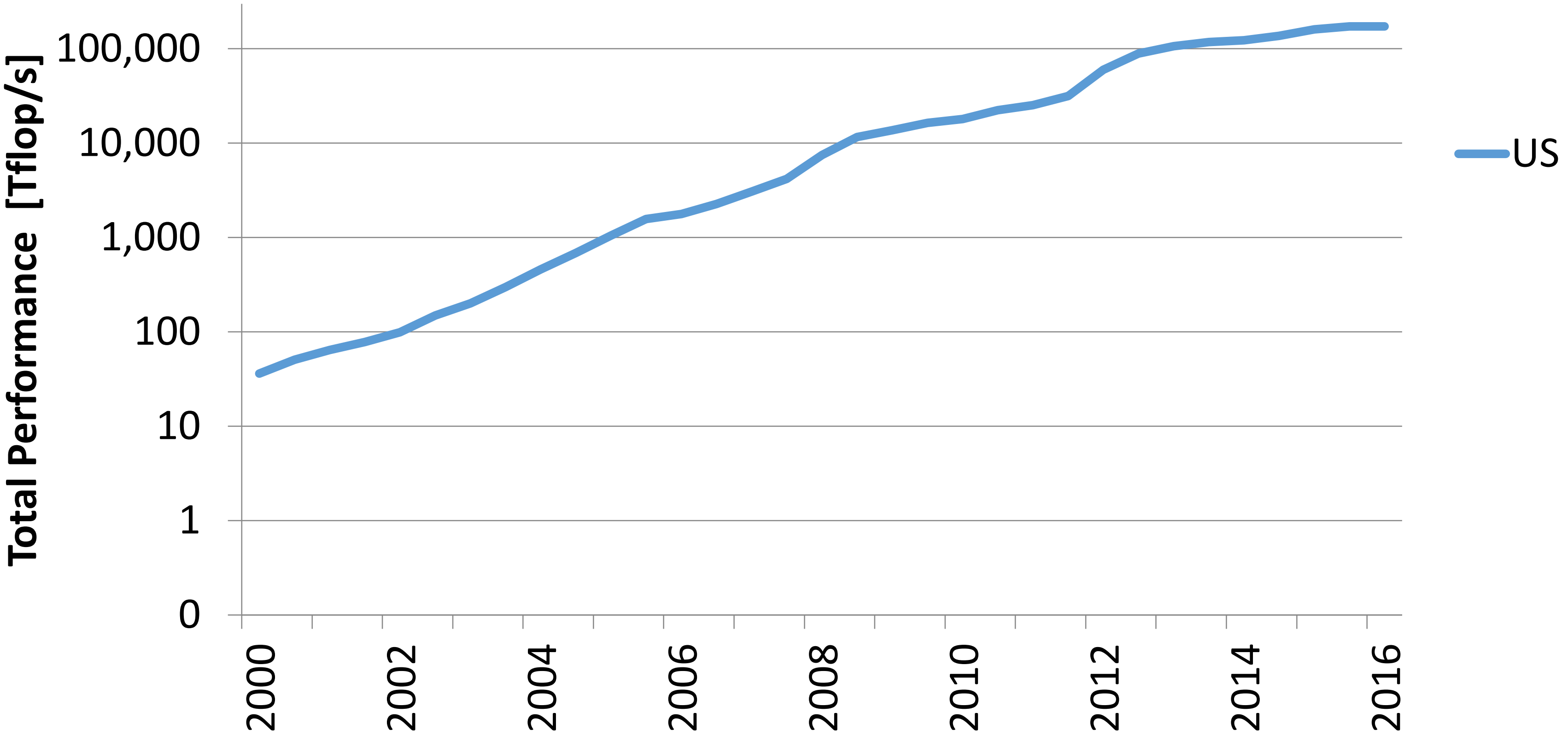




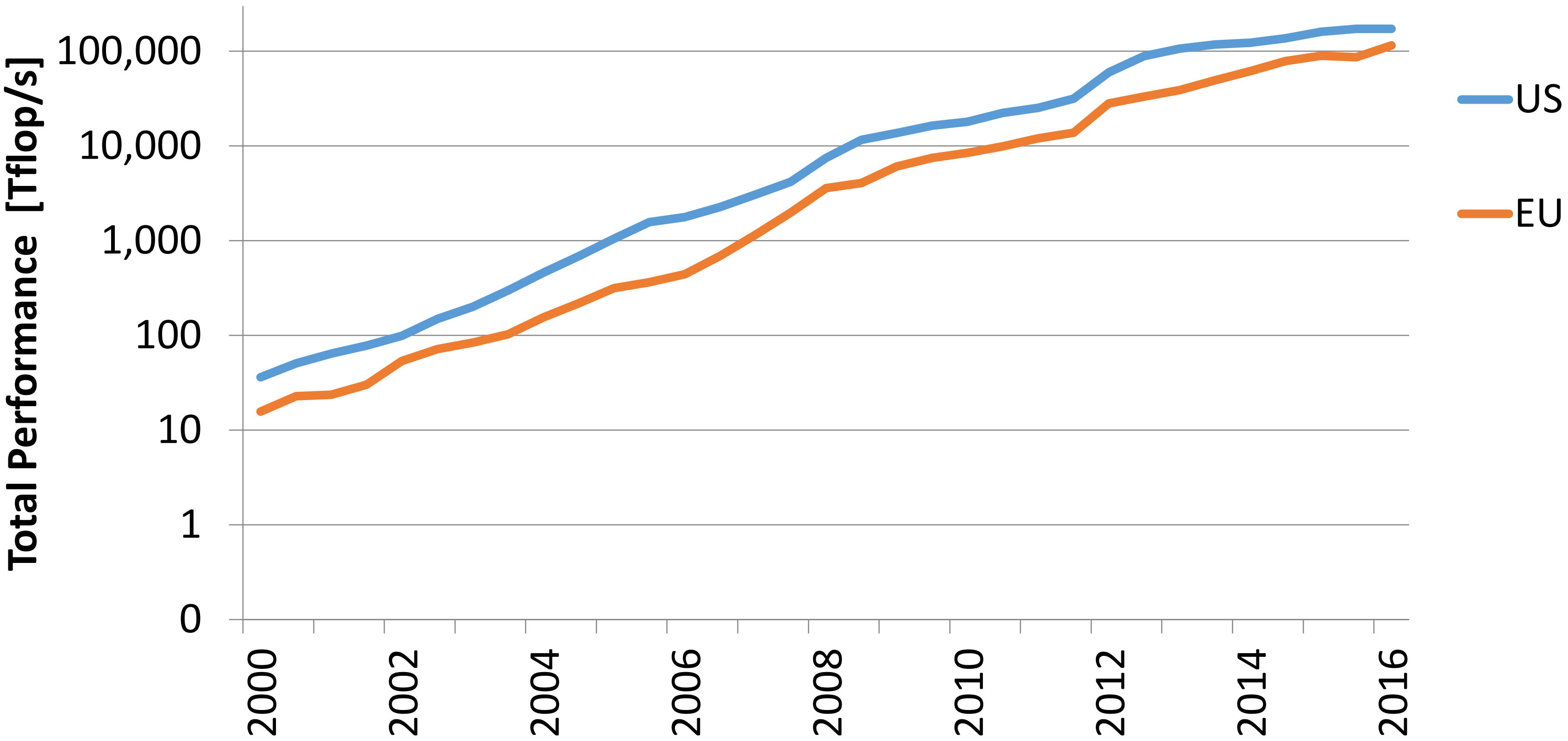
COUNTRY	NUMBER OF SUPERCOMPUTERS
China	**167**
United States	165
Japan	29
Germany	26
France	18
Britain	12
India	9
Russia	7
South Korea	7
Poland	6
other	54

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.

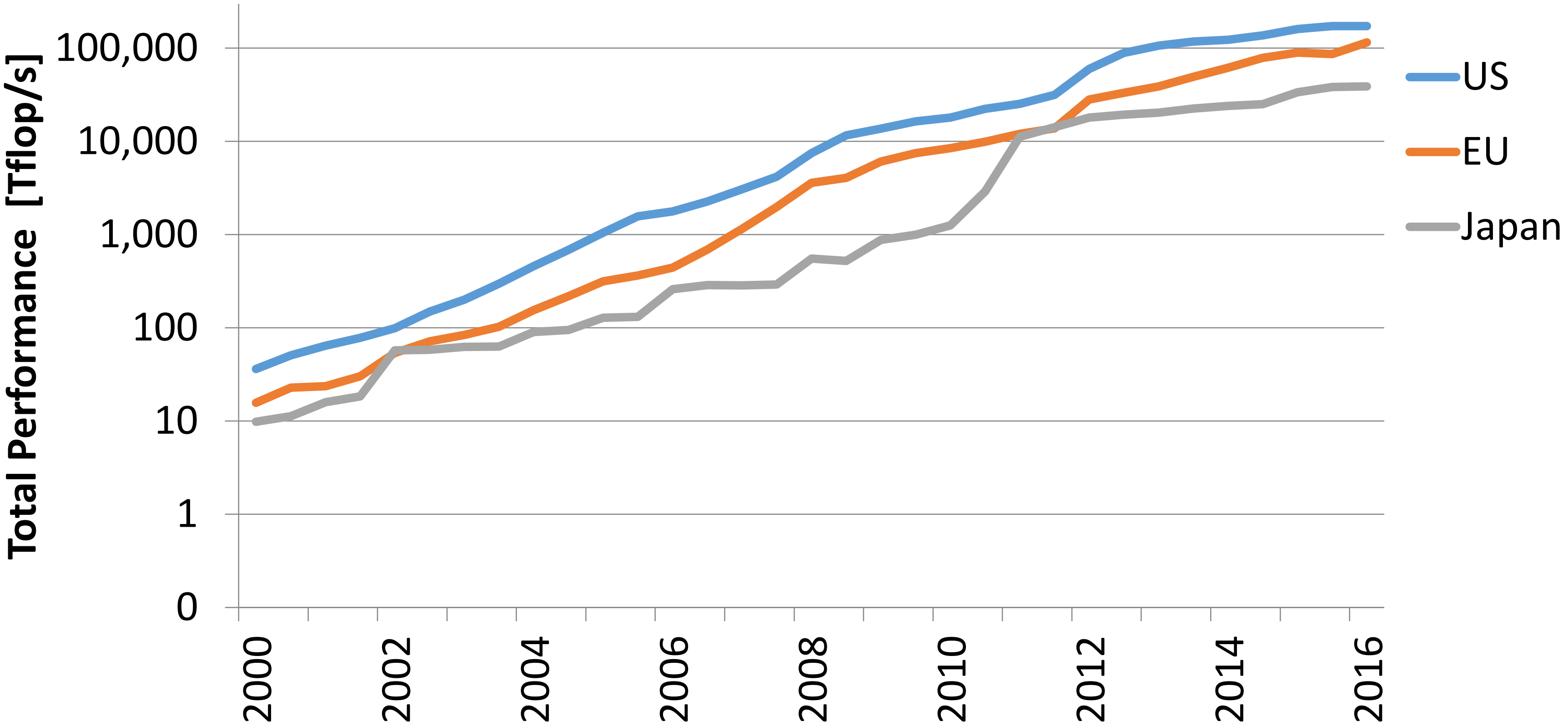
Performance of Countries



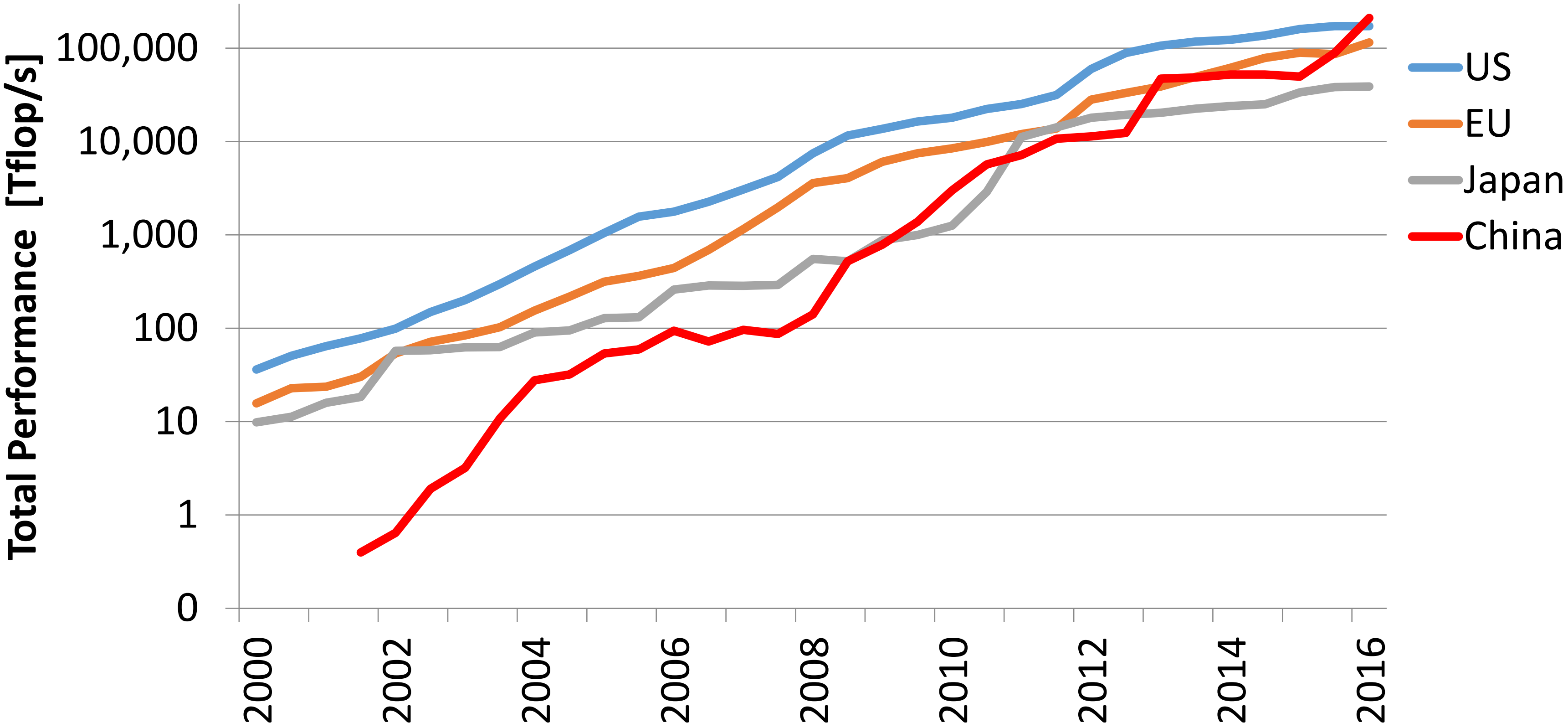
Performance of Countries



Performance of Countries



Performance of Countries



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, Nvidia, Mellanox based systems
- Argonne Lab to receive Intel and Cray based system
  - *After this the next round of systems are an Exaflop*
- US Dept of Commerce is a groups from receiving In
  - cle th the
  - 5.

  - ıngzl
  - njin,
  - National University for Def
  - National SC Center Changsl

Since the Dept of Commerce Action …

- Expanded focus on Chinese made HW and SW
  - "Anything but from the US"
- Three separate developments in HPC
  - Wuxi
    - ShenWei O(100) Pflops all Chinese, June 2016
  - NUDT
    - Tianhe-2A O(100) Pflops will be Chinese ARM + accelerator, 2017
  - Sugon - CAS ICT
    - AMD? new processors
- In the latest "5 Year Plan"
  - Govt push to build out a domestic HPC ecosystem.
  - Exascale system, will not use any US chips

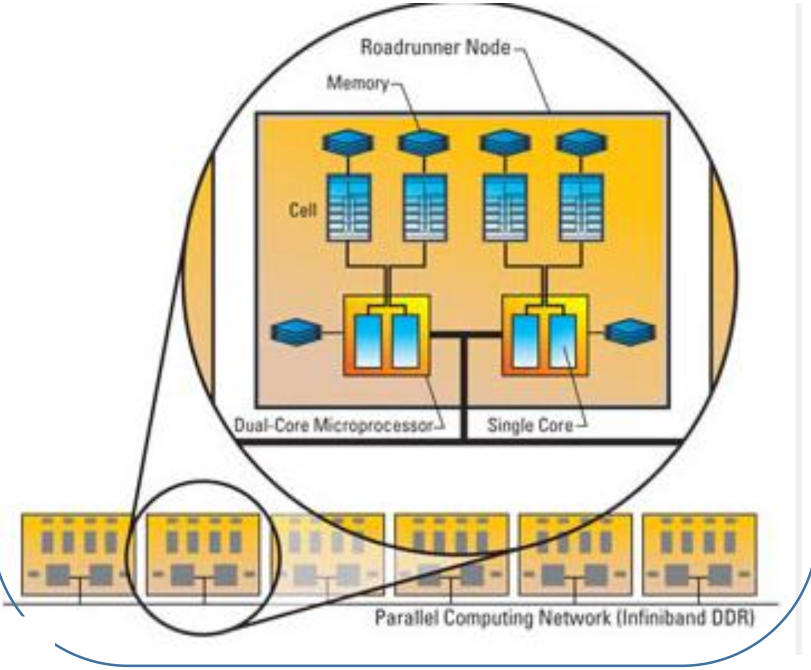
SW26010 Processor

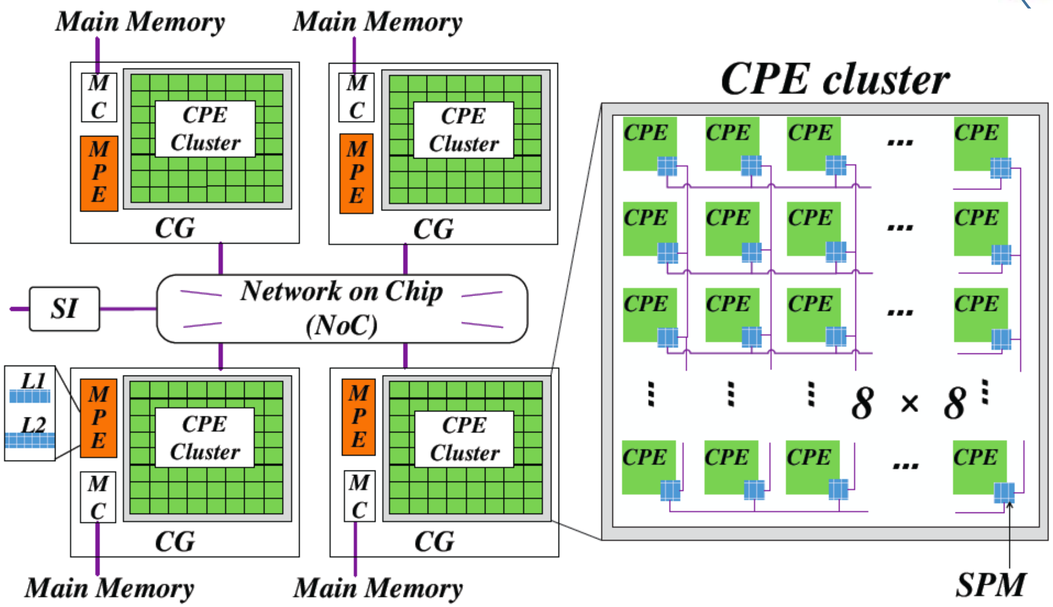
- China's first homegrown many-core processor
  - Vendor: Shanghai High Performance IC Design Center
  - Supported by National Science and Technology Major Project (NMP): Core Electronic Devices, High-end Generic Chips, and Basic Software
  - 28 nm technology
  - 260 Cores
  - 3 Tflop/s peak

SW26010: General Architecture

- 1 node
- 260 cores per processor
- 4 Core Groups (CGs), each of wh
  - 1 Management Processing Elemen
  - 64 (8x8) Computing Processing El

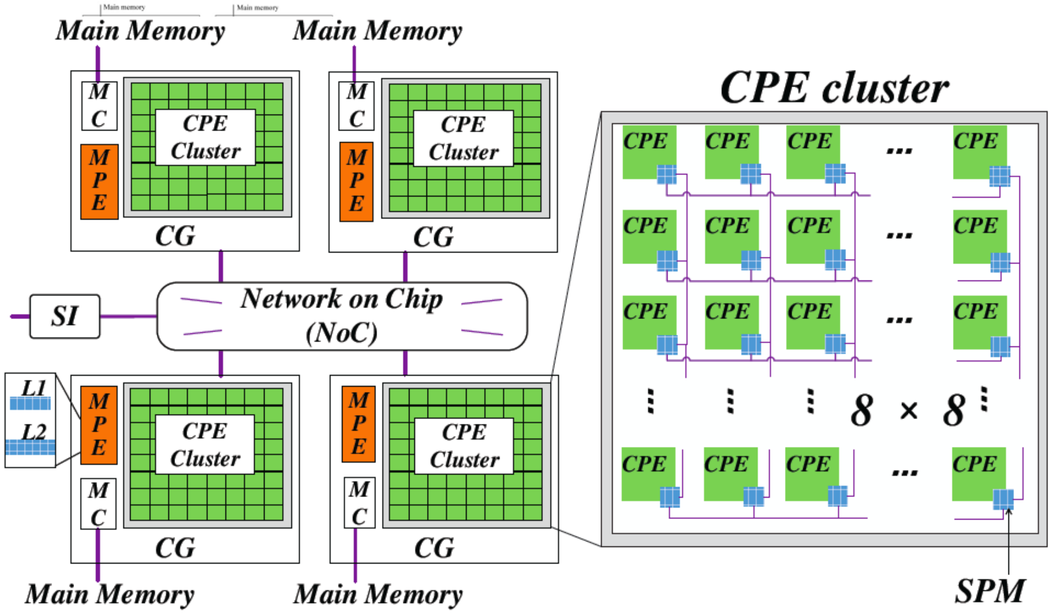
RoadRunner Node

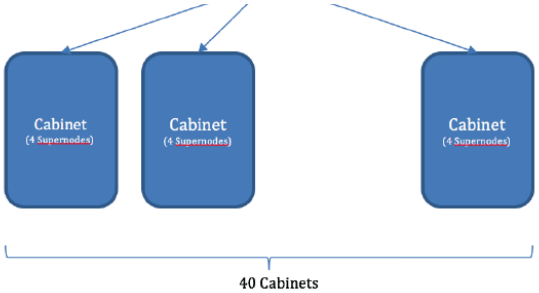




Sunway TaihuLight http://bit.ly/sunway-2016

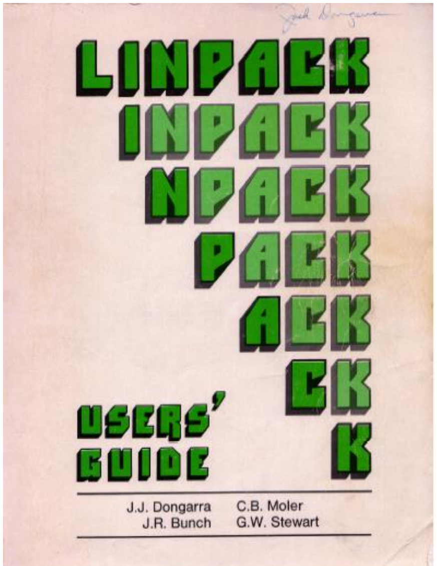
- SW26010 processor
- Chinese design, fab, and ISA
- 1.45 GHz
- Node = 260 Cores (1 socket)
  - 4 – core groups
    - 64 CPE, No cache, 64 KB scratchpad/CPE
    - 1 MPE w/32 KB L1 dcache & 256KB L2 cache
  - 32 GB memory total, 136.5 GB/s
  - ~3 Tflop/s, (22 flops/byte)
- Cabinet = 1024 nodes
  - 4 supernodes=32 boards(4 cards/b(2 no
  - ~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 for HPL, 74% peak
- 15.3 MW, water cooled
- 6.07 Gflop/s per Watt
- 1.8B RMBs ~ $280M, (building, hw, apps, sw, …)
- Planning an air-cooled version, single cabinet for their weather community

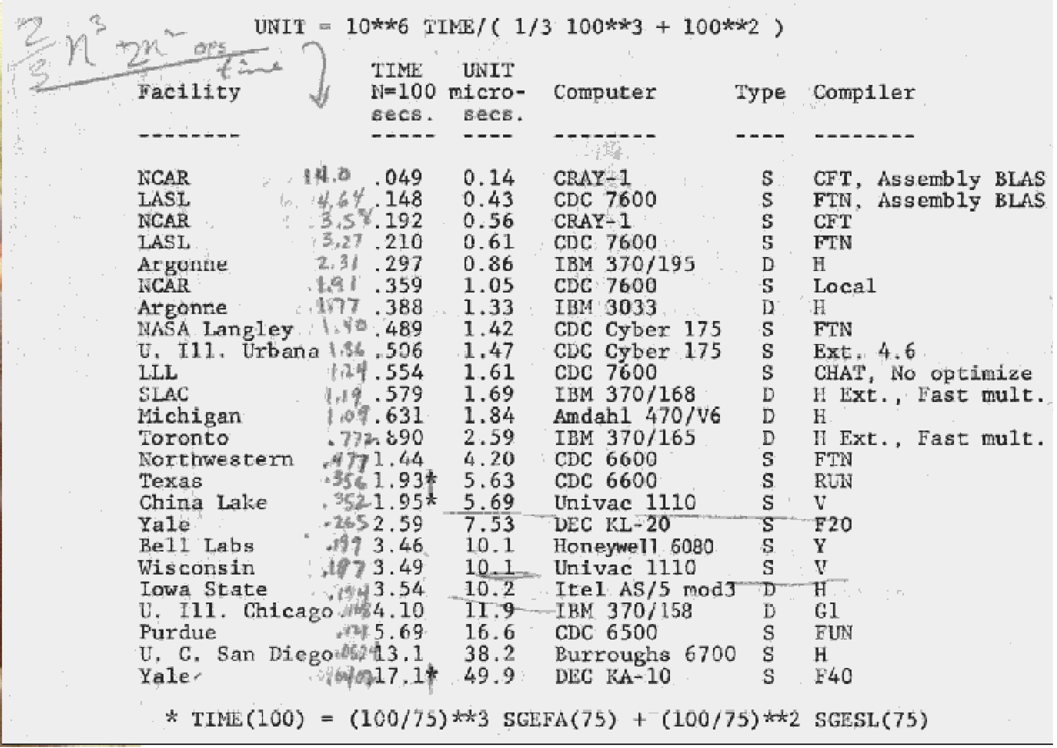




Confessions of an Accidental Benchmarker

- Appendix B of the Linpack Users' Guide
  - Designed to help users extrapolate execution Linpack software package
- First benchmark report from 1977;
  - Cray 1 to DEC PDP-10





$\frac{2}{3} n^3 + 2n^2$ ops / time

UNIT = 10**6 TIME/(1/3 100**3 + 100**2)

Facility		TIME N=100 secs.	UNIT micro-secs.	Computer	Type	Compiler
NCAR	14.0	.049	0.14	CRAY-1	S	CFT, Assembly BLAS
LASL	4.64	.148	0.43	CDC 7600	S	FTN, Assembly BLAS
NCAR	3.58	.192	0.56	CRAY-1	S	CFT
LASL	3.27	.210	0.61	CDC 7600	S	FTN
Argonne	2.31	.297	0.86	IBM 370/195	D	H
NCAR	1.91	.359	1.05	CDC 7600	S	Local
Argonne	1.77	.388	1.33	IBM 3033	D	H
NASA Langley	1.40	.489	1.42	CDC Cyber 175	S	FTN
U. Ill. Urbana	1.36	.506	1.47	CDC Cyber 175	S	Ext. 4.6
LLL	1.24	.554	1.61	CDC 7600	S	CHAT, No optimize
SLAC	1.19	.579	1.69	IBM 370/168	D	H Ext., Fast mult.
Michigan	1.09	.631	1.84	Amdahl 470/V6	D	H
Toronto	.772	.890	2.59	IBM 370/165	D	H Ext., Fast mult.
Northwestern	.477	1.44	4.20	CDC 6600	S	FTN
Texas	.356	1.93*	5.63	CDC 6600	S	RUN
China Lake	.352	1.95*	5.69	Univac 1110	S	V
Yale	.265	2.59	7.53	DEC KL-20	S	F20
Bell Labs	.197	3.46	10.1	Honeywell 6080	S	Y
Wisconsin	.187	3.49	10.1	Univac 1110	S	V
Iowa State	.194	3.54	10.2	Itel AS/5 mod3	D	H
U. Ill. Chicago	[illegible]	4.10	11.9	IBM 370/158	D	G1
Purdue	.121	5.69	16.6	CDC 6500	S	FUN
U. C. San Diego	.0524	13.1	38.2	Burroughs 6700	S	H
Yale	[illegible]	17.1*	49.9	DEC KA-10	S	F40

* TIME(100) = (100/75)**3 SGEFA(75) + (100/75)**2 SGESL(75)

Many Other Benchmarks

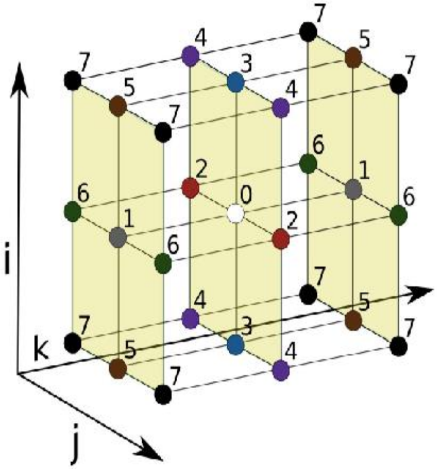
- TOP500
- Green 500
- Graph 500
- Sustained Petascale Performance
- HPC Challenge
- Perfect
- ParkBench
- SPEC-hpc
- Big Data Top100
- Livermore Loops
- EuroBen

- NAS Parallel Benchmarks
- Genesis
- RAPS
- SHOC
- LAMMPS
- Dhrystone
- Whetstone
- I/O Benchmarks
- WRF
- Yellowstone
- Roofline
- Neptune

hpcg-benchmark.org

HPCG

- 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
- Synthetic discretized 3D PDE (FEM, FVM, FDM).
- Sparse matrix:
  - 27 nonzeros/row interior.
  - 8 – 18 on boundary.
  - Symmetric positive definite.



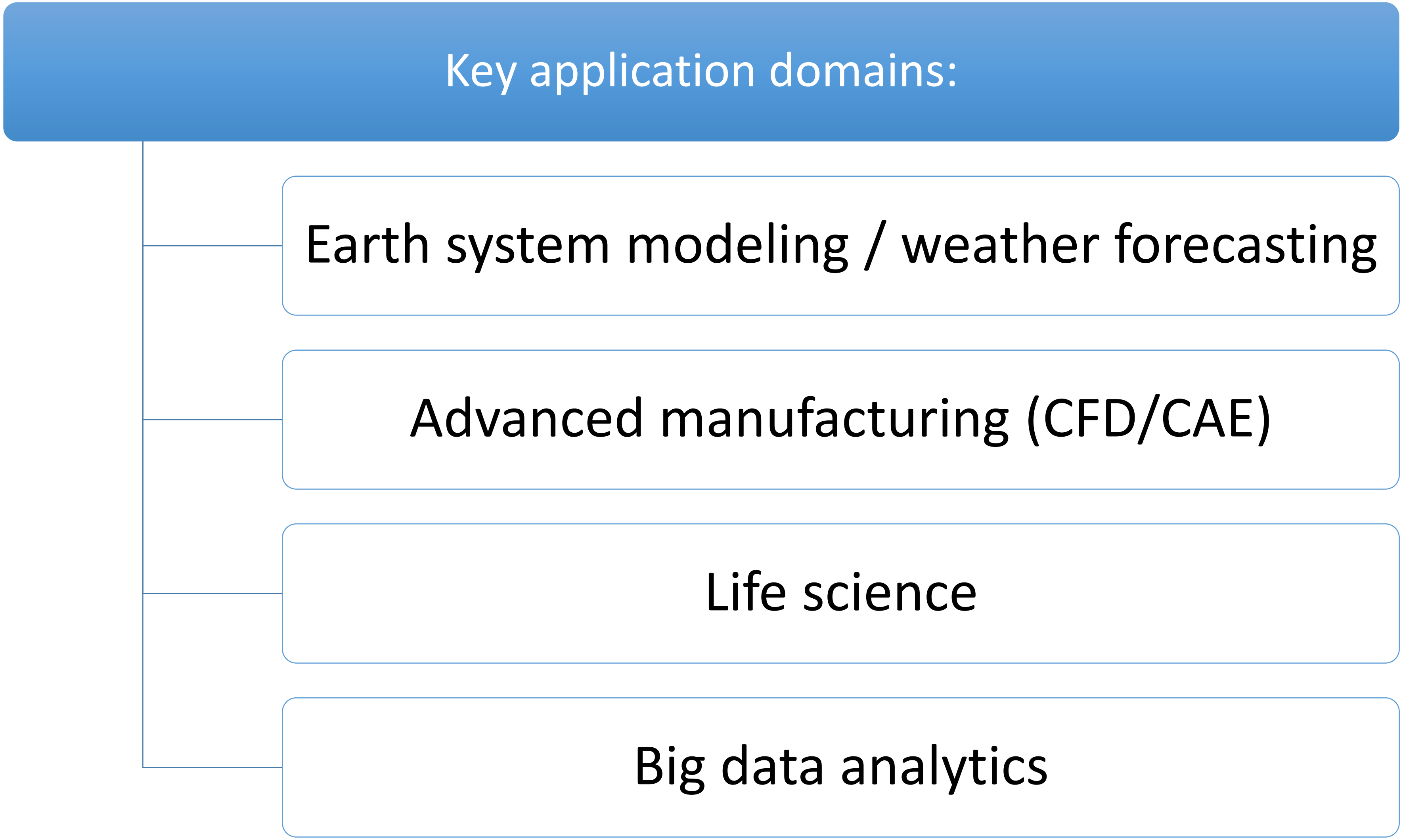
27-point stencil operator

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

Sunway TaihuLight: Applications

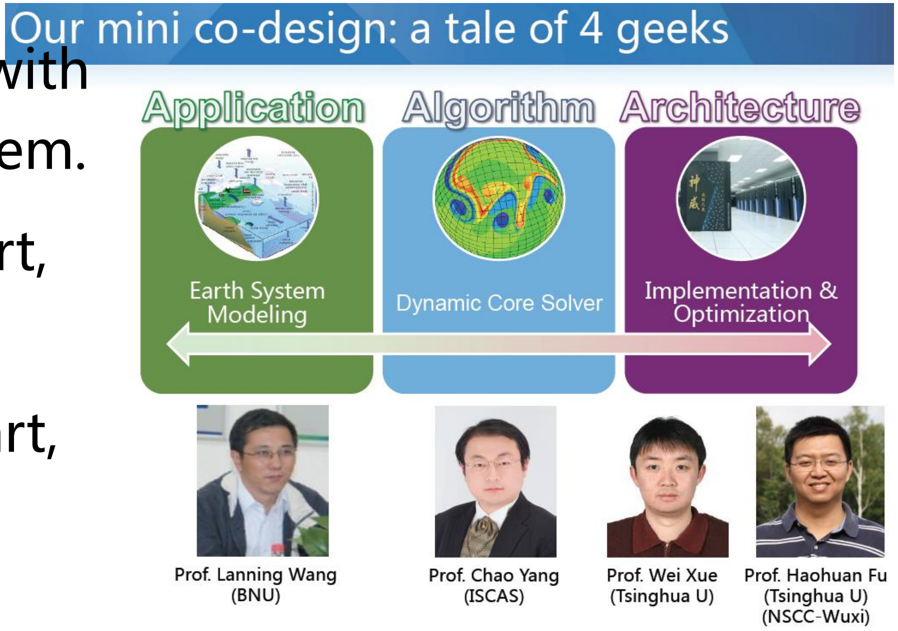


Key application domains:

- Earth system modeling / weather forecasting
- Advanced manufacturing (CFD/CAE)
- Life science
- Big data analytics

Applications on the TaihuLight

- 35 apps running on the system
  - 6 of them are running at full scale
  - 18 of them are running on half the machine
  - 20 applications on million cores
- Apps will typically run "out of the box”
  - No use of CPEs, just on MPE, with poor performance
  - Codes needs to be refactored to use CPE
- The Center has 20 people to help with optimizing apps to run on the system.
- CAM code 20K lines of code to start, ended with 100K lines, 10 people.
- Phase field 12K lines of code to start, ended with 20K, 3 people + help



Our mini co-design: a tale of 4 geeks

Application

Earth System Modeling

Algorithm

Dynamic Core Solver

Architecture

Implementation & Optimization

Prof. Lanning Wang (BNU)

Prof. Chao Yang (ISCAS)

Prof. Wei Xue (Tsinghua U)

Prof. Haohuan Fu (Tsinghua U) (NSCC-Wuxi)

Gordon Bell Award

- Since 1987 the Gordon Bell Prize is awarded at the SC conference to recognize outstanding achievement in high-performance computing.
- The purpose of the award is to track the progress of parallel computing, with emphasis on rewarding innovation in applying HPC to applications.
- Financial support of the $10,000 award is provided by Gordon Bell, a pioneer in high-performance and parallel computing.

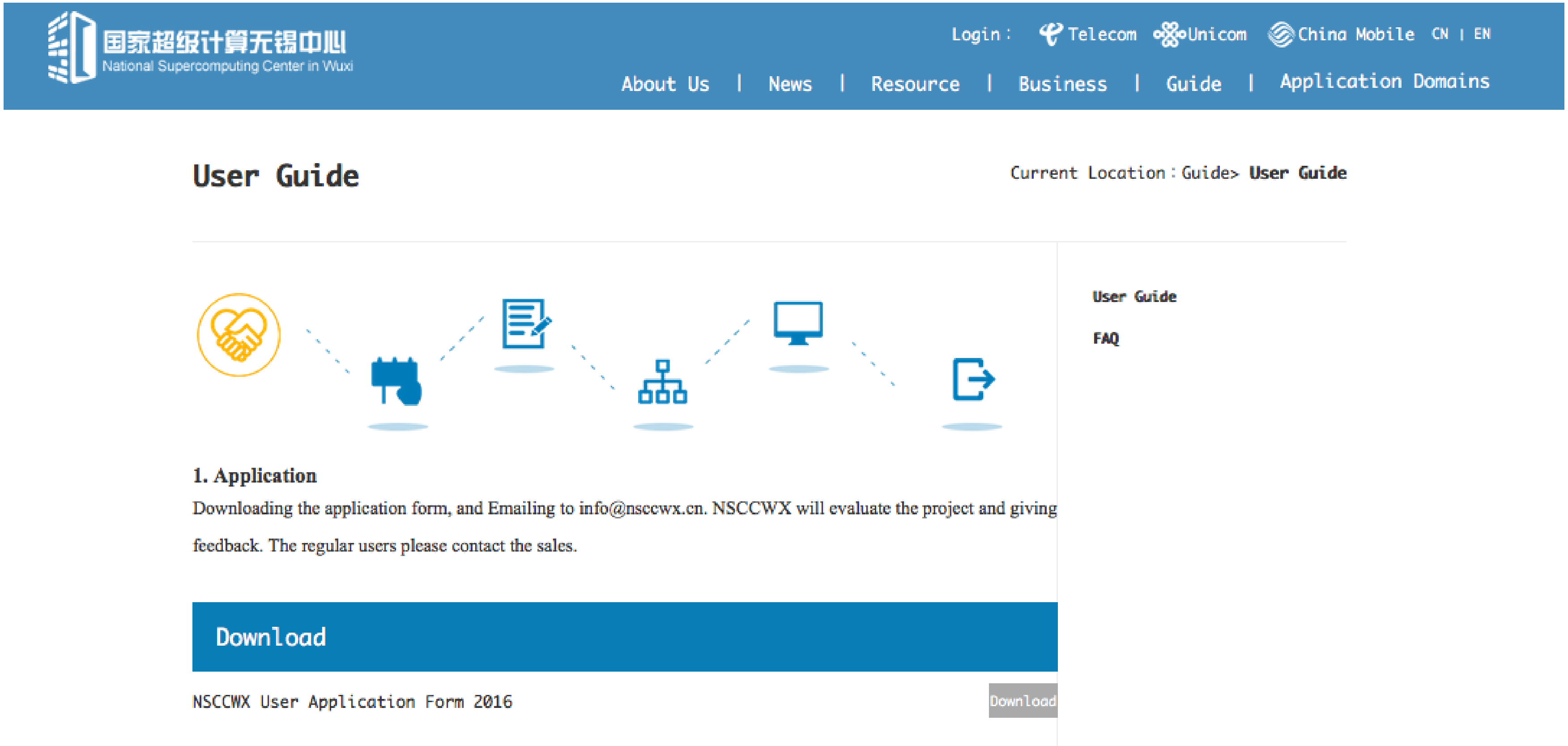
- Authors‘ mark their SC paper as a possible Gordon Bell Prize competitor.
- Gordon Bell committee reviews the papers and selects 6 papers for the competition.
- Presentations are made at SC and a winner is chosen.

Gordon Bell Award Finalists at SC16

- **“Modeling Dilute Solutions Using First-Principles Molecular Dynamics: Computing More than a Million Atoms with Over a Million Cores,”**
  - Lawrence-Livermore National Laboratory (Calif.)
- **“Towards Green Aviation with Python at Petascale,”**
  - Imperial College London (England)
- **“Simulations of Below-Ground Dynamics of Fungi: 1.184 Pflops Attained by Automated Generation and Autotuning of Temporal Blocking Codes,”**
  - RIKEN (Japan), Chiba University (Japan), Kobe University (Japan) and Fujitsu Ltd. (Japan)
- **“Extreme-Scale Phase Field Simulations of Coarsening Dynamics on the Sunway Taihulight Supercomputer,”**
  - Chinese Academy of Sciences, the University of South Carolina, Columbia University (New York), the National Research Center of Parallel Computer Engineering and Technology (China) and the National Supercomputing Center in Wuxi (China)
- **“A Highly Effective Global Surface Wave Numerical Simulation with Ultra-High Resolution,”**
  - First Institute of Oceanography (China), National Research Center of Parallel Computer Engineering and Technology (China) and Tsinghua University (China)
- **“10M-Core Scalable Fully-Implicit Solver for Nonhydrostatic Atmospheric Dynamics,”**
  - Chinese Academy of Sciences, Tsinghua University (China), the National Research Center of Parallel Computer Engineering and Technology (China) and Beijing Normal University (China)

Sunway TaihuLight is Available …

- The TaihuLight was put on the internet last month.
- If you would like to use the TaihuLight, go to…
  - http://www.nsccwx.cn/wxcyw/process.php?word=process&i=54



国家超级计算无锡中心
National Supercomputing Center in Wuxi

Login： Telecom Unicom China Mobile CN | EN

About Us | News | Resource | Business | Guide | Application Domains

User Guide

Current Location：Guide> **User Guide**

User Guide

FAQ

1. Application

Downloading the application form, and Emailing to info@nsccwx.cn. NSCCWX will evaluate the project and giving feedback. The regular users please contact the sales.

Download

NSCCWX User Application Form 2016 Download

Peak Performance - Per Core

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

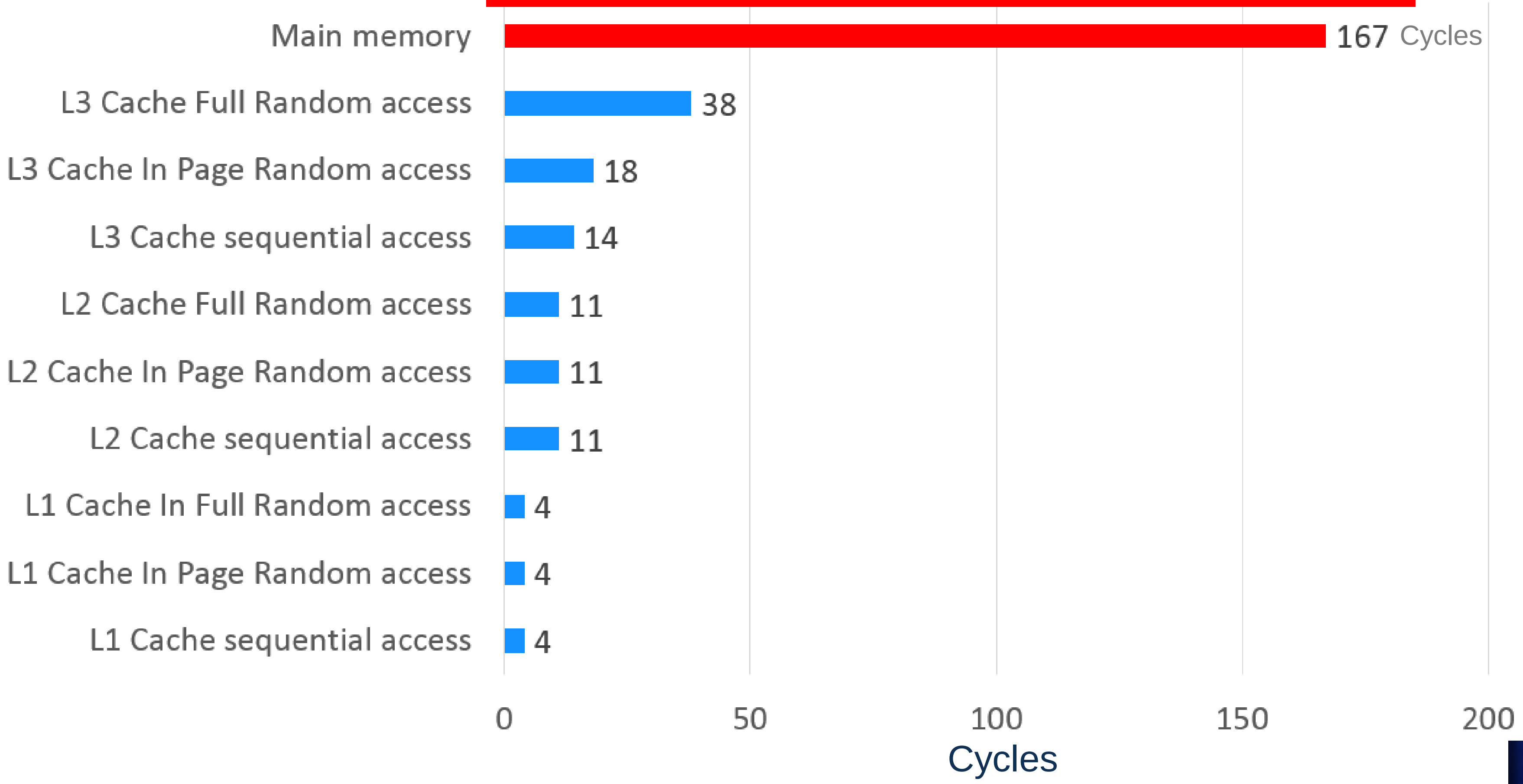
Floating point operations per cycle per core

- 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) AVX 512
  - 32 flops/cycle DP & 64 flops/cycle SP
  - Knight's Landing

We are here (almost)

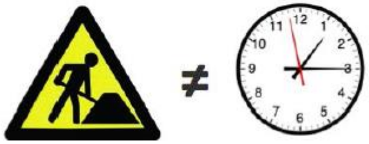
CPU Access Latencies in Clock Cycles

In 167 cycles can do 2672 DP Flops



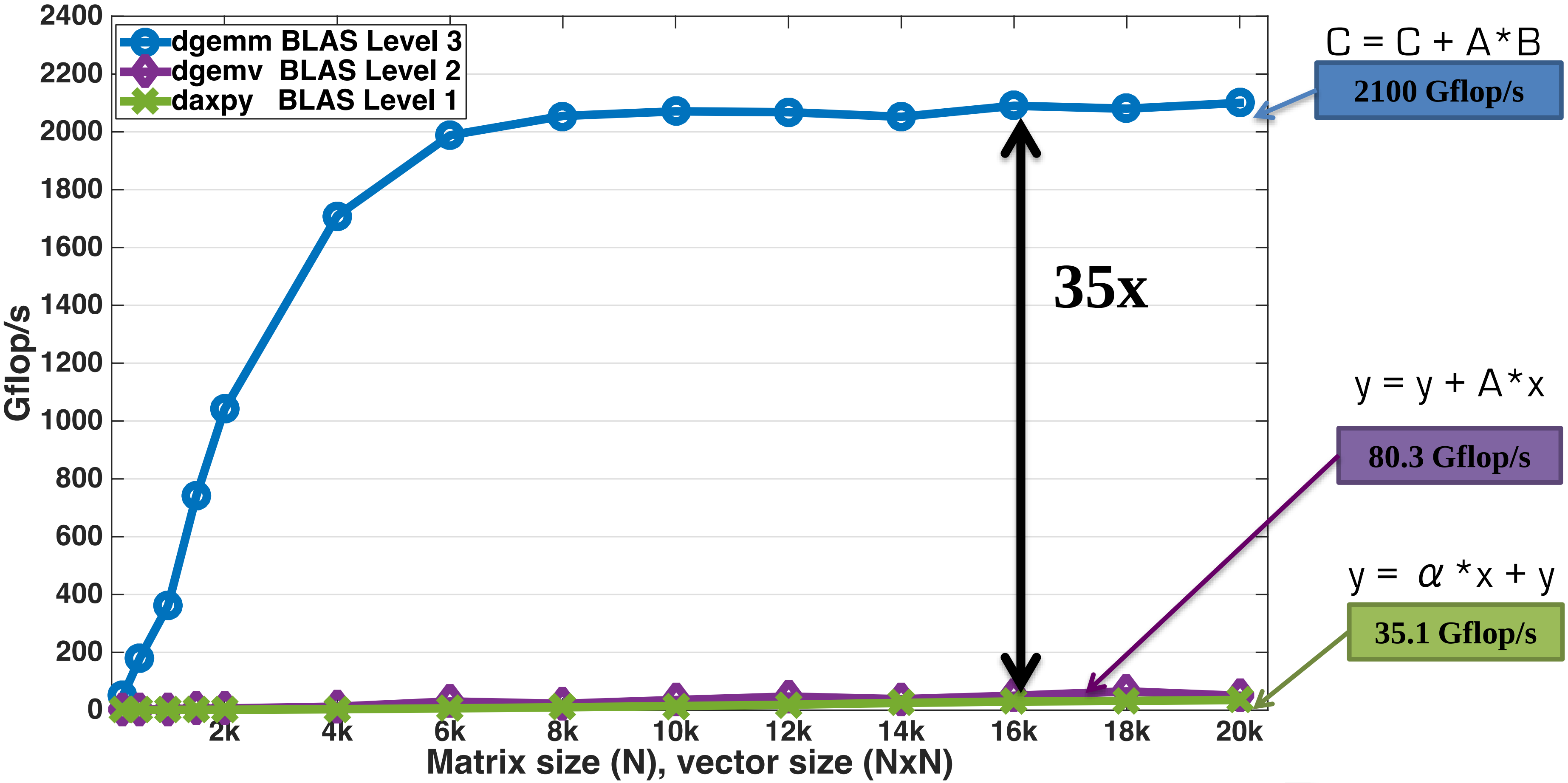
Classical Analysis of Algorithms May Not be Valid

- 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.



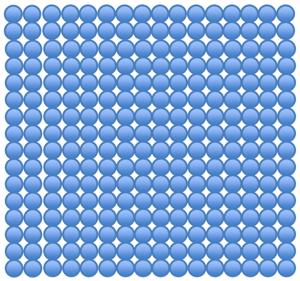
Level 1, 2 and 3 BLAS

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



68 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

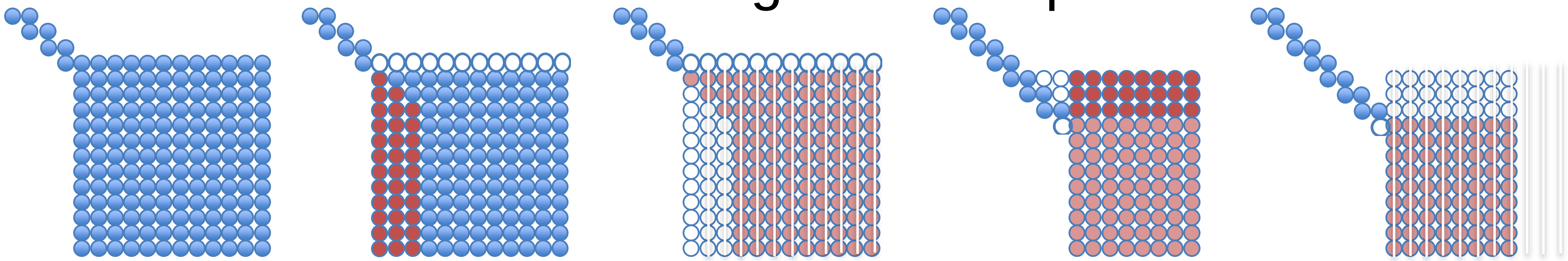
Singular Value Decomposition

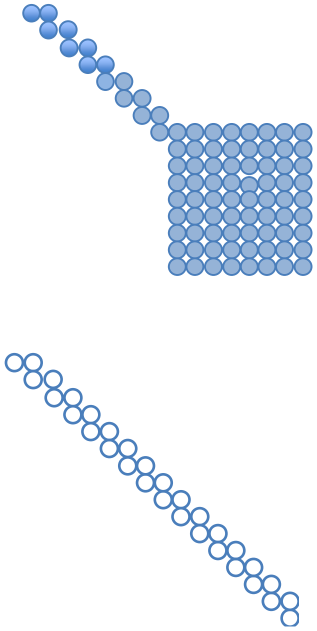


LAPACK Version 1991

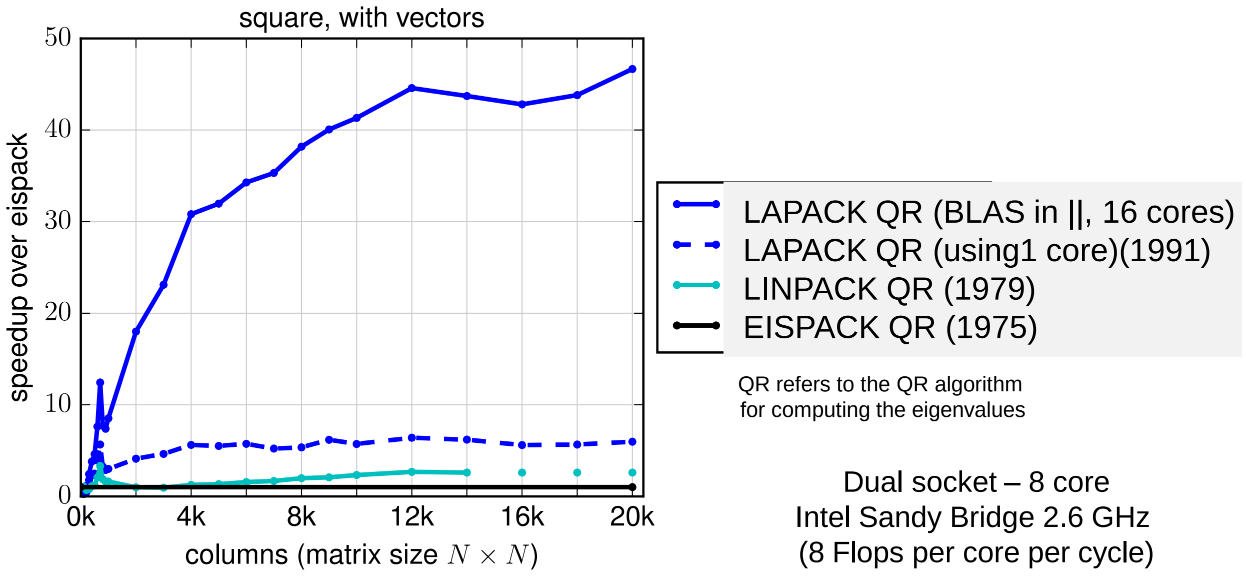
Level 1, 2, & 3 BLAS

First Stage 8/3 n^3 Ops





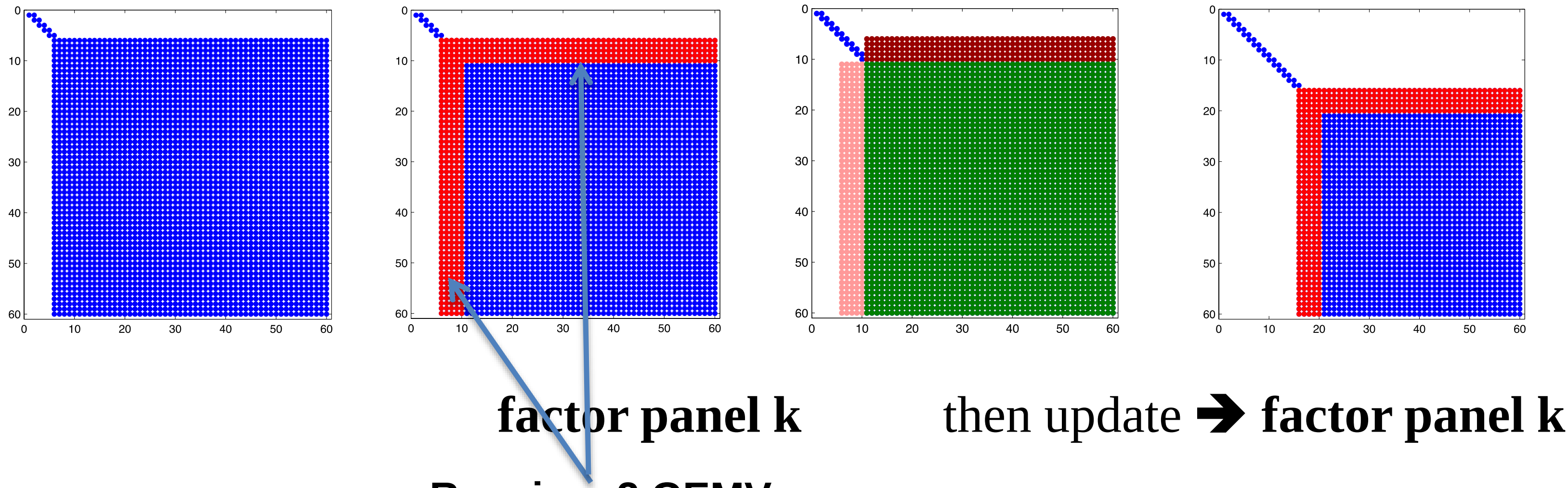
3 Generations of software compared



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



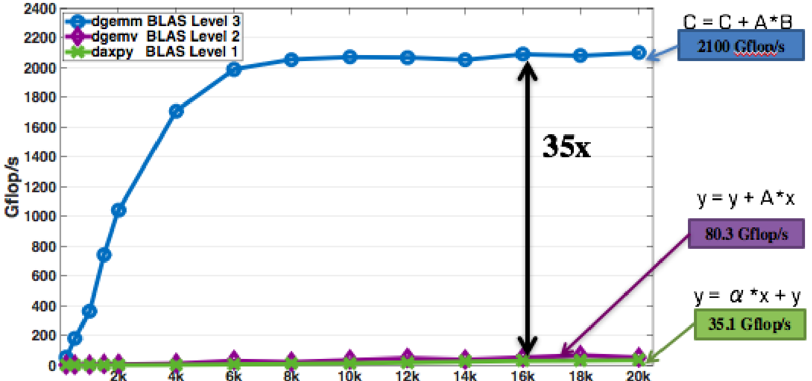
factor panel k then update ➔ **factor panel k+1**

Requires 2 GEMVs

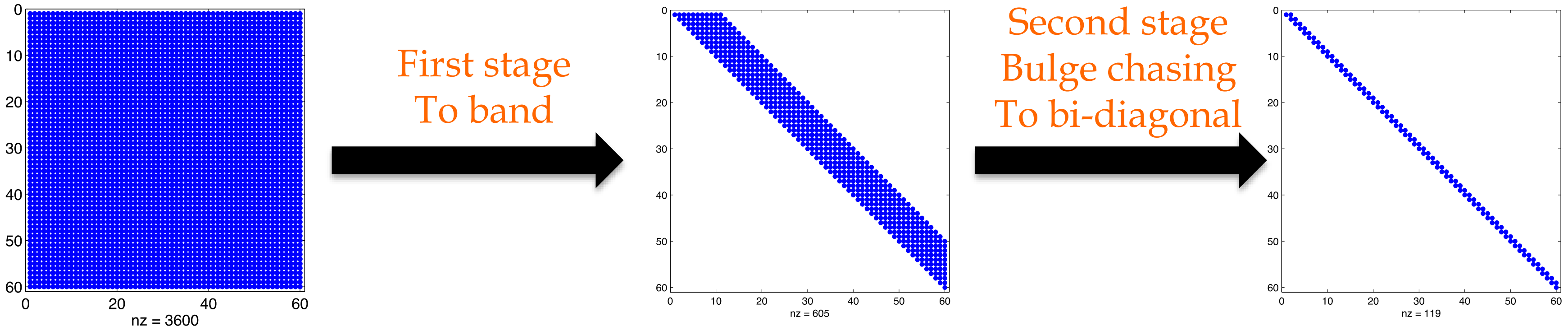
$Q*A*P^H$

✷ Characteristics

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



Recent Work on 2-Stage Algorithm



First stage
To band

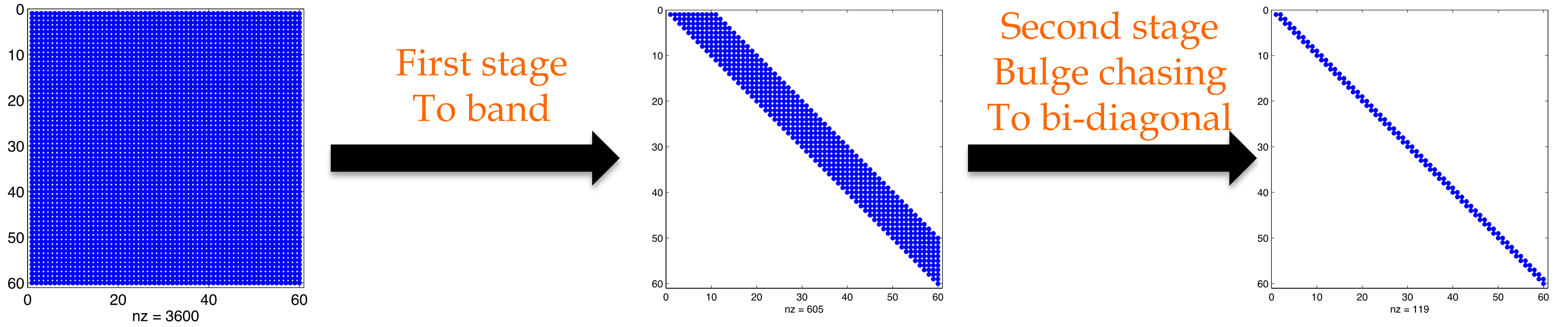
Second stage
Bulge chasing
To bi-diagonal

✸ 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



First stage
To band

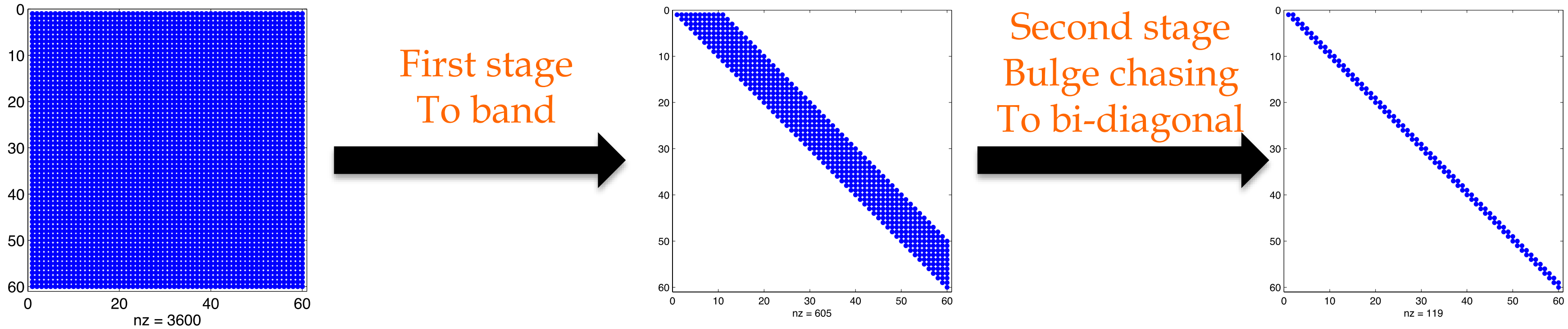
Second stage
Bulge chasing
To bi-diagonal

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

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

More Flops, original did 8/3 n^3
25% More flops

Recent work on developing new 2-stage algorithm



First stage
To band

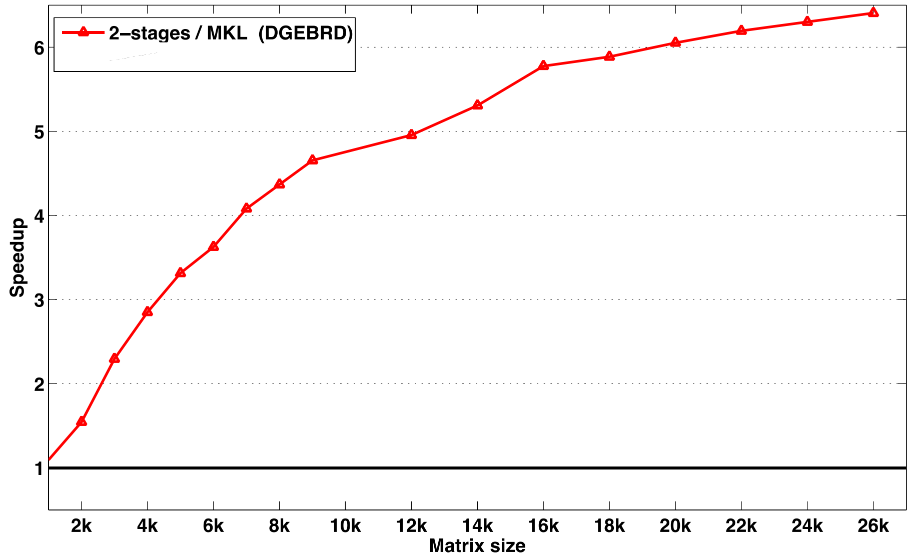
Second stage
Bulge chasing
To bi-diagonal

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

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

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

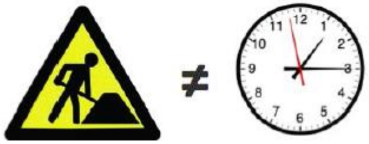
$$\Longrightarrow 1.8 \leq \text{Speedup} \leq 7$$



16 Sandy Bridge cores 2.6 GHz

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

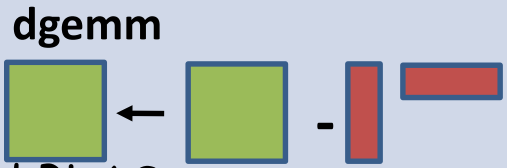
25% More flops and 1.8 – 6 times faster

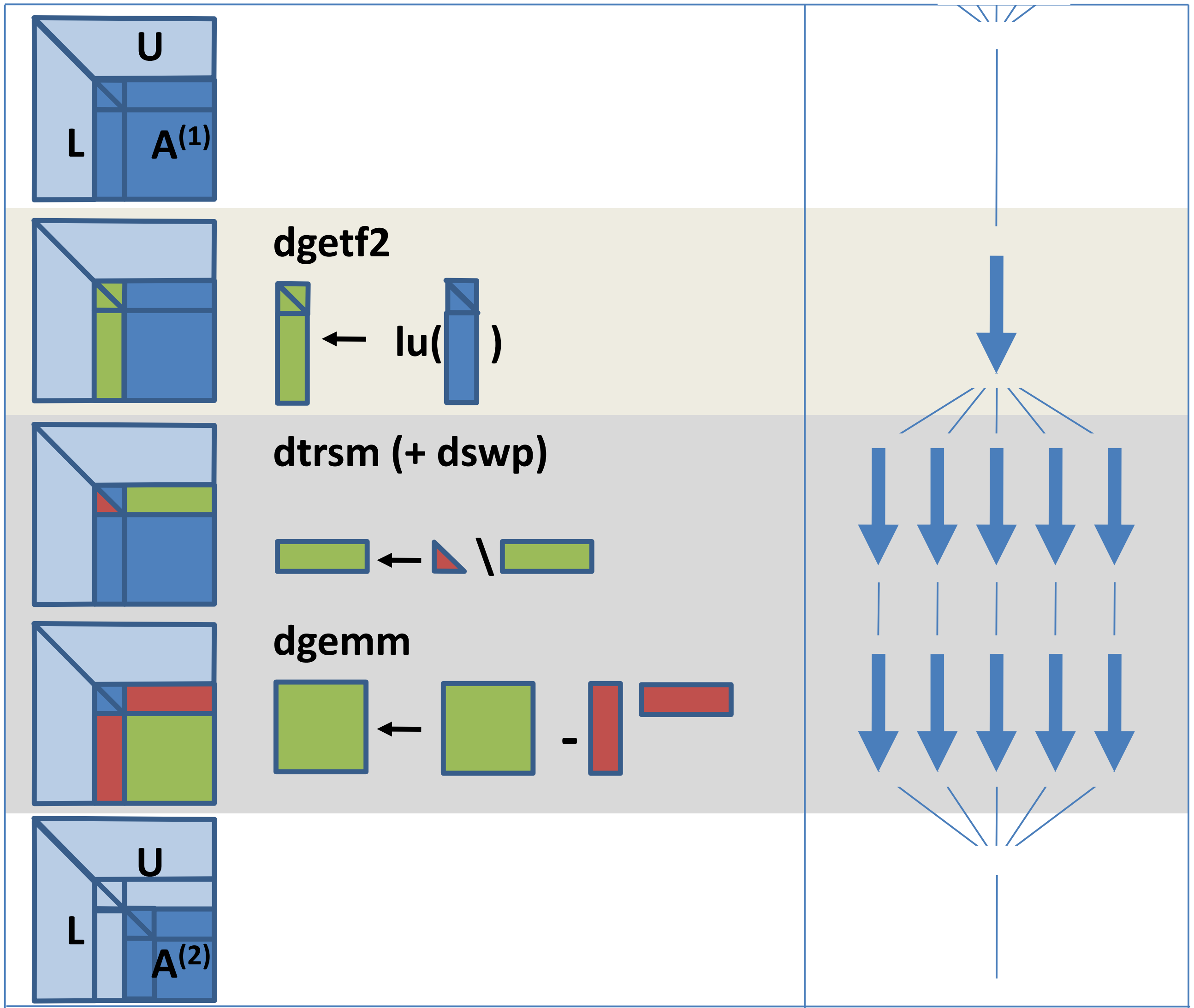


Parallelization of LU and QR.

Parallelize the update:

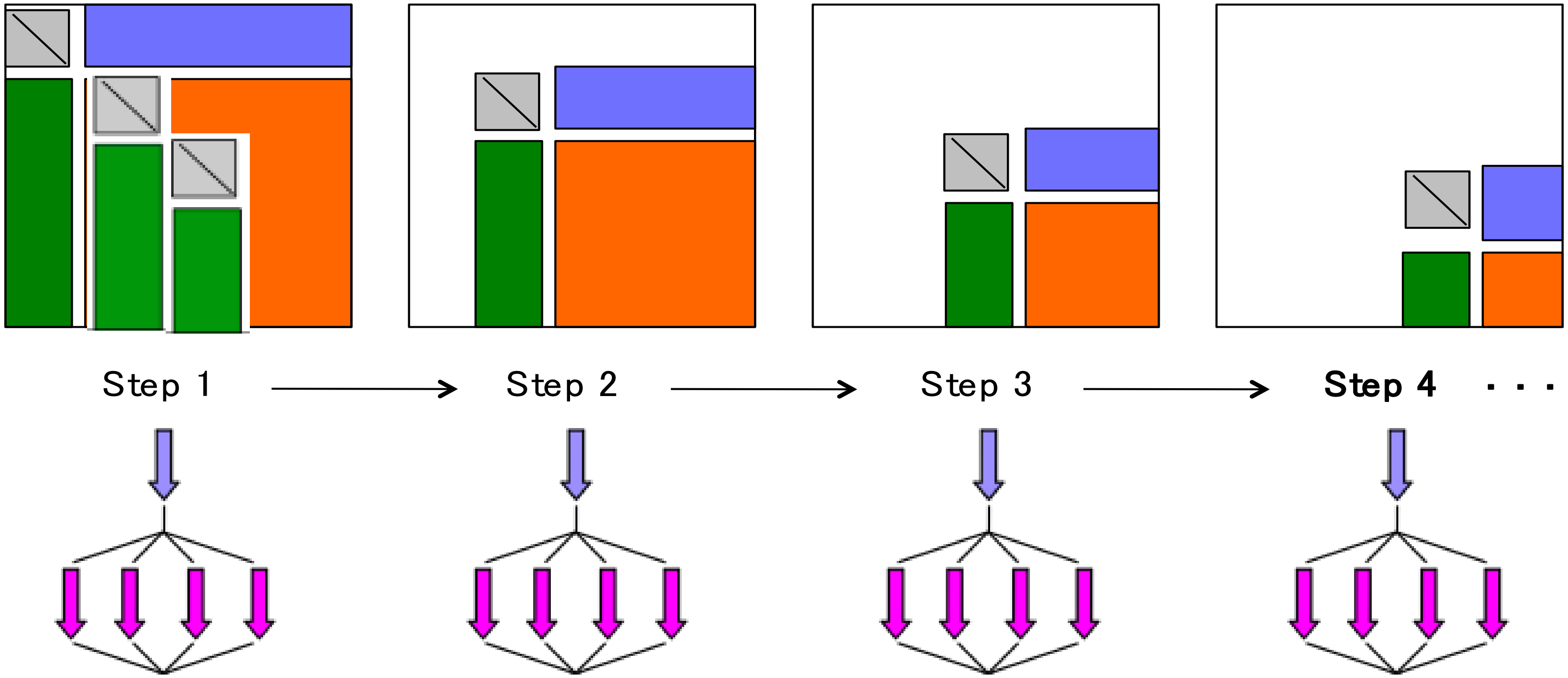
- Easy and done in any reasonable software.
- This is the $2/3n^3$ term in the FLOPs count.
- Can be done efficiently with LAPACK+multithreaded BLAS

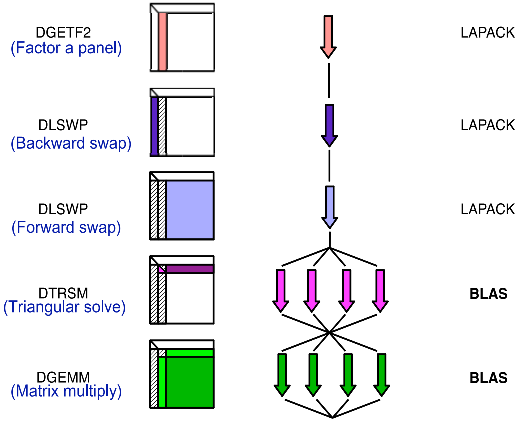




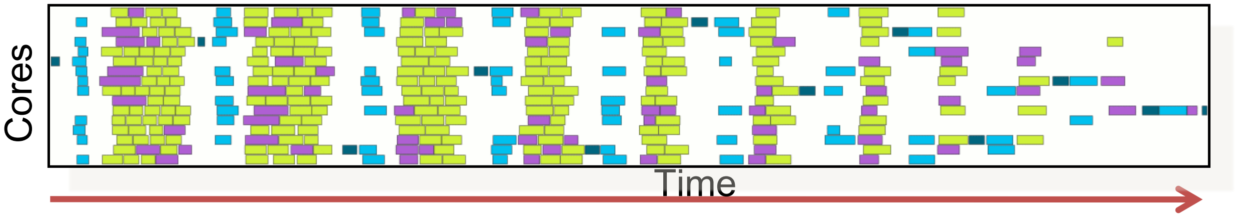
Fork - Join parallelism
Bulk Sync Processing

Synchronization (in LAPACK LU)





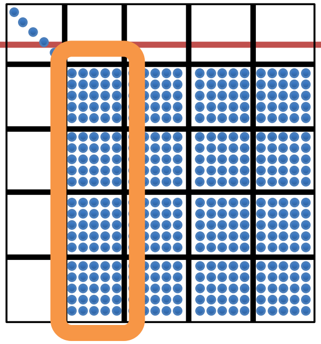
- fork join
- bulk synchronous processing

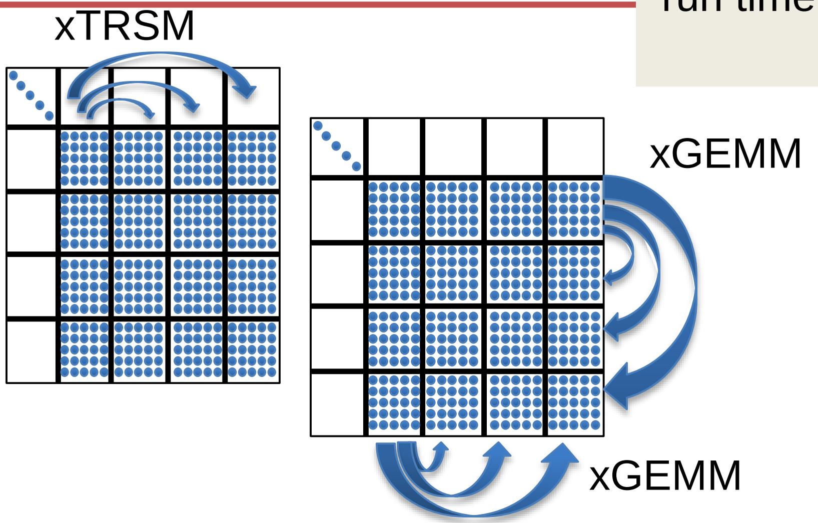


PLASMA LU Factorization

Dataflow Driven

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

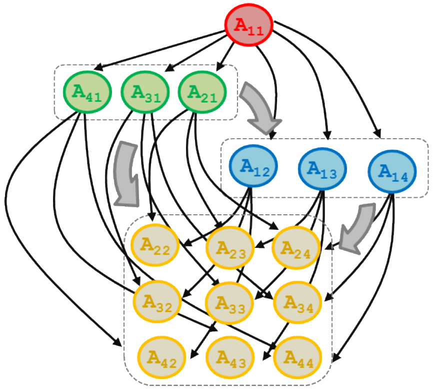




Sparse / Dense Matrix System

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

DAG-based factorization



Batched LA

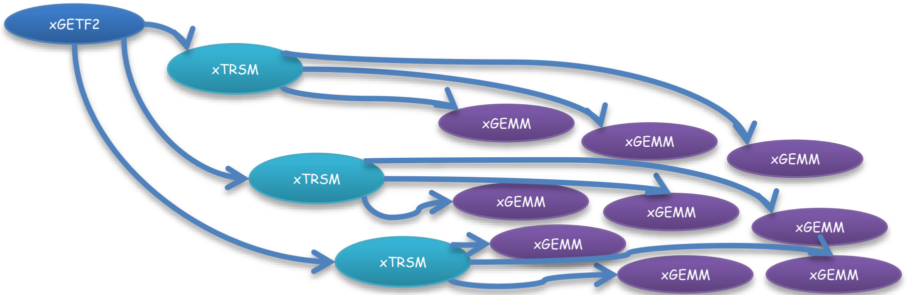
▯ **LU, QR,** or **Cholesky** on small diagonal matrices

▯ **TRSM**s, **QR**s, or **LU**s

▯ **TRSM**s, **TRMM**s

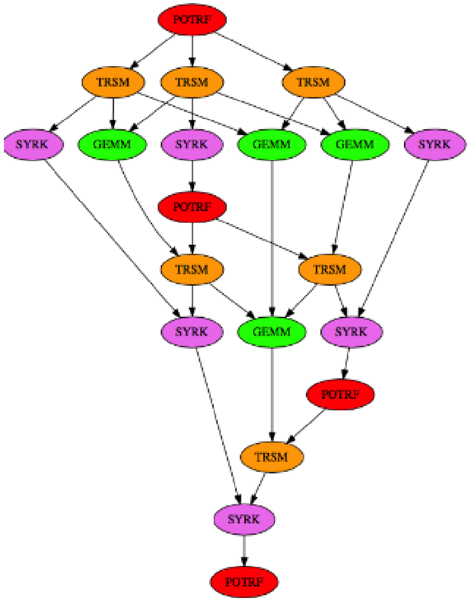
▯ **Updates (Schur complement) GEMM**s, **SYRK**s, **TRMM**s

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

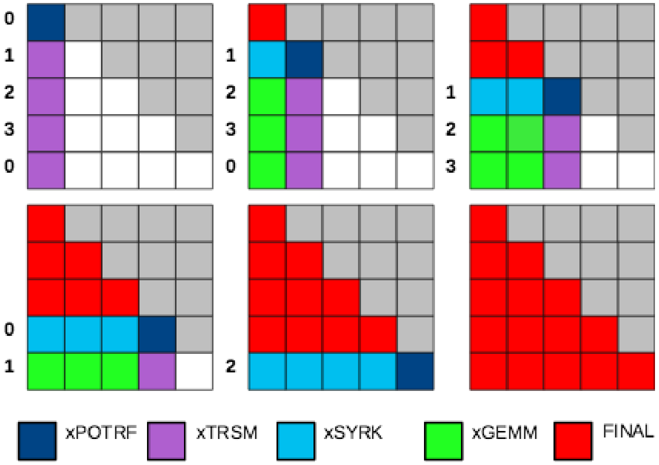


OpenMP tasking

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



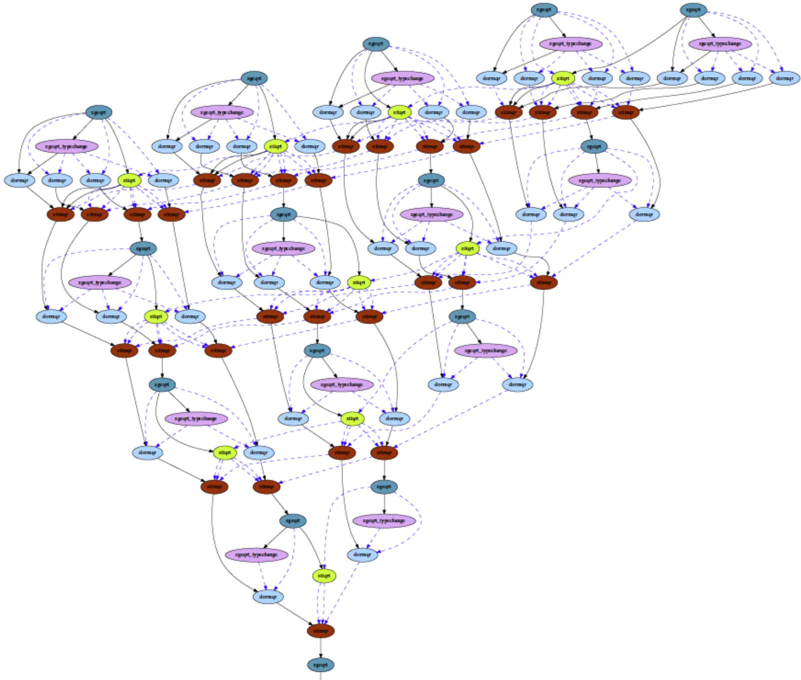
Tiled Cholesky Decomposition

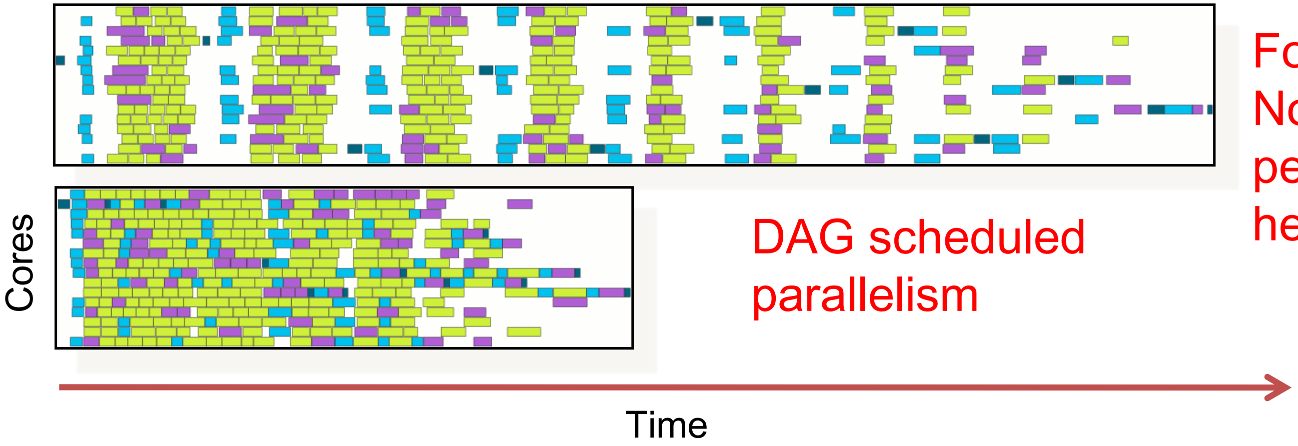


```
#pragma omp parallel
#pragma omp master
{   CHOLESKY( A );   }
CHOLESKY( A ) {
    for (k = 0; k < M; k++) {
        #pragma omp task depend(inout:A(k,k)[0:tilesize]
        {   POTRF( A(k,k) );   }
        for (m = k+1; m < M; m++) {
            #pragma omp task \
                depend(in:A(k,k)[0:tilesize])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





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

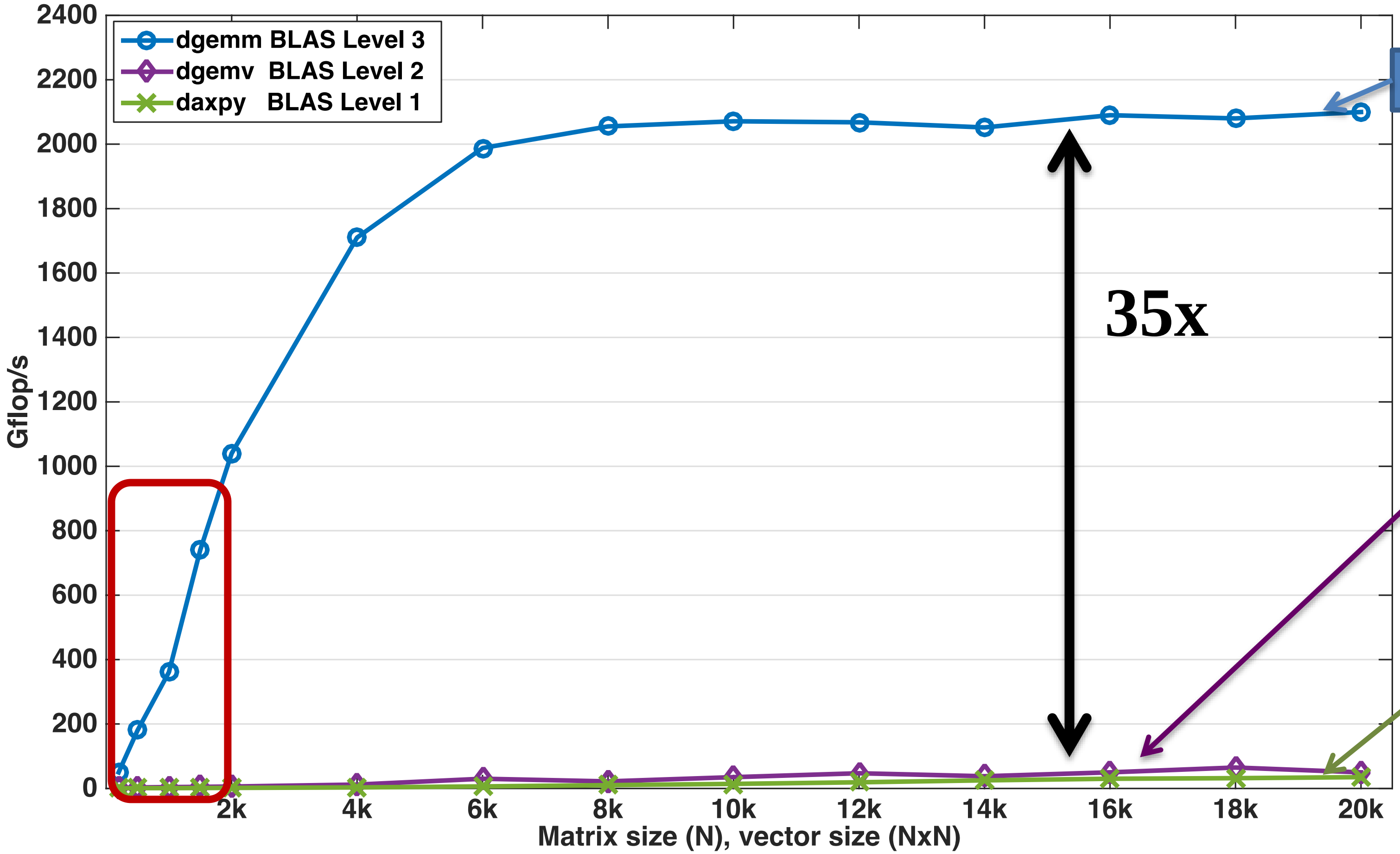
DAG scheduled parallelism

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

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



C = C + A*B

2100 Gflop/s

y = y + A*x

60.3 Gflop/s

y = α *x + y

35.1 Gflop/s

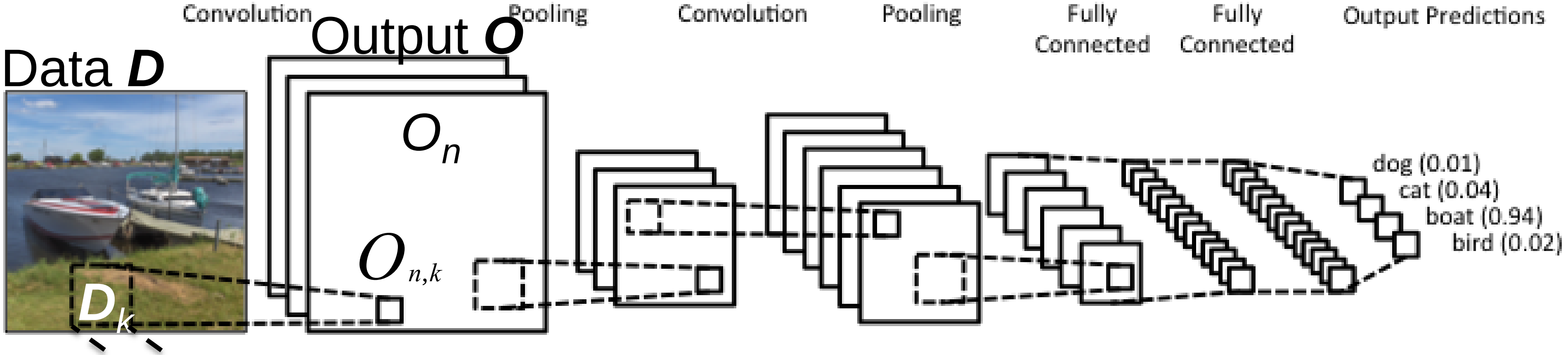
68 cores Intel Xeon Phi KNL, 1.3 GHz
The theoretical peak double precision is 2662 Gflop/s
Compiled with icc and using Intel MKL 2017b1 20160506

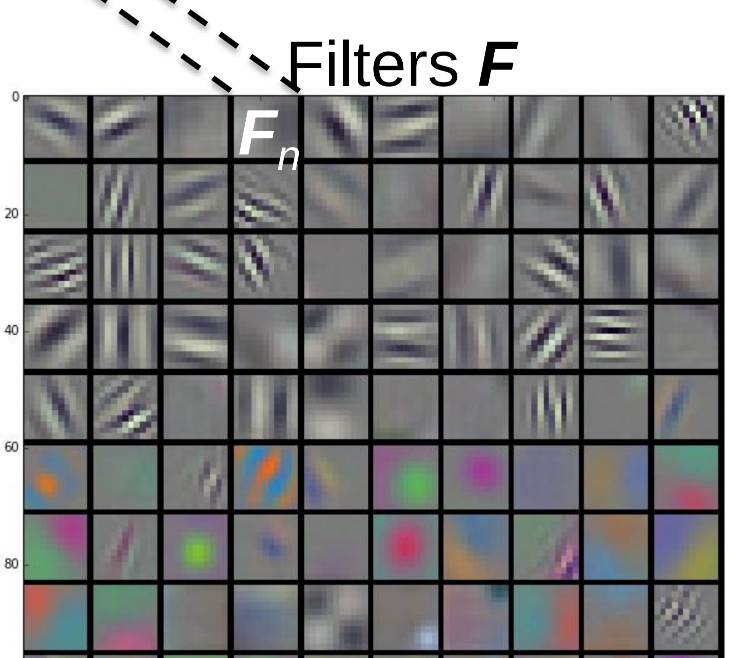
Machine Learning

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 Fi (feature detector) and input image D (data):





Convolution operation:

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

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

This problem can get away with 16 bit floating point

=> Some architectures are now implementing this

Examples

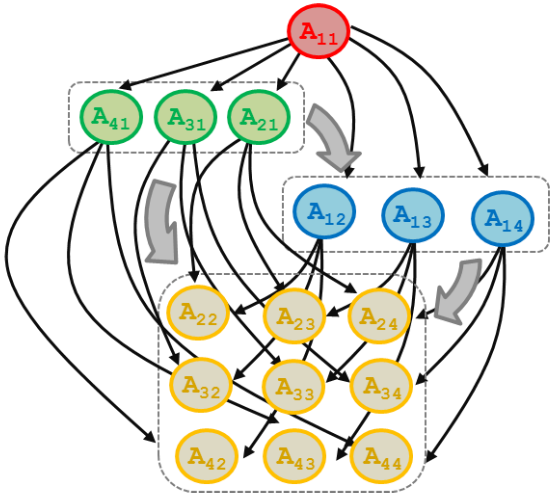
Need of **Batched** routines for **Numerical LA**

[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

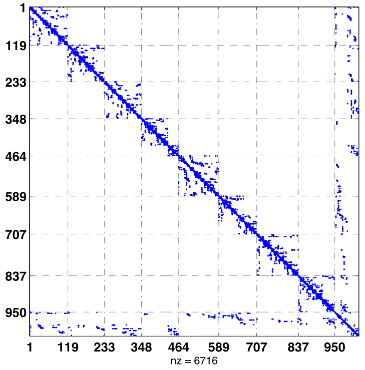
$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

DAG-based factorization

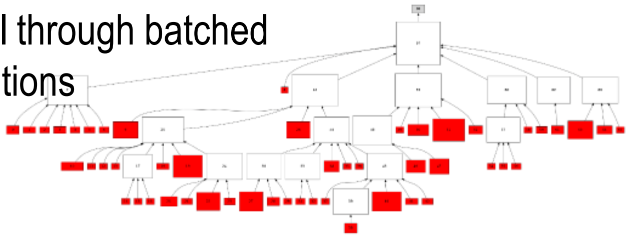


To capture main LA patterns needed in a numerical library for Batched LA

- **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



- 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



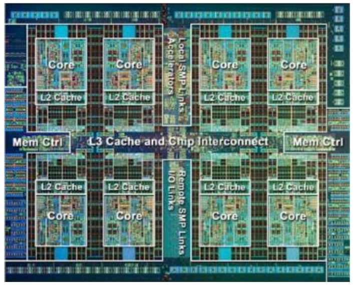
MAGMA Batched Computations CPU

1. Non-batched computation

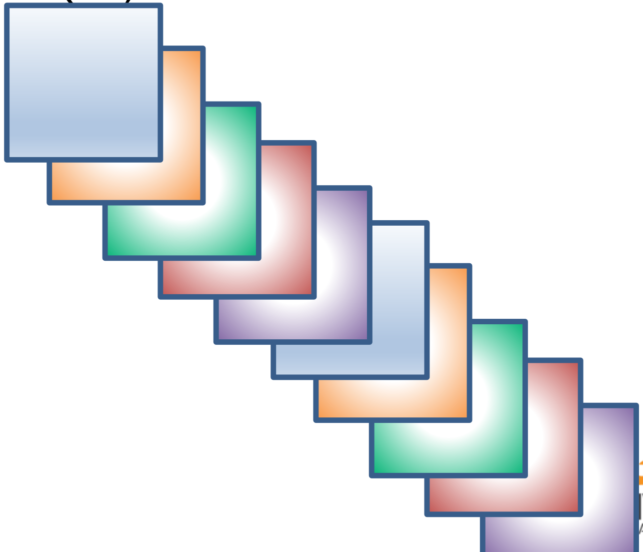
loop over the matrices one by one and compute either:

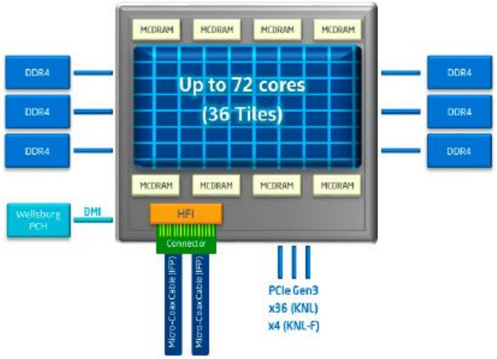
- One call for each matrix.
- Sequentially wasting all the other cores, and attaining very poor performance

- 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)


Core
L2 Cache
Local SMP Links Accelerators
Mem Ctrl
L3 Cache and Chip Interconnect
Remote SMP Links I/O Links

```
for (i=0; i<batchount; i++)
      dgemm(…)
```





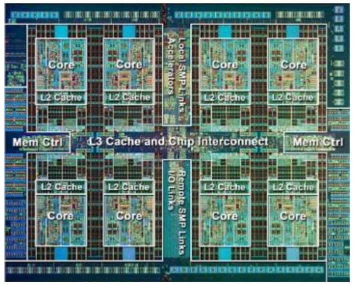
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

MAGMA Batched Computations CPU

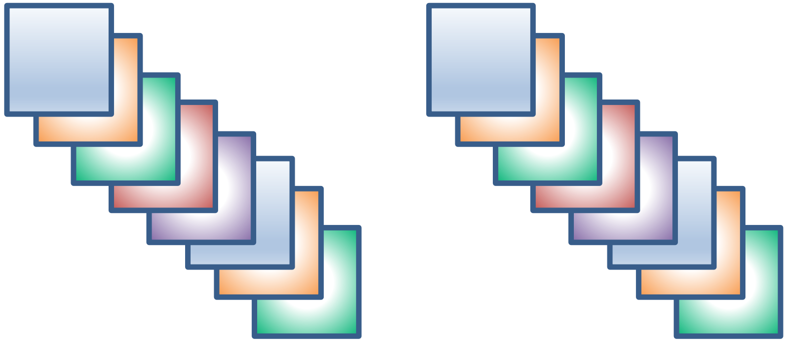
2. Batched computation

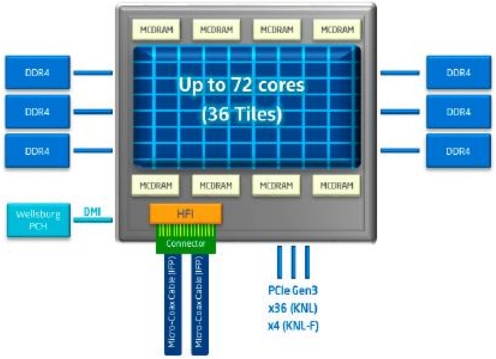
loop over the matrices and assign a matrix to each core working on it sequentially and independently

- 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



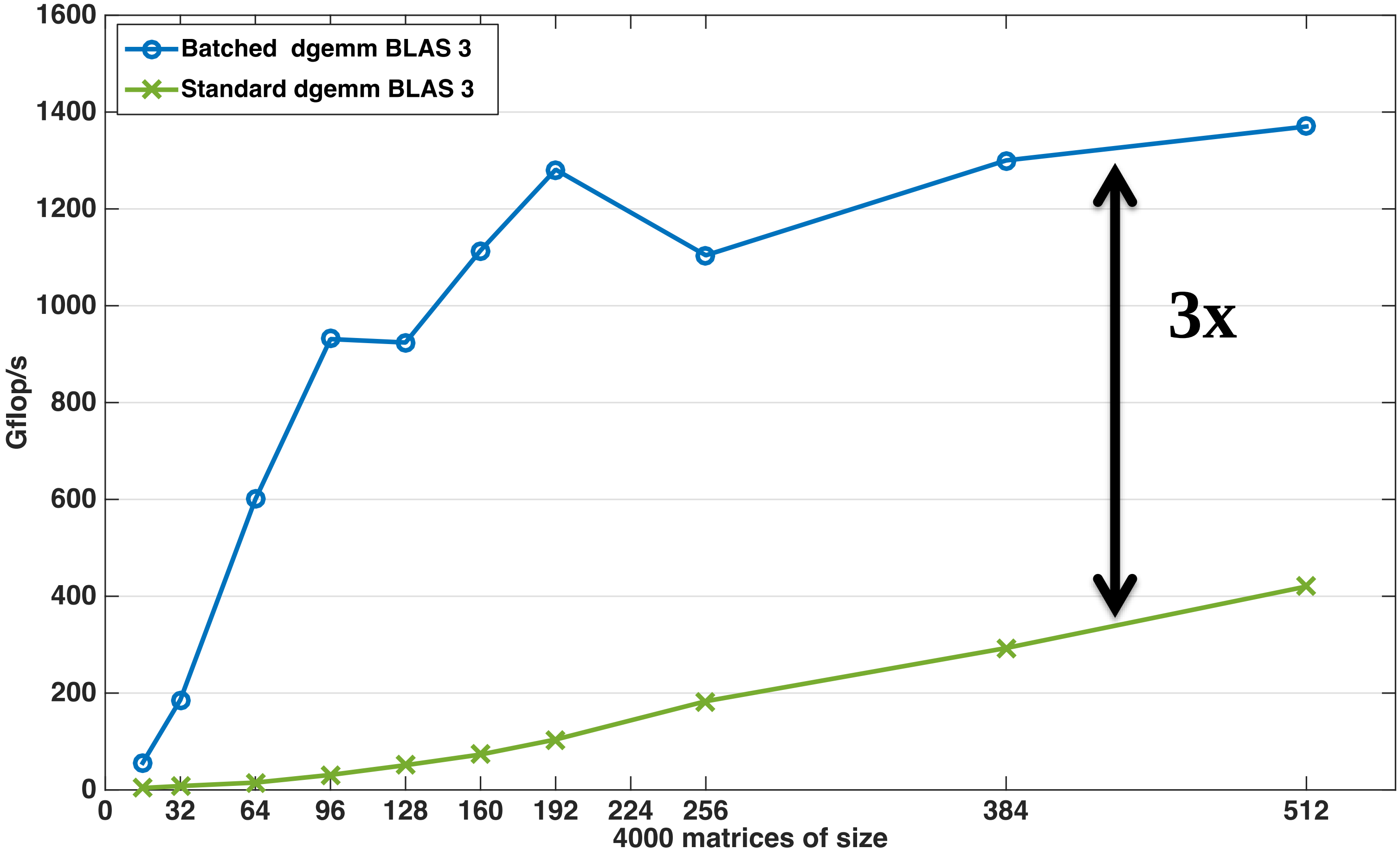
```
for (i=cpu_id; i<batchcount; i+=n_cpu)
    batched_dgemm(…)
```





Level 1, 2 and 3 BLAS

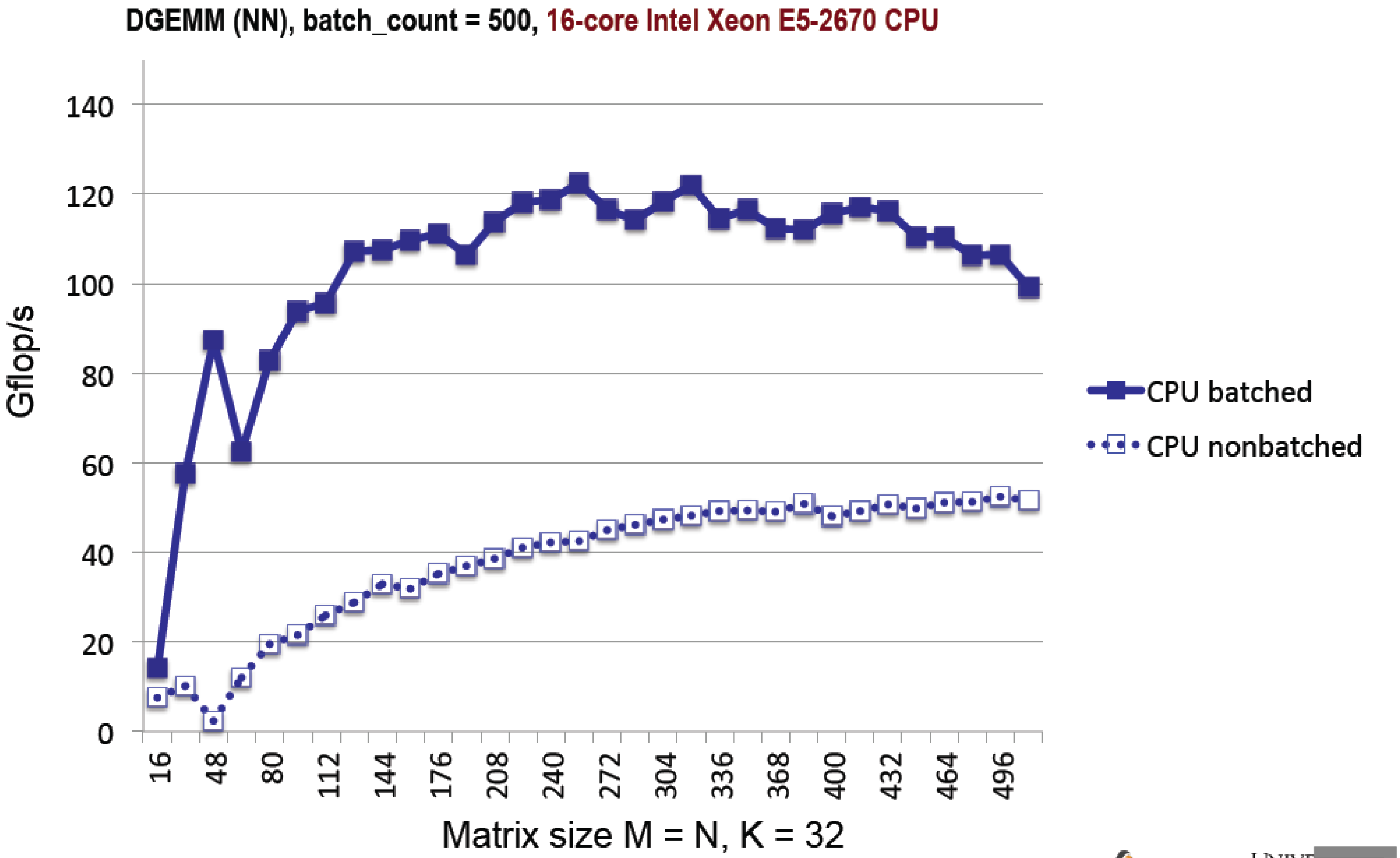
68 cores **Intel Xeon Phi KNL**, 1.3 GHz, Peak DP = 2662 Gflop/



C = C + A*B

68 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



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.

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

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

52

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.

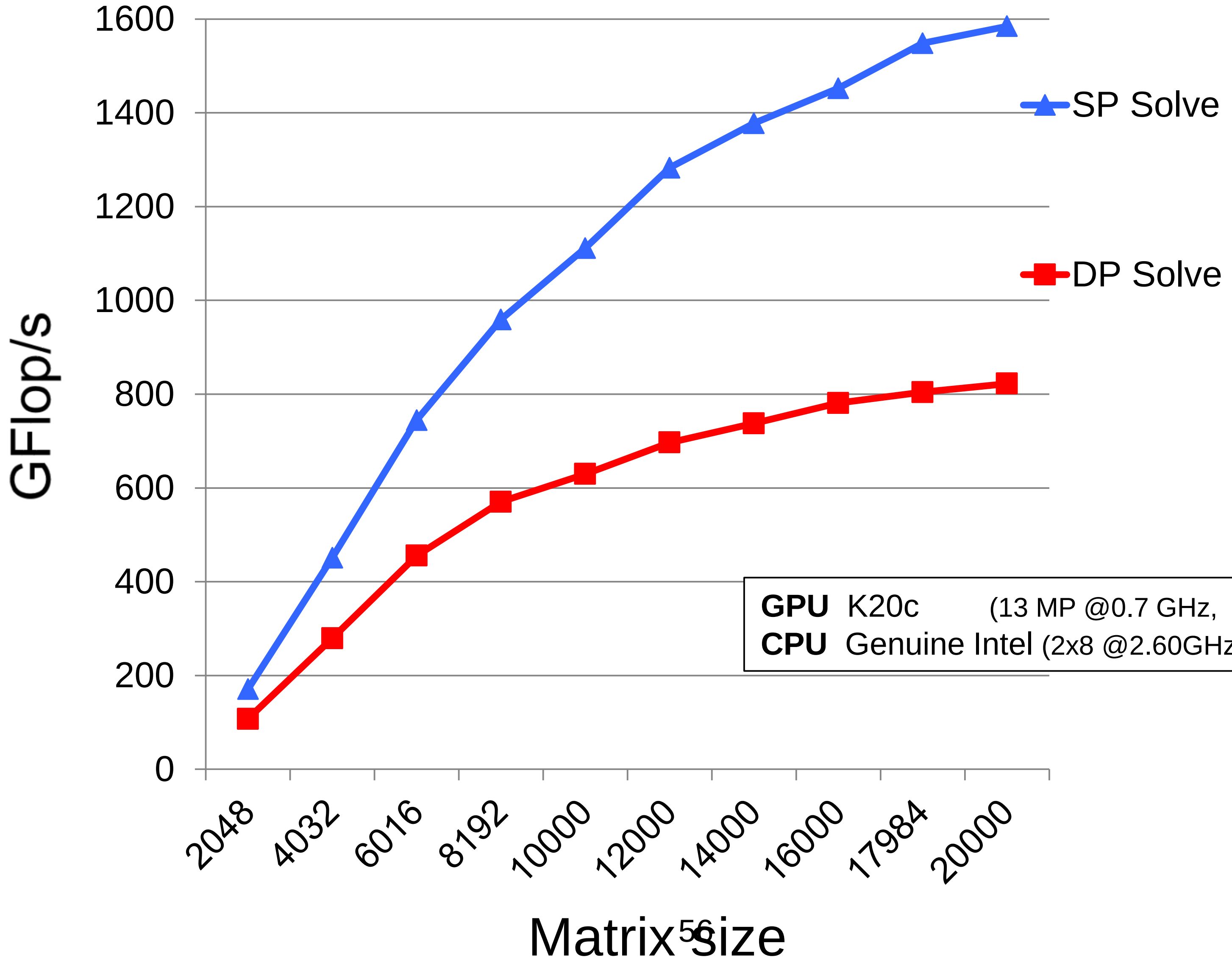
```
L U = lu(A)                        SINGLE      $O(n^3)$
x = L\(U\b)                        SINGLE      $O(n^2)$
r = b – Ax                         DOUBLE      $O(n^2)$
WHILE || r || not small enough
        z = L\(U\r)                SINGLE      $O(n^2)$
        x = x + z                  DOUBLE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

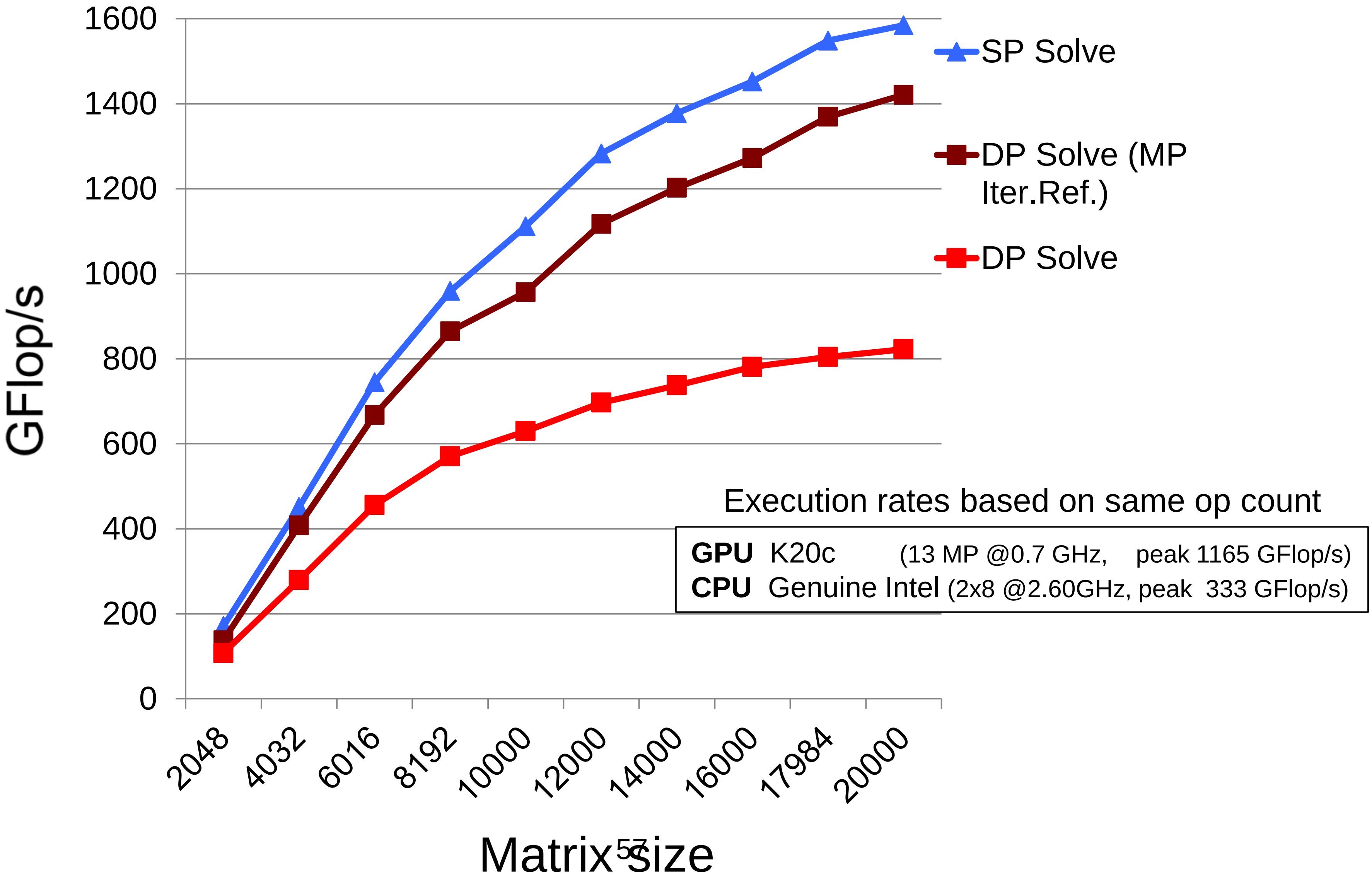
Solving general dense linear systems using mixed precision iterative refinement



GPU K20c (13 MP @0.7 GHz, peak 1165 GFlop/s)
CPU Genuine Intel (2x8 @2.60GHz, peak 333 GFlop/s)

Mixed precision iterative refinement

Solving general dense linear systems using mixed precision iterative refinement

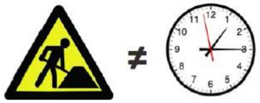


Execution rates based on same op count

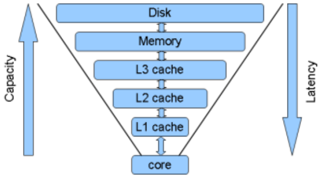
GPU K20c (13 MP @0.7 GHz, peak 1165 GFlop/s)
CPU Genuine Intel (2x8 @2.60GHz, peak 333 GFlop/s)

Software and Algorithm Must Keep Pace with the Changes in Hardware

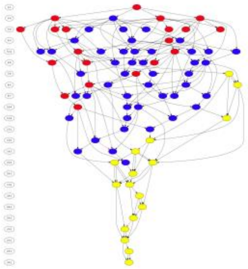
- Classical analysis of algorithms may not be valid,
  - # of floating point ops ≠ computation time.



- Algorithms and software must take advantage by reducing data movement.
  - Need latency tolerance in our algorithms



- Communication and synchronization reducing algorithms and software are critica
  - As parallelism grows



- Many existing algorithms can’t fully exploit the features of modern architecture
- Time to rethink

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

intel

NVIDIA

The MathWorks

AMD

NSF

DEPARTMENT OF DEFENSE · UNITED STATES OF AMERICA

U.S. DEPARTMENT OF ENERGY

UMEÅ UNIVERSITET

Umeå University

Inria informatics mathematics

INRIA

Science & Technology Facilities Council

Rutherford Appleton Laboratory

MANCHESTER 1824
The University of Manchester

University of Manchester