Toward petaflop numerical simulation on parallel hybrid architectures

june, 6-10, 2011
INRIA
Sophia Antipolis, France

The Past, Present and Future of HPC (Learning from the Past)

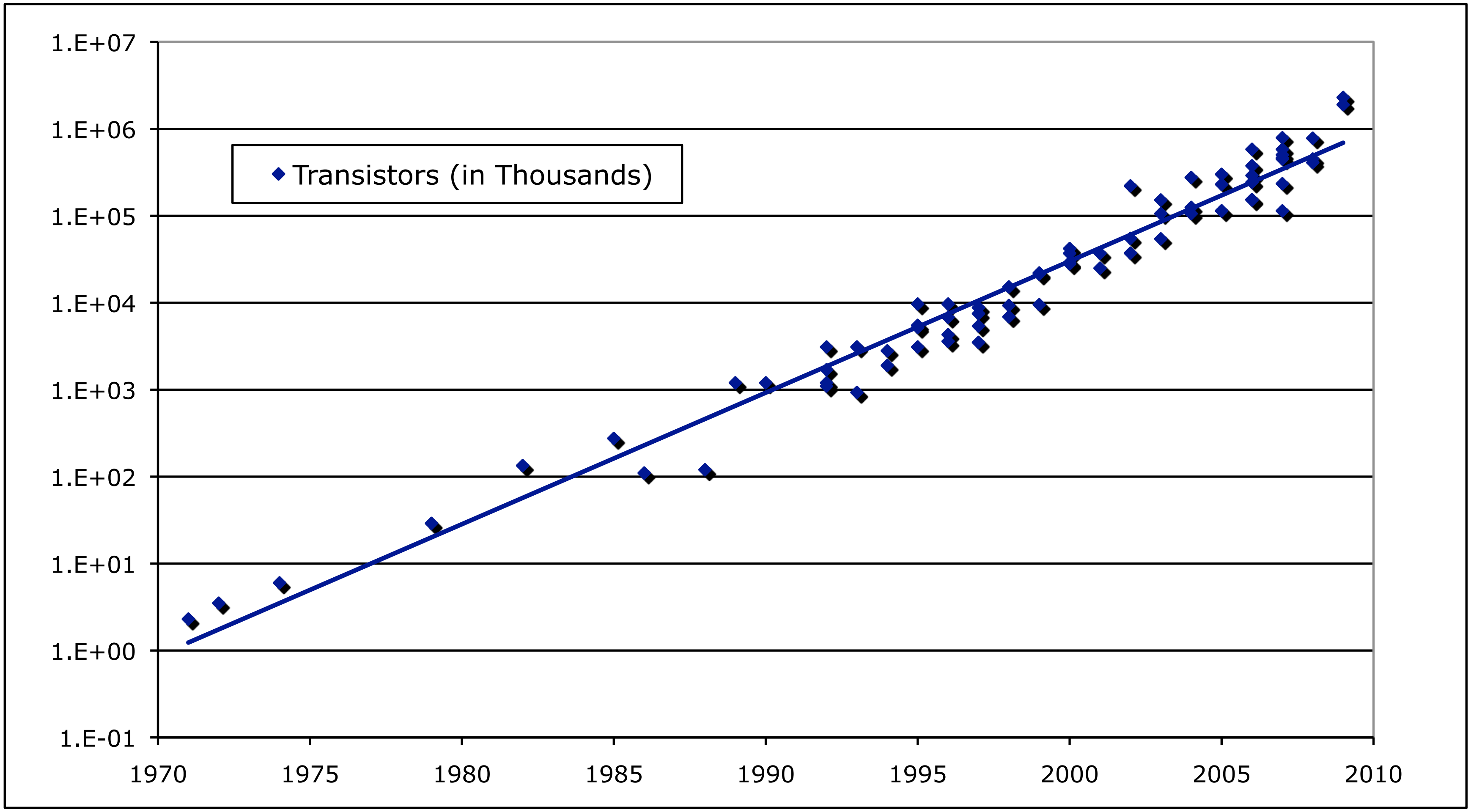
Jack Dongarra

University of Tennessee
Oak Ridge National Laboratory
University of Manchester

First …

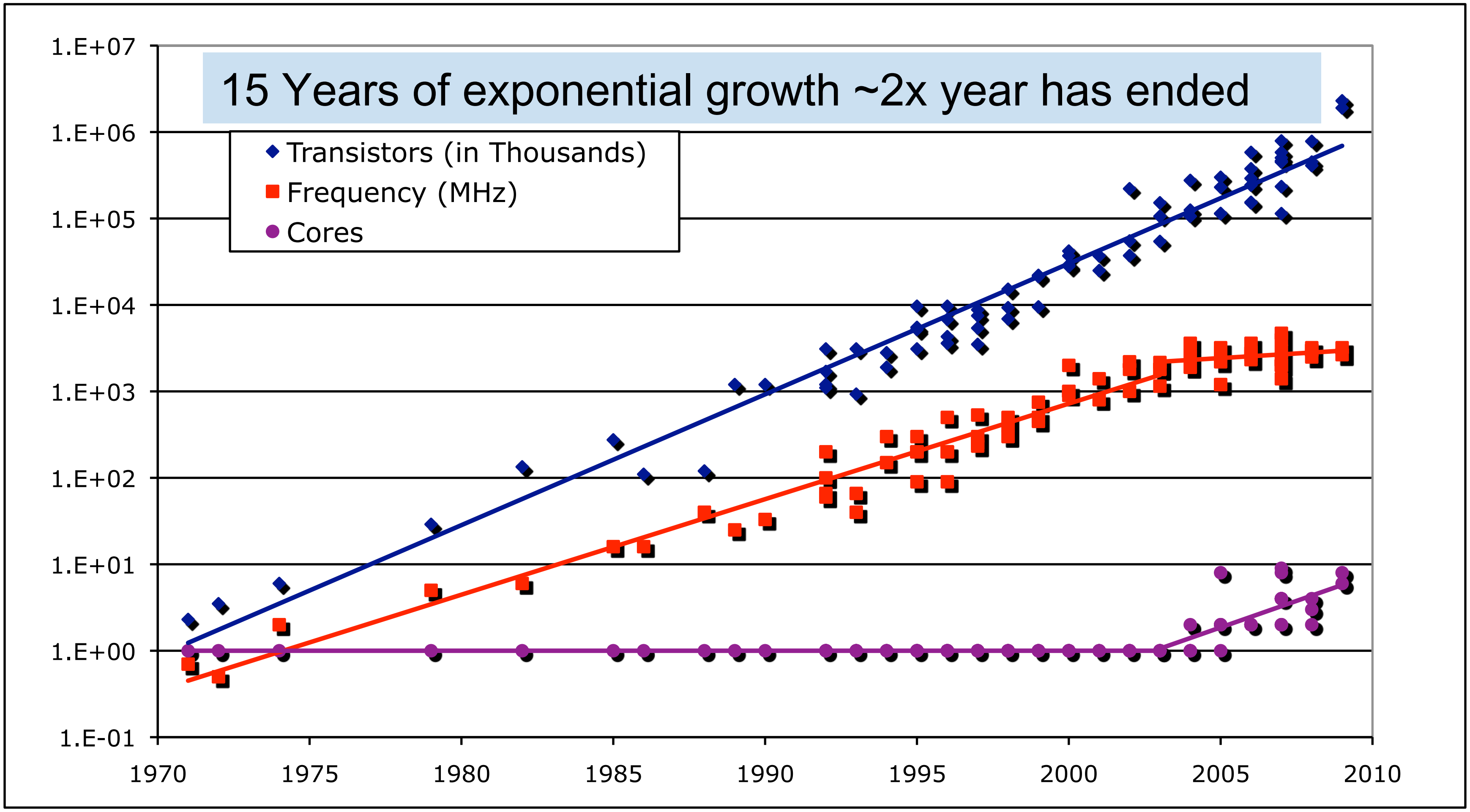
- Thank a number of people who have helped with this work
  - Emmanuel Agullo, George Bosilca, Aurelien Bouteiller, Anthony Danalis, Jim Demmel, Tingxing "Tim" Dong, Mathieu Faverge, Azzam Haidar, Thomas Herault, Mitch Horton, Jakub Kurzak, Julien Langou, Julie Langou, Pierre Lemarinier, Piotr Luszczek, Hatem Ltaief, Fengguang Song, Stanimire Tomov, Asim YarKhan, …
- Much of what I will describe has been done before, at least in theory.

Moore’s Law is Alive and Well



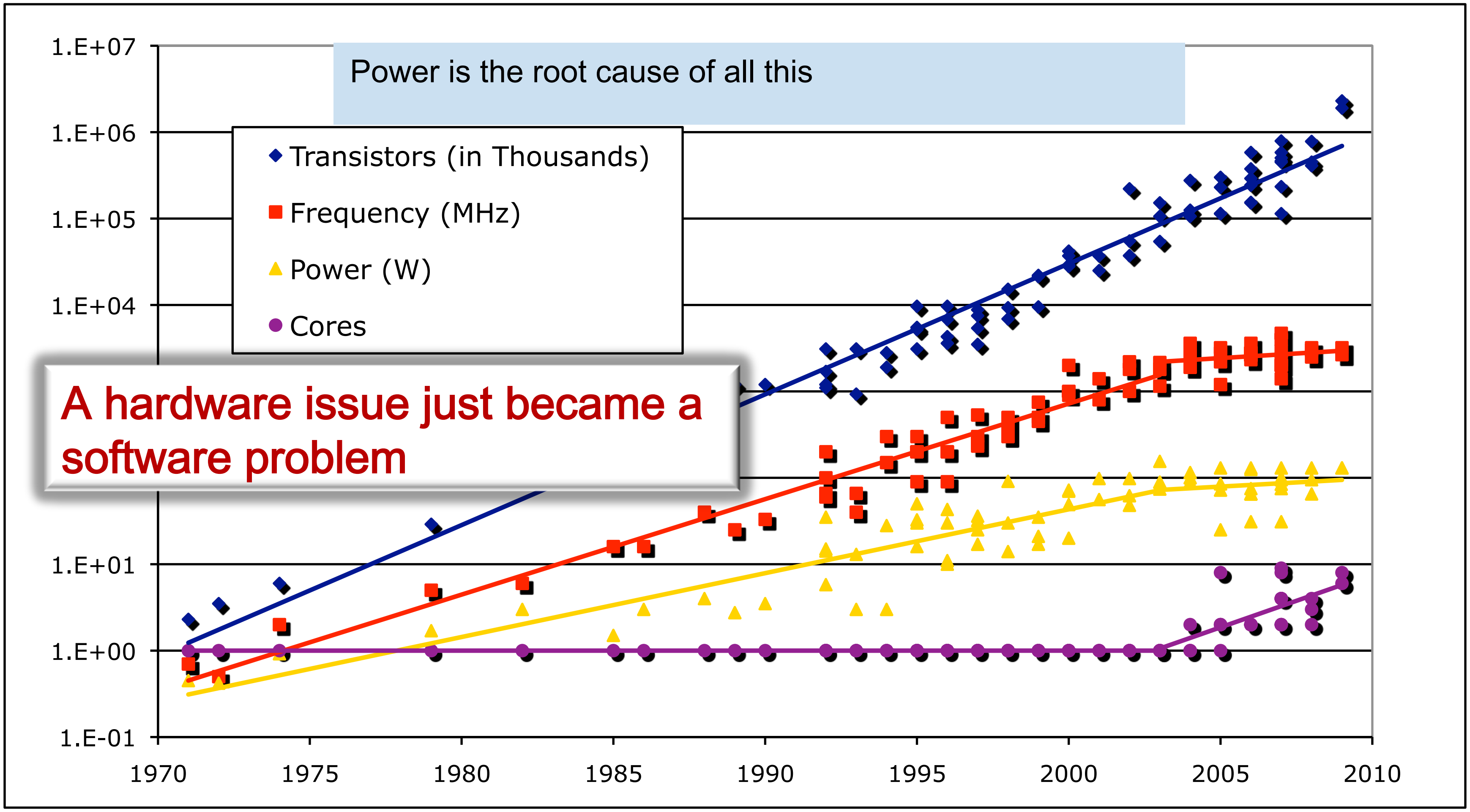
Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

But Clock Frequency Scaling Replaced by Scaling Cores / Chip



Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

Performance Has Also Slowed, Along with Power



Power is the root cause of all this

A hardware issue just became a software problem

Data from Kunle Olukotun, Lance Hammond, Herb Sutter, Burton Smith, Chris Batten, and Krste Asanoviç

Power Cost of Frequency

- Power ∝ Voltage^2 x Frequency (V^{2}F)
- Frequency ∝ Voltage
- Power ∝Frequency^3

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
"New" Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X

Power Cost of Frequency

- Power ∝ Voltage^2 x Frequency (V^{2}F)
- Frequency ∝ Voltage
- Power ∝Frequency^3

	Cores	V	Freq	Perf	Power	PE (Bops/watt)
Superscalar	1	1	1	1	1	1
"New" Superscalar	1X	1.5X	1.5X	1.5X	3.3X	0.45X
Multicore	2X	0.75X	0.75X	1.5X	0.8X	1.88X

(Bigger # is better)

50% more performance with 20% less power

Preferable to use multiple slower devices, than one superfast device

Moore’s Law reinterpreted

- Number of cores per chip will double every two years
- Clock speed will not increase (possibly decrease) because of Power

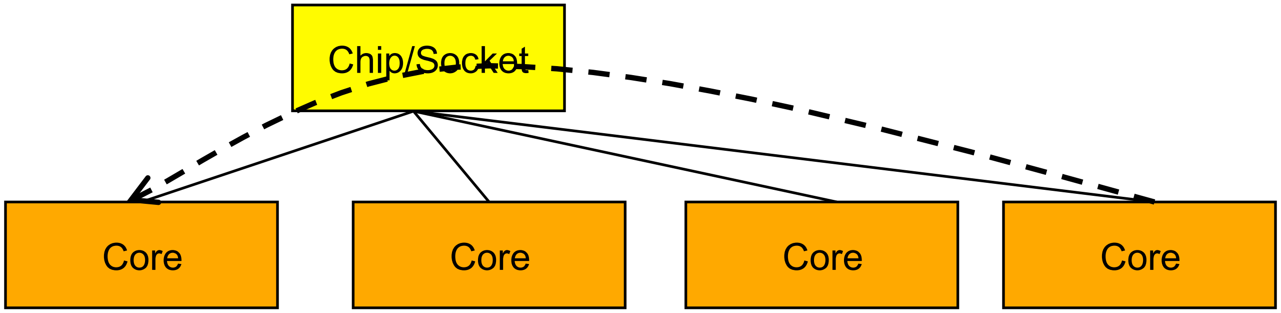
$$Power \propto Voltage^2 * Frequency$$

$$Voltage \propto Frequency$$

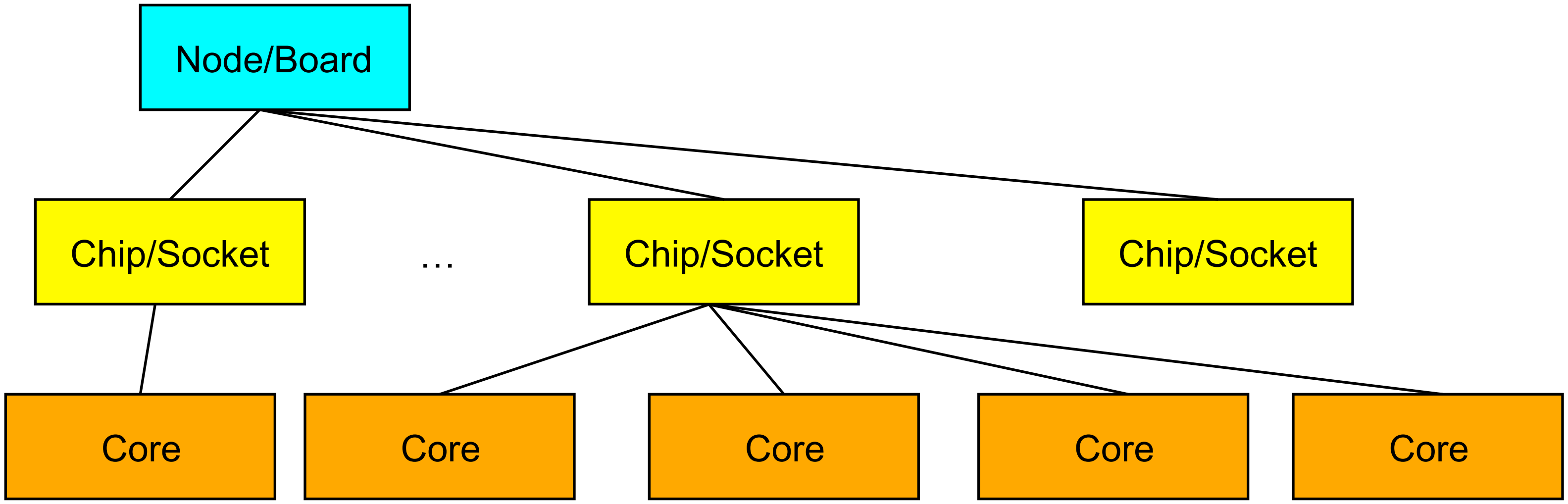
$$Power \propto Frequency^3$$

- Need to deal with systems with millions of concurrent threads
- Need to deal with inter-chip parallelism as well as intra-chip parallelism

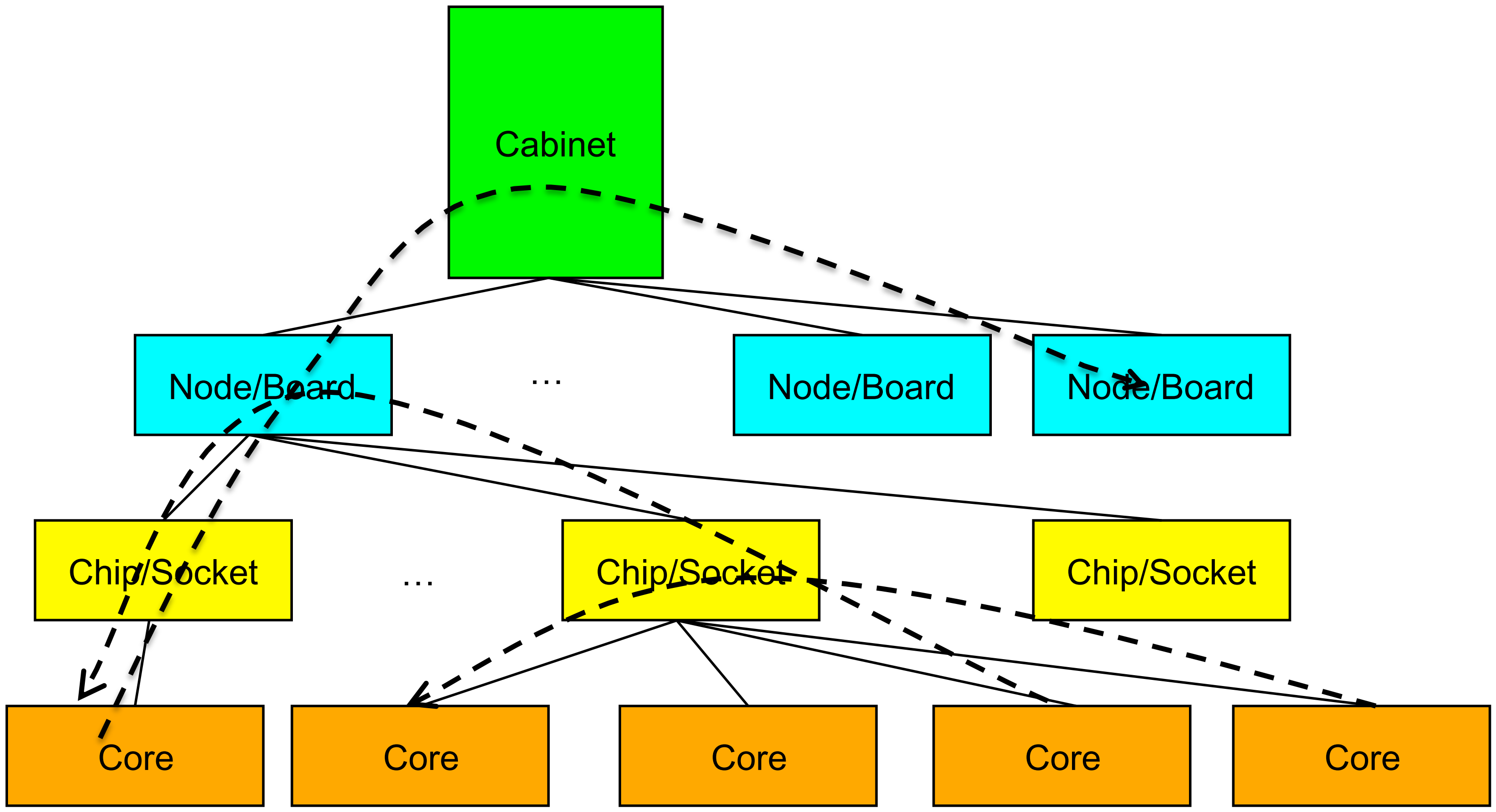
Example of typical parallel machine



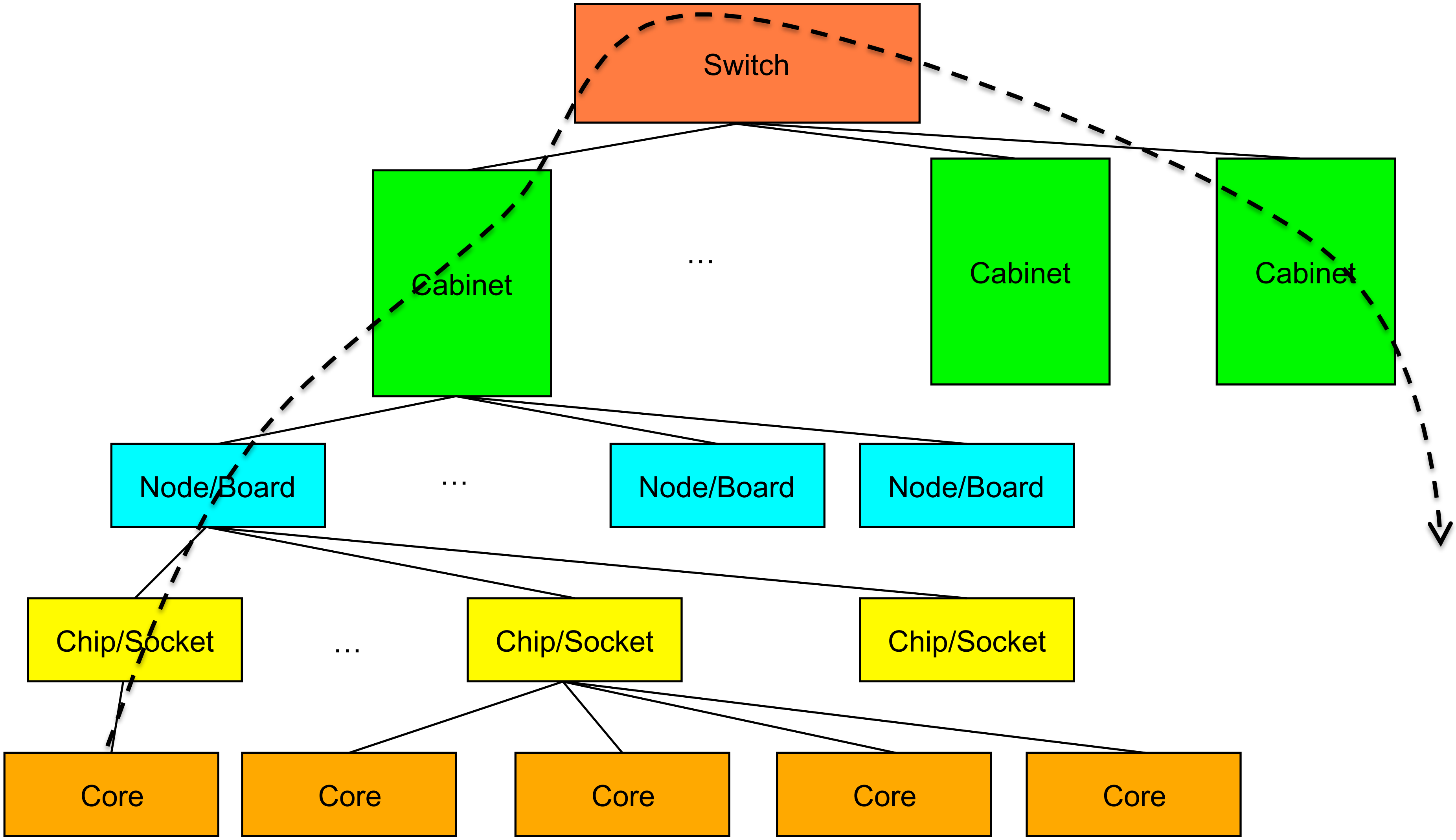
Example of typical parallel machine



Example of typical parallel machine



Example of typical parallel machine

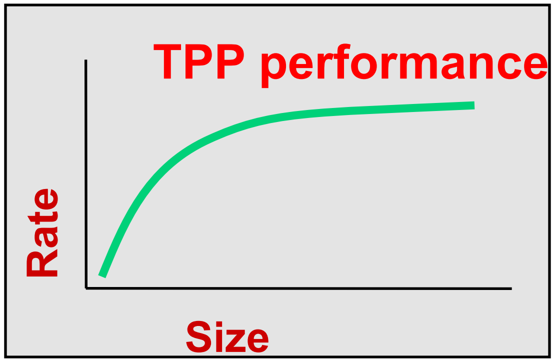


TOP500 superCOMPUTER

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*



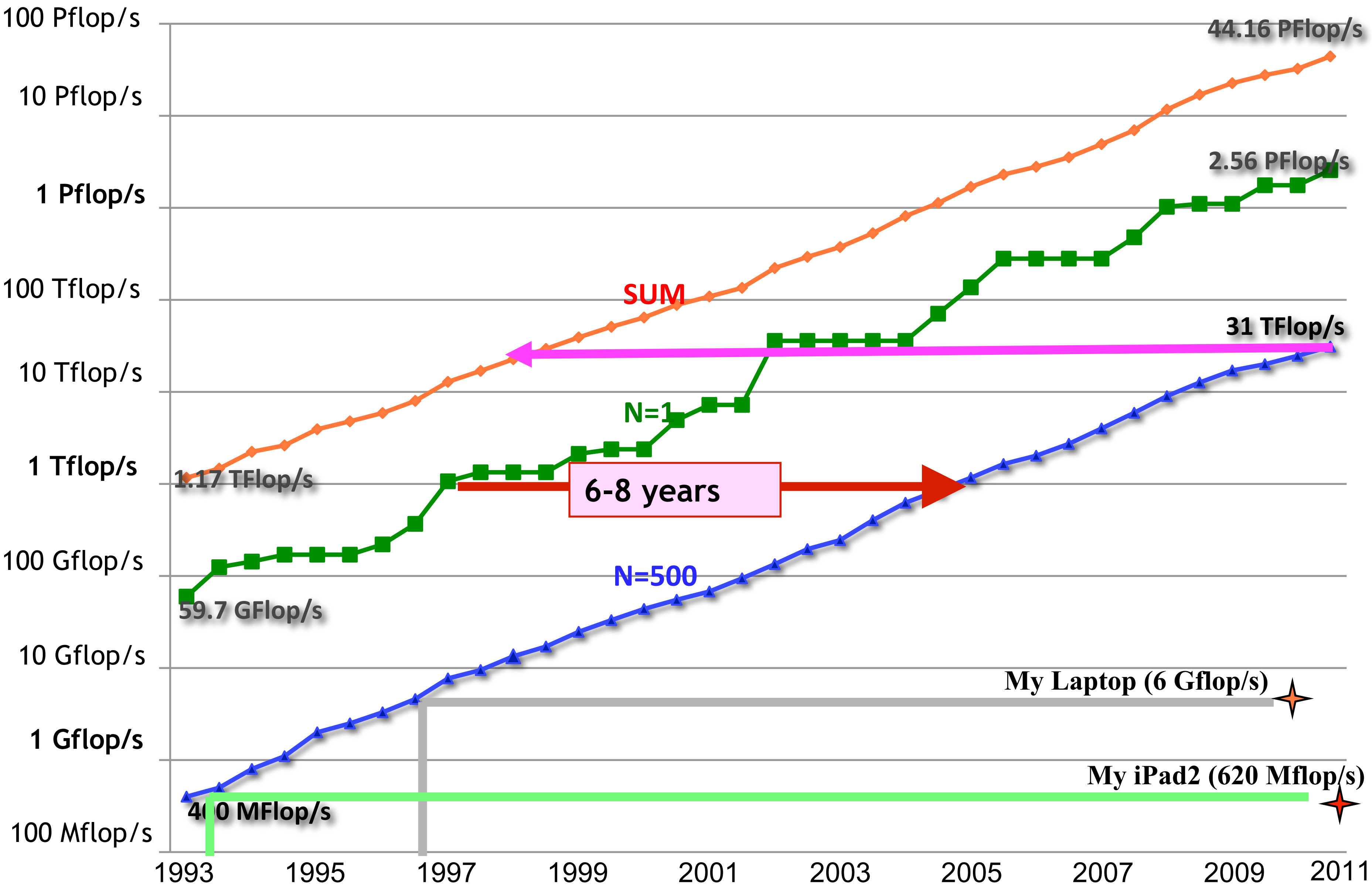
- Updated twice a year

  SC‘xy in the States in November

  Meeting in Germany in June

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

Performance Development



36rd List: The TOP10

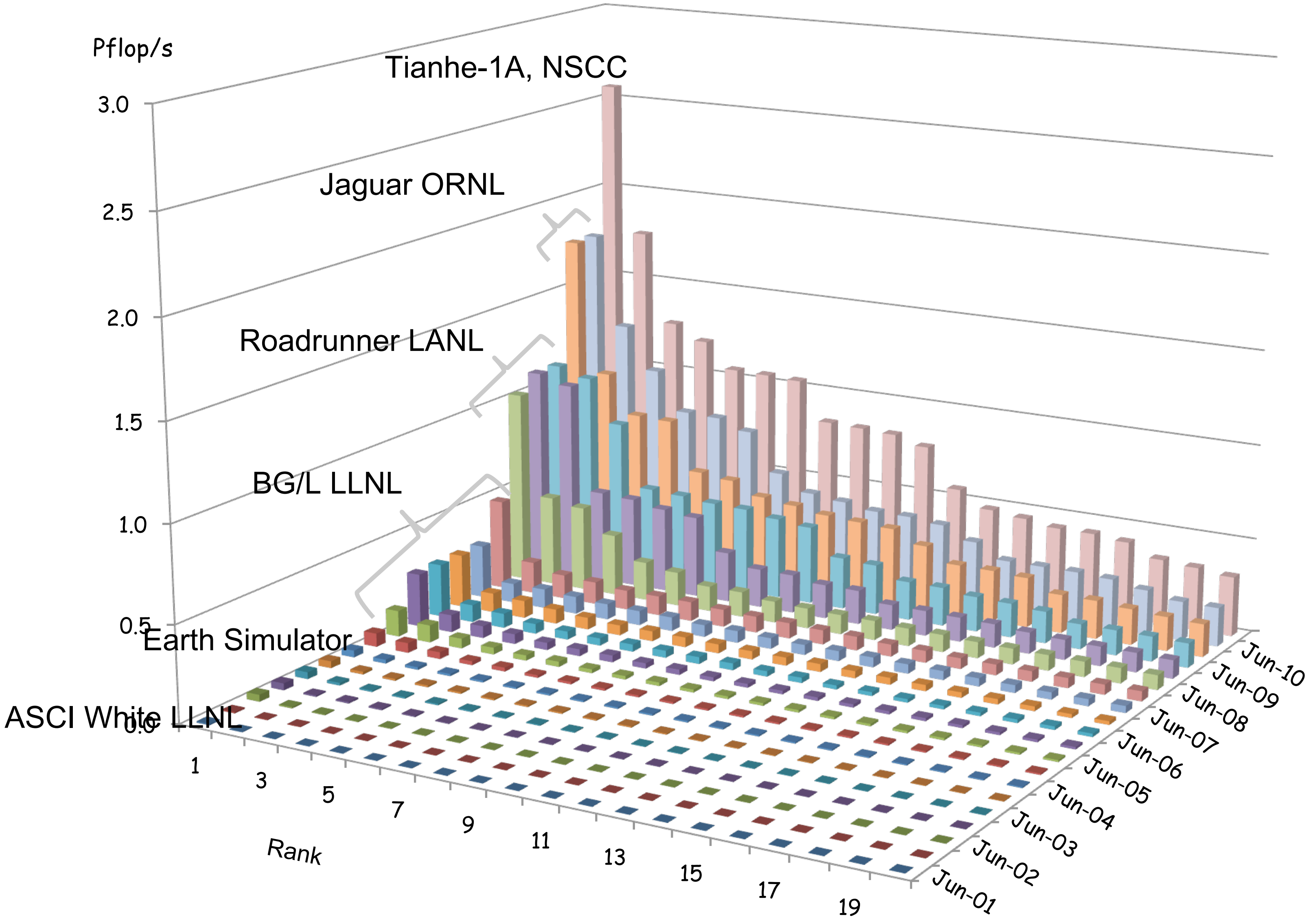
Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak
1	Nat. SuperComputer Center in Tianjin	Tianhe-1A, NUDT Intel + Nvidia GPU + custom	China	186,368	2.57	55
2	DOE / OS Oak Ridge Nat Lab	Jaguar, Cray AMD + custom	USA	224,162	1.76	75
3	Nat. Supercomputer Center in Shenzhen	Nebulea, Dawning Intel + Nvidia GPU + IB	China	120,640	1.27	43
4	GSIC Center, Tokyo Institute of Technology	Tusbame 2.0, HP Intel + Nvidia GPU + IB	Japan	73,278	1.19	52
5	DOE / OS Lawrence Berkeley Nat Lab	Hopper, Cray AMD + custom	USA	153,408	1.054	82
6	Commissariat a l'Energie Atomique (CEA)	Tera-10, Bull Intel + IB	France	138,368	1.050	84
7	DOE / NNSA Los Alamos Nat Lab	Roadrunner, IBM AMD + Cell GPU + IB	USA	122,400	1.04	76
8	NSF / NICS U of Tennessee	Kraken, Cray AMD + custom	USA	98,928	.831	81
9	Forschungszentrum Juelich (FZJ)	Jugene, IBM Blue Gene + custom	Germany	294,912	.825	82
10	DOE / NNSA LANL & SNL	Cielo, Cray AMD + custom	USA	107,152	.817	79

36rd List: The TOP10

Rank	Site	Computer	Country	Cores	Rmax [Pflops]	% of Peak	Power [MW]	GFlops/ Watt
1	Nat. SuperComputer Center in Tianjin	Tianhe-1A, NUDT Intel + Nvidia GPU + custom	China	186,368	2.57	55	4.04	636
2	DOE / OS Oak Ridge Nat Lab	Jaguar, Cray AMD + custom	USA	224,162	1.76	75	7.0	251
3	Nat. Supercomputer Center in Shenzhen	Nebulea, Dawning Intel + Nvidia GPU + IB	China	120,640	1.27	43	2.58	493
4	GSIC Center, Tokyo Institute of Technology	Tsubame 2.0, HP Intel + Nvidia GPU + IB	Japan	73,278	1.19	52	1.40	850
5	DOE / OS Lawrence Berkeley Nat Lab	Hopper, Cray AMD + custom	USA	153,408	1.054	82	2.91	362
6	Commissariat a l'Energie Atomique (CEA)	Tera-10, Bull Intel + IB	France	138,368	1.050	84	4.59	229
7	DOE / NNSA Los Alamos Nat Lab	Roadrunner, IBM AMD + Cell GPU + IB	USA	122,400	1.04	76	2.35	446
8	NSF / NICS U of Tennessee	Kraken, Cray AMD + custom	USA	98,928	.831	81	3.09	269
9	Forschungszentrum Juelich (FZJ)	Jugene, IBM Blue Gene + custom	Germany	294,912	.825	82	2.26	365
10	DOE / NNSA LANL & SNL	Cielo, Cray AMD + custom	USA	107,152	.817	79	2.95	277
500	Computacenter LTD	HP Cluster, Intel + GigE	UK	5,856	.031	53		

Quiz: How Many of the Top500 systems use GPUs?

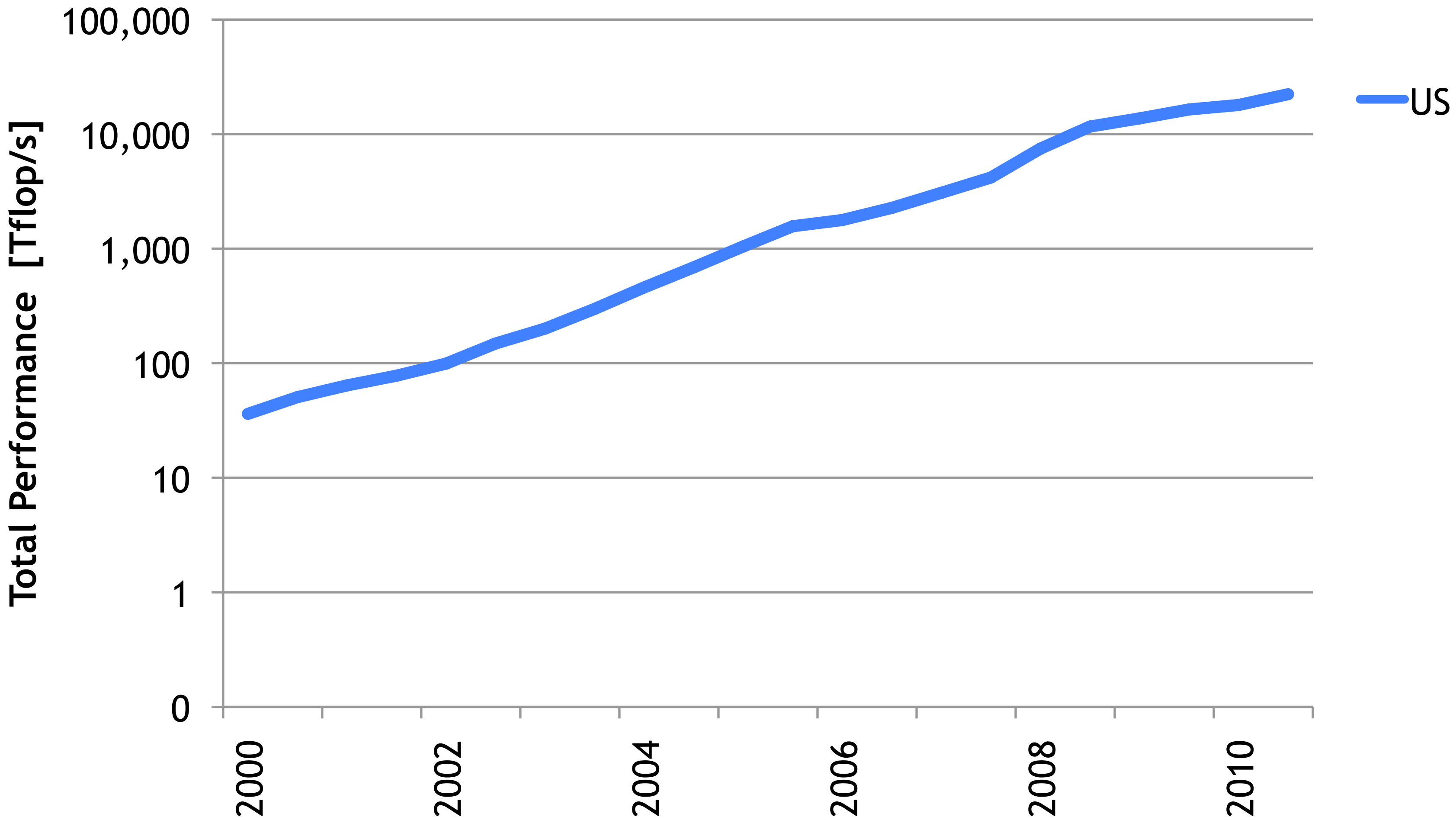
Performance of Top20 Over 10 Years



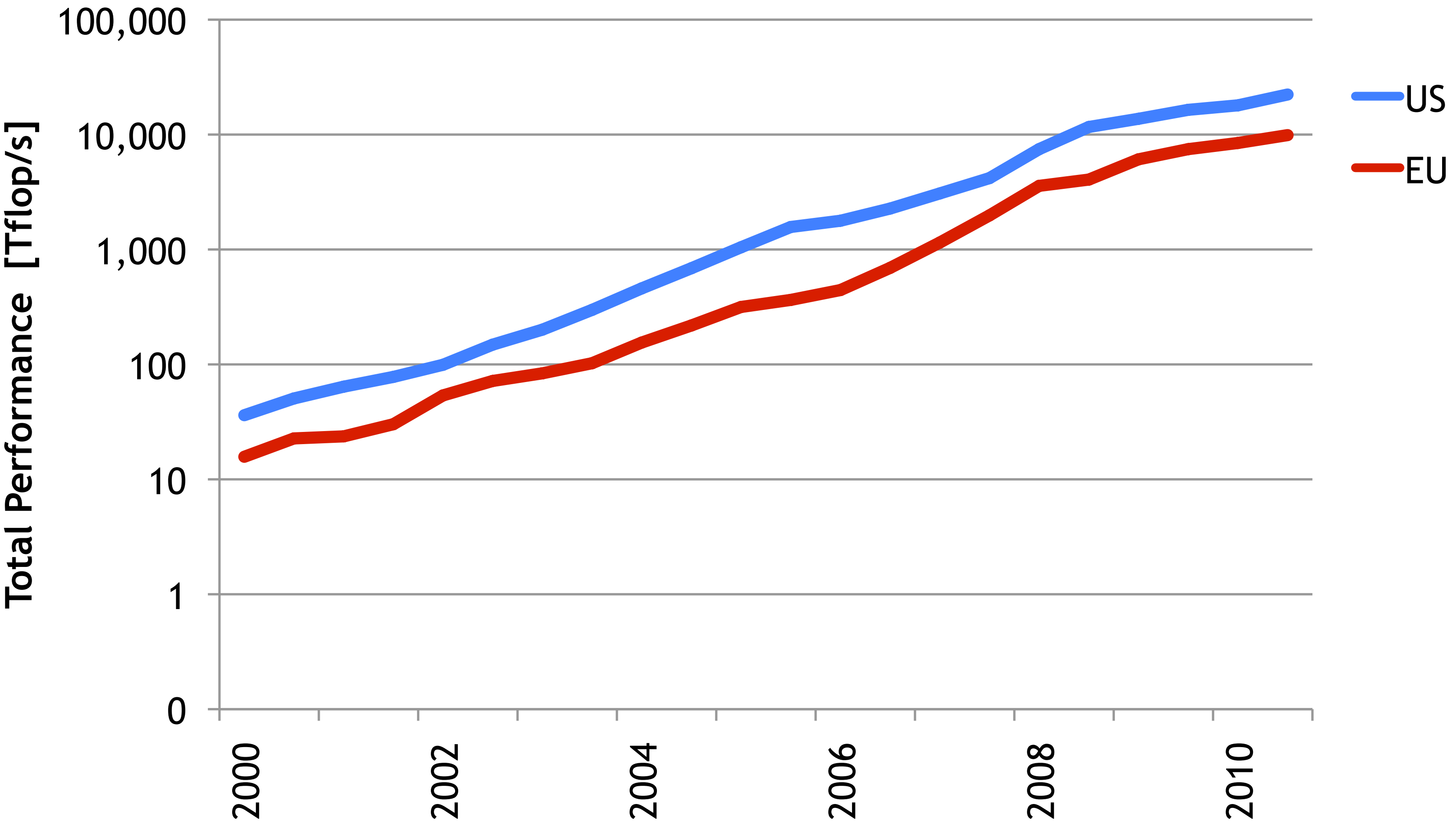
Pflop/s Club (11 systems; Peak)

Name	Peak Pflop/s	Country	
Tianhe-1A	4.70	China	NUDT: Hybrid Intel/Nvidia/Self
Nebulea	2.98	China	Dawning: Hybrid Intel/Nvidia/IB
Jaguar	2.33	US	Cray: AMD/Self
Tsubame 2.0	2.29	Japan	HP: Hybrid Intel/Nvidia/IB
RoadRunner	1.38	US	IBM: Hybrid AMD/Cell/IB
Hopper	1.29	US	Cray: AMD/Self
Tera-100	1.25	France	Bull: Intel/IB
Mole-8.5	1.14	China	CAS: Hybrid Intel/Nvidia/IB
Kraken	1.02	US	Cray: AMD/Self
Cielo	1.02	US	Cray: AMD/Self
JuGene	1.00	Germany	IBM: BG-P/Self

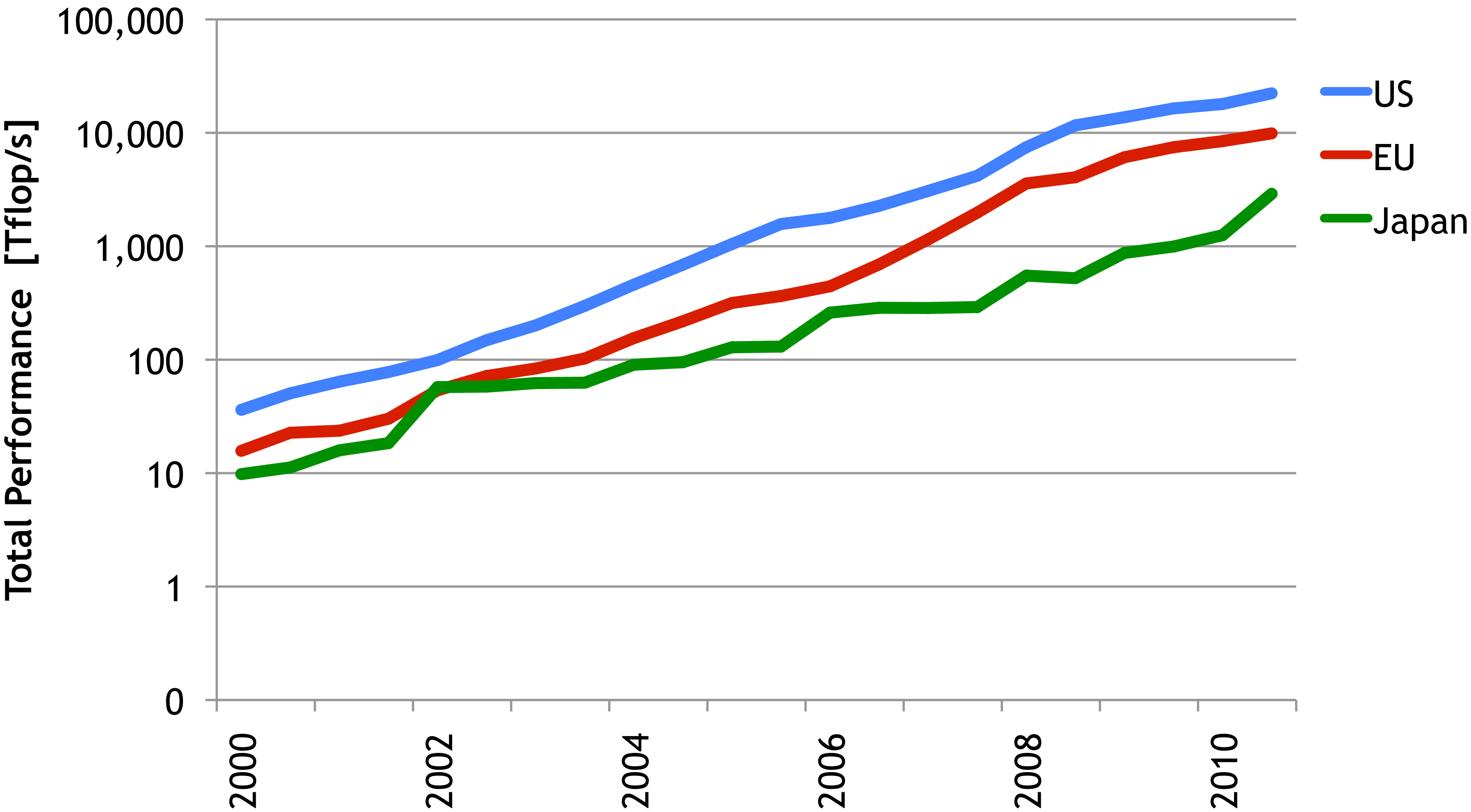
Performance of Countries



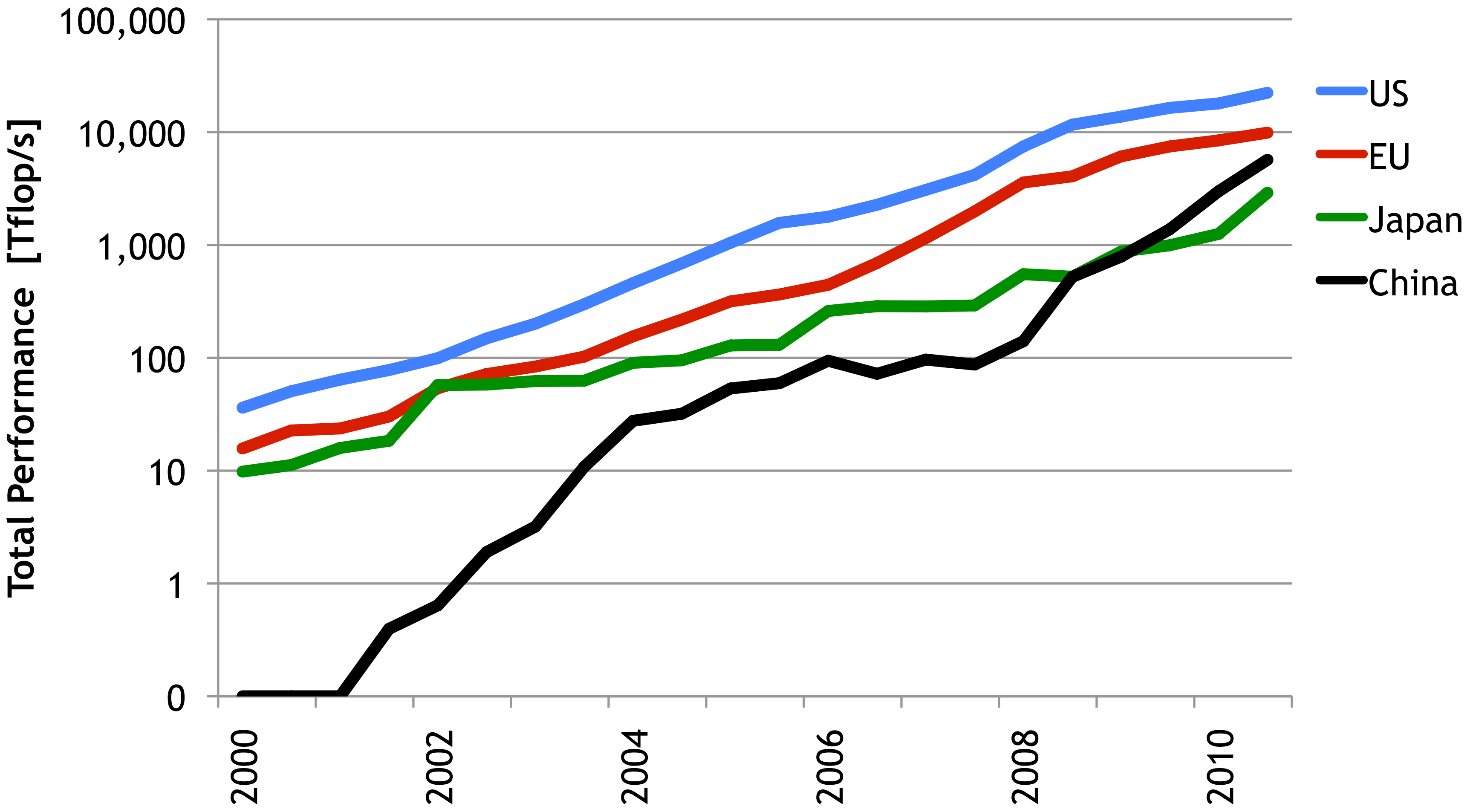
Performance of Countries



Performance of Countries



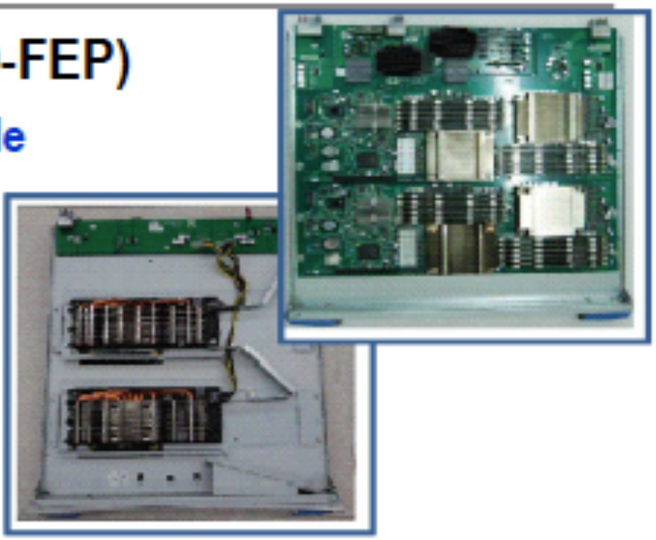
Performance of Countries



Over the last decade China has become very active in HPC
A sign not a race.

Main configuration of TH-1A system

- 7,168 compute nodes (YH-X5670-FEP)
  - 2 six-core CPU and 1 GPU per node
  - CPU: Xeon X5670 (Westmere)
    - Processor speed - 2.93GHz
  - GPU: nVIDIA M2050
    - Connected with CPU by PCI-E
  - 32GB memory per node
  - 2U height



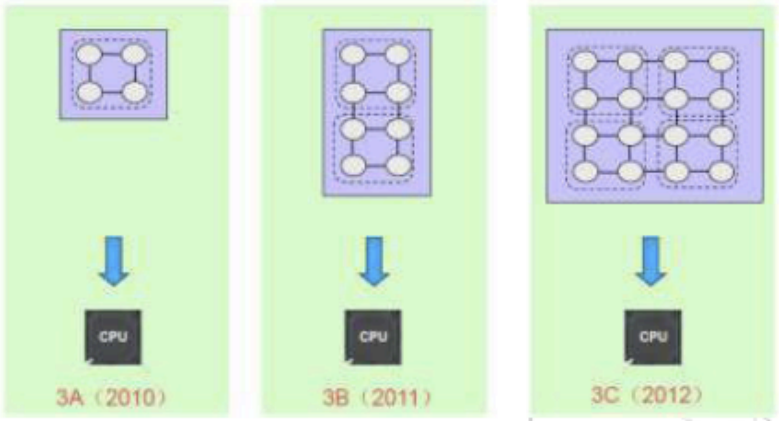
7168(nodes) × 2(CPU) ×2.93(GHz)× 6(Cores) ×4
=1.008PFlops

7168(nodes) × 1(GPU) ×1.15(GHz)*448(CUDA Cores)
=3.692PFlops

+

Total:
4,701,061 GFlops

Godson 3 /Loongson Processor



High end Roadmap: more cores on a chip

- **High end Line: multi-core CPUs for server and HPCs**

  4-core 3A (2010): 1.0GHz@65nm CMOS,10W

  Used: Dawning blade servers, personal Teraflops HPC called KD-60, a bioinformatics-oriented system called SuperDragon-I,

  8-core 3B (2011): 1.0GHz@65nm, 128GFLOPS@40W

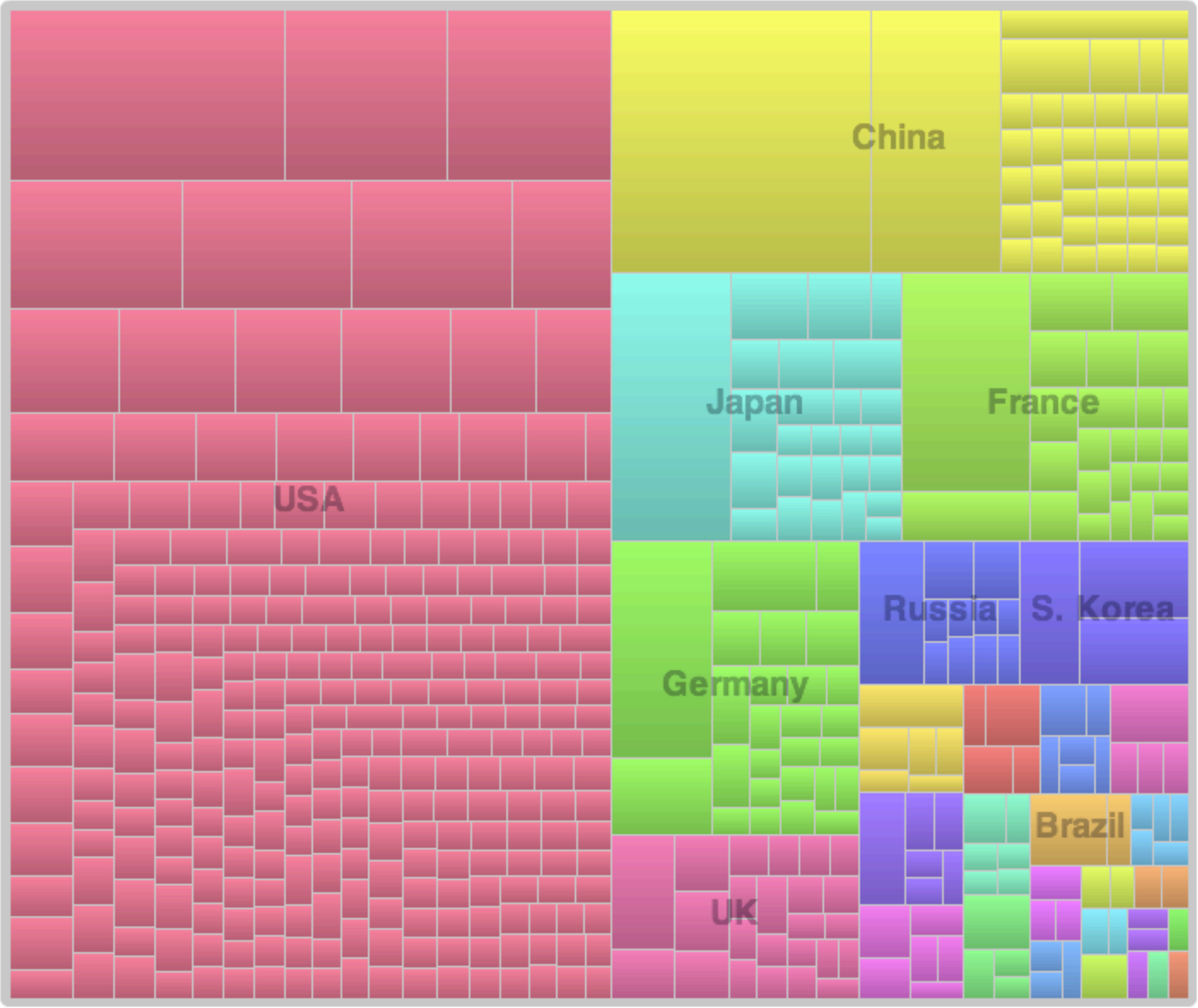
  To be used: Petaflops Dawning 6000, personal Teraflops HPC called KD-50-III

  16-core 3C (2013): 1.5-2.0GHz@28nm, 384-512GFLOPS@20W 4DDR2 4 HT controllers

  (to be taped out 2011)

  Planned to be used: 10 Petaflops system

Countries Share



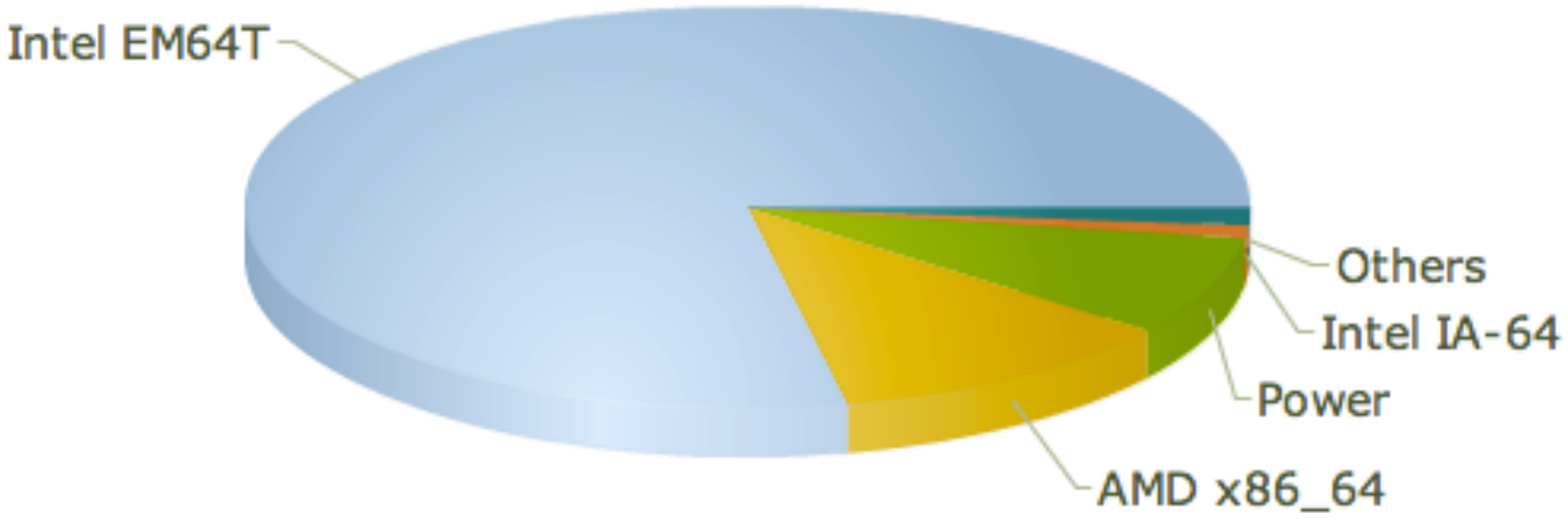
Absolute Counts

US:	274
China:	41
Germany:	26
Japan:	26
France:	26
UK:	25

Systems in France

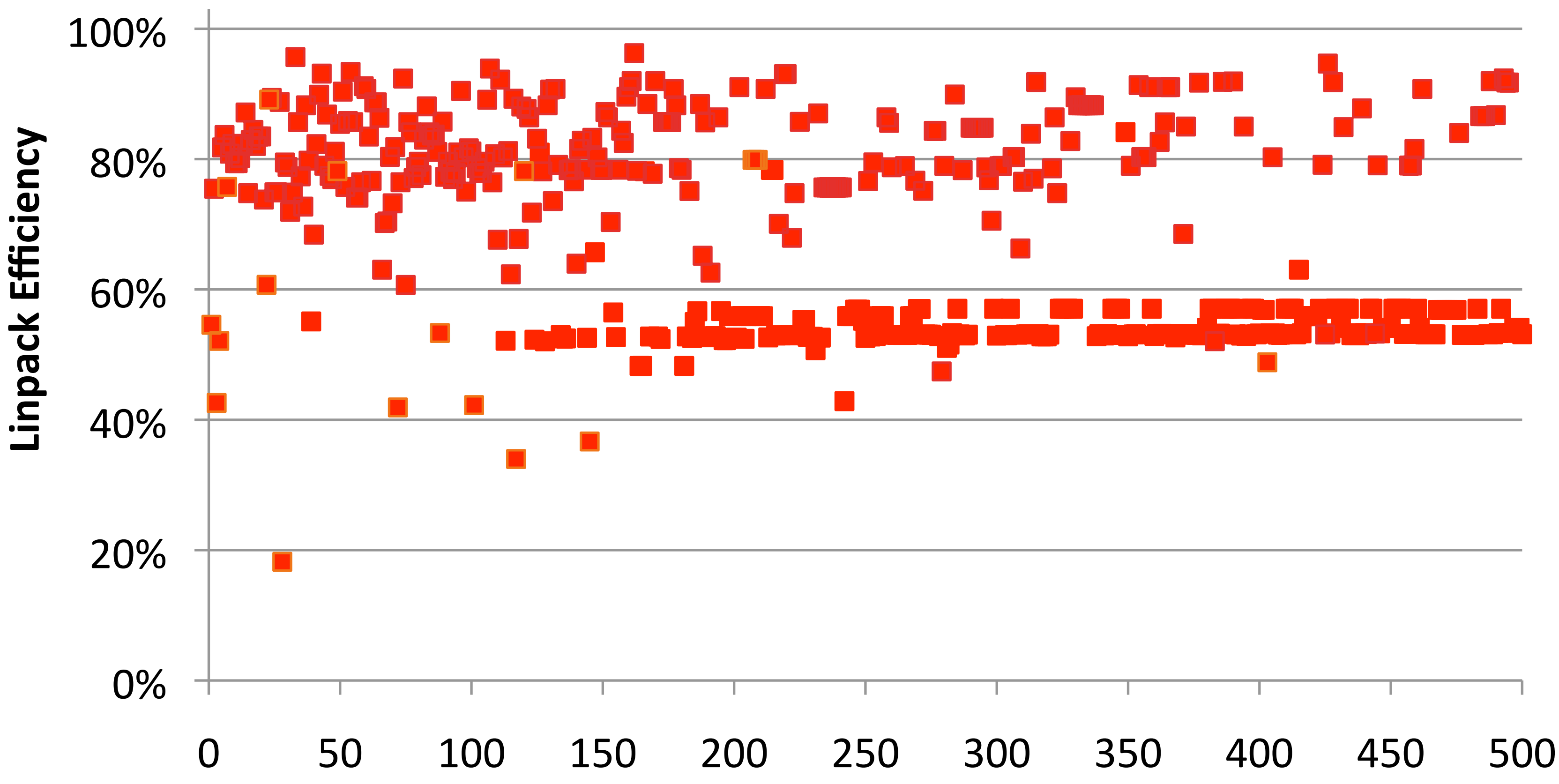
Rank	Site	Manufacturer	Computer	Cores	RMax
6	CEA	Bull SA	Bull bullx super-node S6010/S6030	138368	1050000
27	GENCI-CINES	SGI	SGI Altix ICE 8200EX, Xeon E5472 3.0/X5560 2.8 GHz	23040	237800
36	Government	HP	Cluster Platform 3000 BL2x220, L54xx 2.5 Ghz, Infiniband	24704	179634
37	EDF R&D	IBM	iDataPlex, Xeon X56xx 6C 2.93 GHz, Infiniband	16320	168800
55	IDRIS	IBM	Blue Gene/P Solution	40960	119310
61	CEA	Bull SA	BULL Novascale R422-E2	11520	108500
65	Total Exploration	SGI	SGI Altix ICE 8200EX, Xeon quad core 3.0 GHz	10240	106100
76	EDF R&D	IBM	Blue Gene/P Solution	32768	95450
83	Manufacturing	HP	Cluster Platform 3000 BL460c G6, Xeon X5570 2.93 GHz, Infiniband	8576	88550
85	Bull	Bull SA	Bull bullx super-node S6010/S6030	11520	87470
86	CEA	Bull SA	Bullx S6010 Cluster, Xeon 2.26 Ghz 8-core, QDR Infiniband	11520	87470
142	CEA	Bull SA	NovaScale 5160, Itanium2 1.6 GHz, Quadrics	9968	52840
143	IDRIS	IBM	Power 575, p6 4.7 GHz, Infiniband	3584	52810
225	CEA	Bull SA	Novascale 3045, Itanium2 1.6 GHz, Infiniband	7680	42130
233	Financial Institute	HP	Cluster Platform 3000 BL460c G1, Xeon L5410 2.33 GHz, GigE	8432	41347
340	Food Industry	HP	Cluster Platform 3000 BL460c G1, Xeon 5430 2.66 GHz, GigE	6400	36105
401	Financial Institution	IBM	iDataPlex, Xeon E55xx QC 2.26 GHz, GigEthernet	6528	33660
402	Financial Institution	IBM	iDataPlex, Xeon E55xx QC 2.26 GHz, GigEthernet	6528	33660
416	Financial Institution	HP	Cluster Platform 3000 BL460c G1, Xeon E5450 3.0 GHz, GigE	5120	32733
426	ONERA	SGI	SGI Altix ICE 8200EX, Xeon Nehalem quad core 2.8 GHz	3072	32570
455	Financial Institution	HP	Cluster Platform 3000 BL260c G5, Xeon QC 2.66Ghz, GigE	5632	31864
476	Financial Institution	HP	Cluster Platform 3000 BL460c G6, Xeon X5570 2.93 GHz, Infiniband	3200	31507
492	Information Service	IBM	BladeCenter HS22 Cluster, Xeon QC X55xx 2.53 GHz, GigEthernet	5408	31217
497	Communications	IBM	xSeries x3650M2 Cluster Xeon QC GT 2.66 GHz, GigEthernet	5392	31124
498	Communications	IBM	xSeries x3650M2 Cluster Xeon QC GT 2.66 GHz, GigEthernet	5392	31124
499	Communications	IBM	xSeries x3650M2 Cluster Xeon QC GT 2.66 GHz, GigEthernet	5392	31124

Processors Used in the Top500 Systems

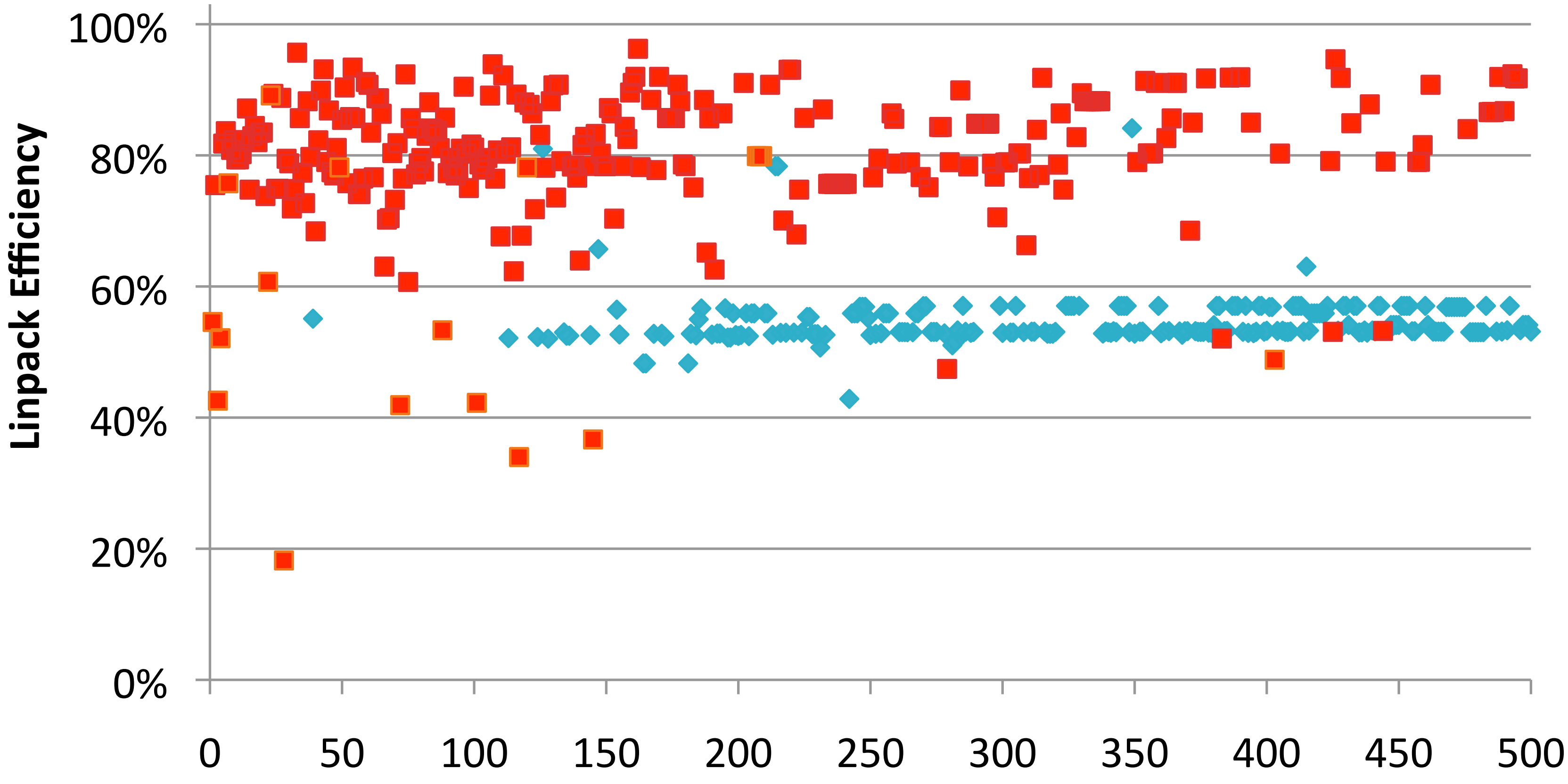


Intel 81% (406)
AMD 11% (57)
IBM 8% (40)

Linpack Efficiency



Linpack Efficiency

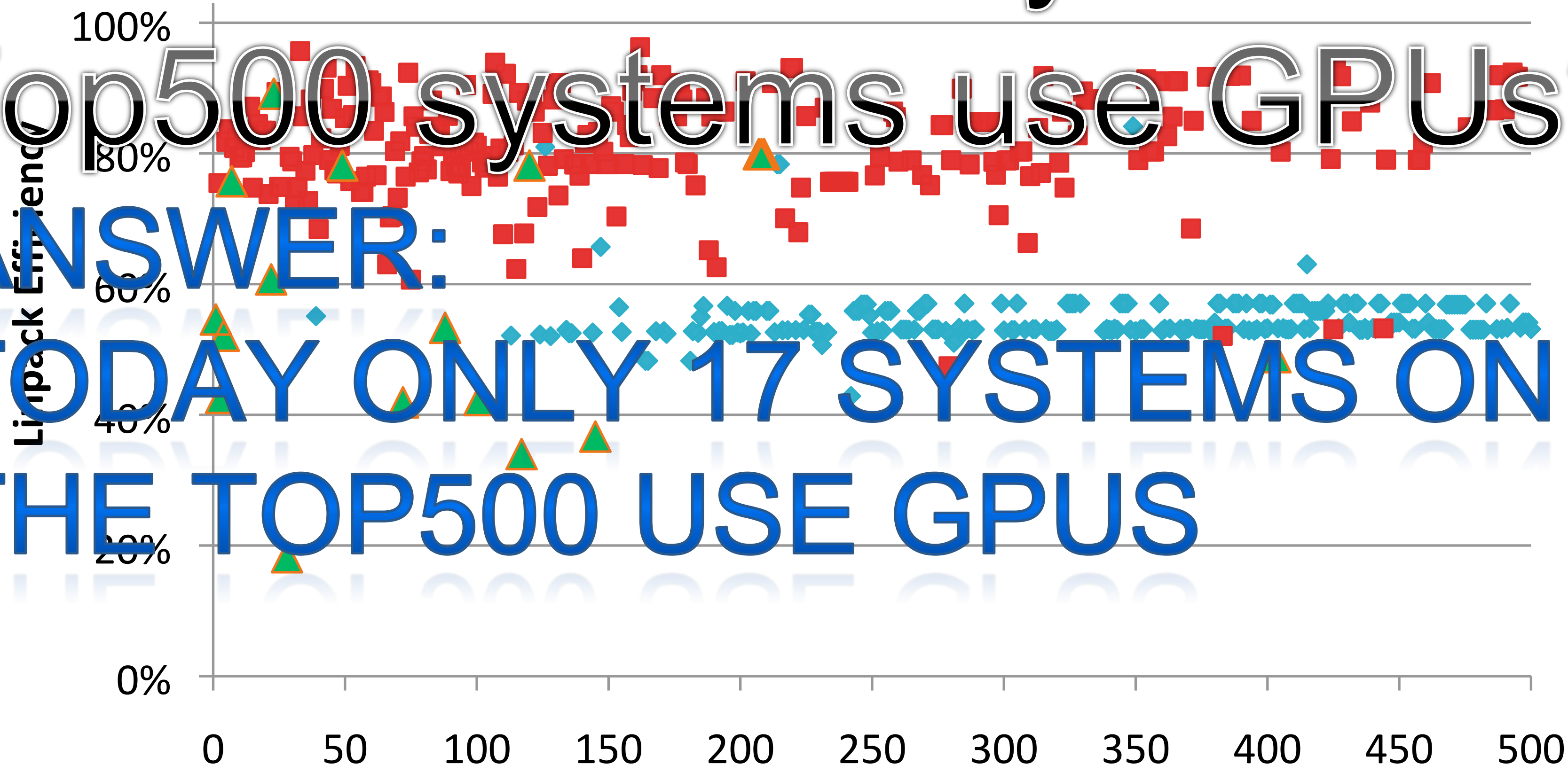


Linpack Efficiency

Quiz: How Many of the Top500 systems use GPUs?

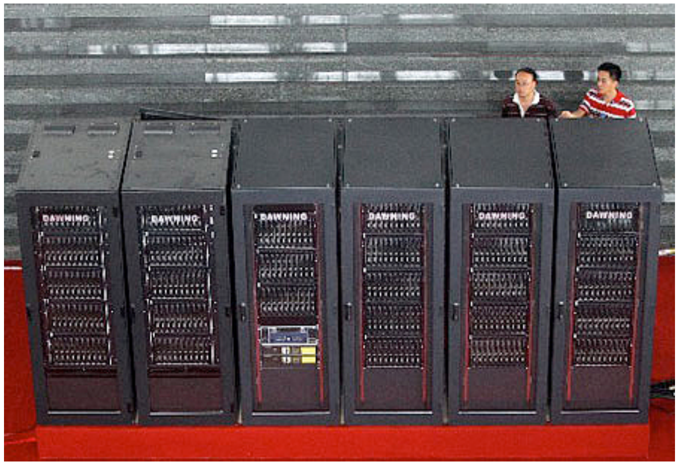
ANSWER:

TODAY ONLY 17 SYSTEMS ON THE TOP500 USE GPUS



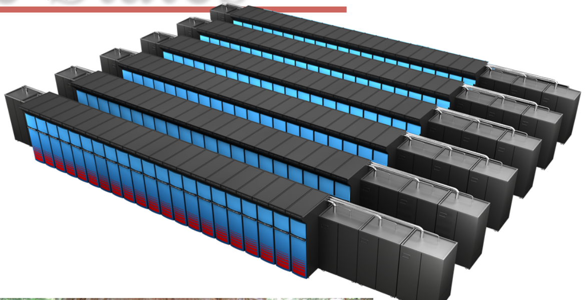
China

- Has 3 Pflops systems
  - #1, NUDT, Tianhe-1A, located in Tianjin
    Dual-Intel 6 core + Nvidia Fermi w/custom interconnect
    - Budget 600M RMB
      - MOST 200M RMB, Tianjin Government 400M RMB
  - #3, CIT, Dawning 6000, Nebulea, located in Shenzhen
    Dual-Intel 6 core + Nvidia Fermi w/QDR Ifiniband
    - Budget 600M RMB
      - MOST 200M RMB, Shenzhen Government 400M RMB
  - #28, CAS IPE, Mole-8.5 Cluster/320x2 Intel QC Xeon E5520 2.26 Ghz + 320x6 Nvidia Tesla C2050/QDR Infiniband
  - #35, Shanghai SCC, **Dawning 5000A, QC Opteron 1.9 Ghz, Infiniband**
- Others planned for Shandong and NUDT



10+ Pflop/s Systems Planned in the States

- DOE SC, Titan at ORNL,
  - Based on Cray design with accelerators, 20 Pflop/s
- DOE NNSA, Sequoia at Lawrence Livermore Nat. Lab,
  - Based on IBM's BG/Q, 20 Pflop/s
- DOE SC, BG/Q at Argonne National Lab,
  - Based on IBM's BG/Q, 10 Pflop/s
- NSF, Blue Waters at University of Illinois UC,
  - Based on IBM's Power 7 Proc, 10 Pflop/s



Japanese 10 PF Facility @ Kobe, Japan

Construction: started in March, 2008 and complete in May, 2010,
Machine operation ~ early 2012

Computer Wing

Total Floor Area:17,500m^2

2 Computer rooms: 12,600m^2

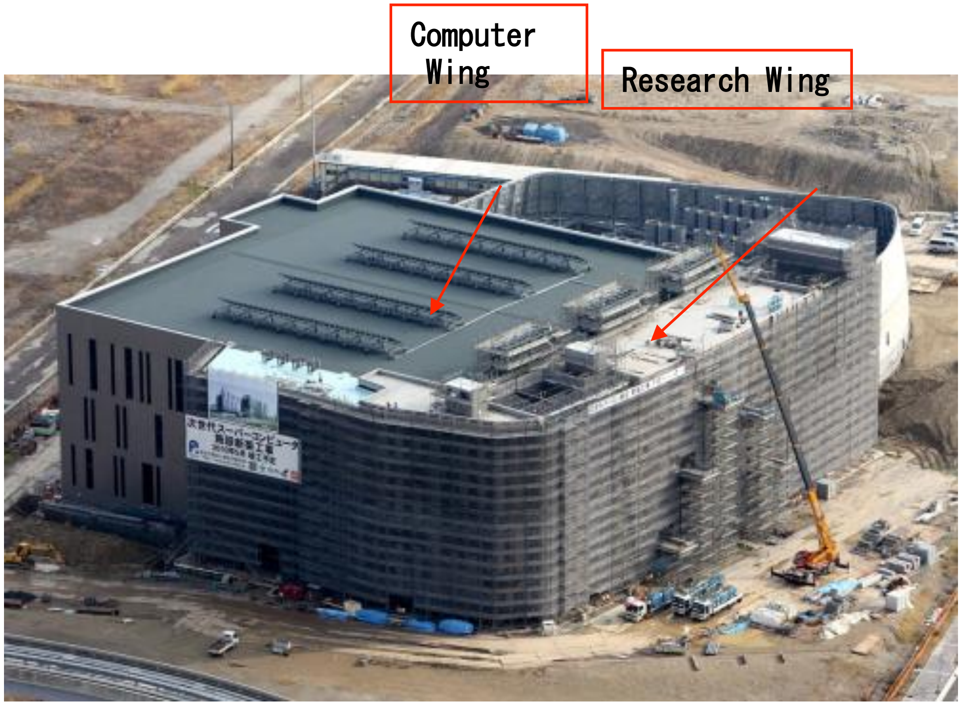
4 Floors (1 underground floor)

Other Facilities

Co-generation System

Water chiller system

Electric Subsystem

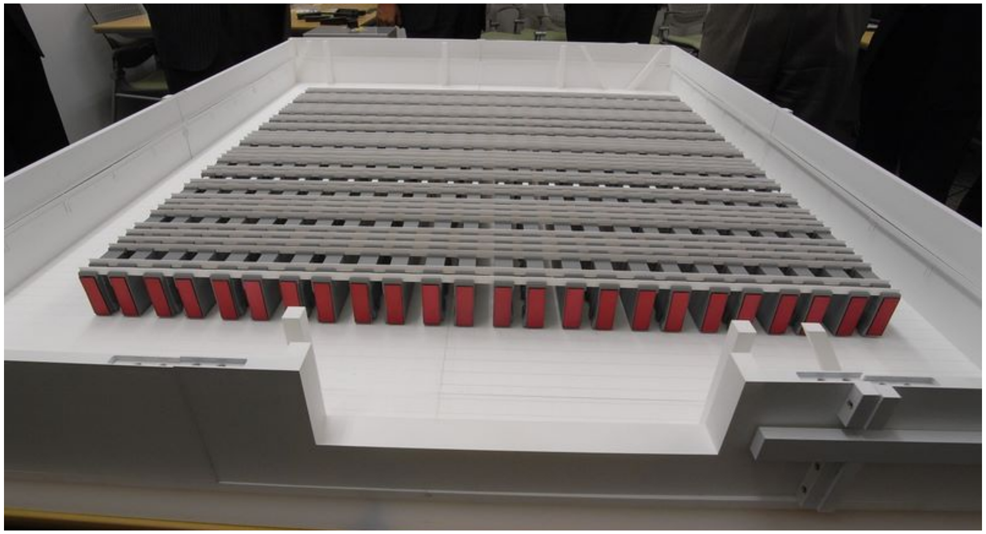


Initially 30MW capability@2011

K computer Delivery Began in Late September

- The first eight racks of the K computer were delivered to Kobe from Fujitsu on September 28, 2010. More than 800 racks are required for a 10 Pflop/s Performance.
- A computer rack weighs about 1,300 kg in average. The rack contains 96 water -cooled Fujitsu SPARC64 VIIIfx CPU chips, each of which performs 128 Gflop/s, interconnected with the 3D Torus network that Fujitsu named Tofu.

Photo of First delivery, Sep 28, 2010

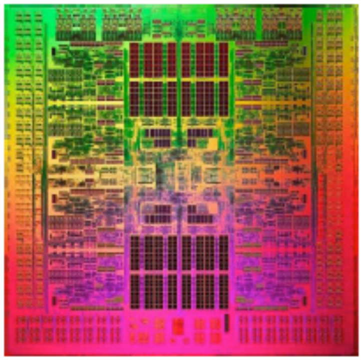


15-17MW

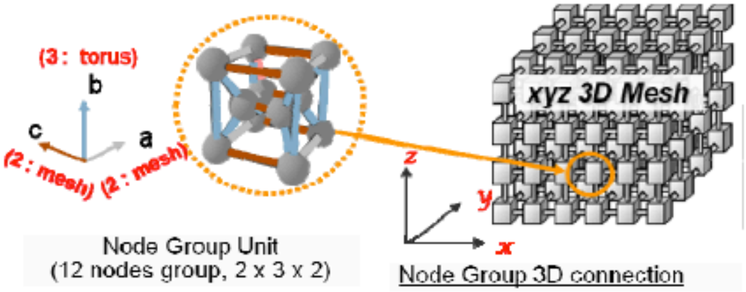
Hardware Technologies

- SPARC64^{TM} VIIIfx: New HPC-enhanced CPU based on SPARCV9 architecture
  - 8 cores, 2 GHz, 2.2 GFLOP/s per watt
  - Extended HPC-ACE instruction set
    - Reciprocal, trig functions, max/min, SIMD, masked SIMD
  - Enlarged number of registers (Floating:256, Int:64)
  - User-controllable sector cache

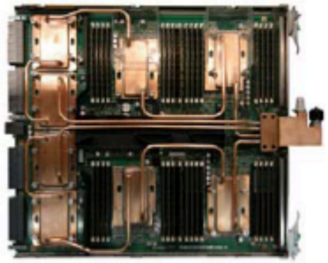
45 nm



- Tofu: 6D mesh/torus interconnect
  - Fast node to node communication, 5 GB/s (bi-directional)
  - Integrated MPI support for collective operations and global hardware barrier
  - Scalable to 100,000s of nodes



- Direct water cooling packaging



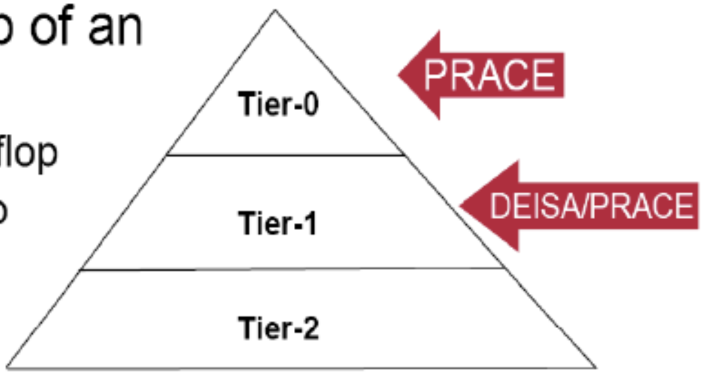
Direct water cooling System Board

PRACE Systems

- First PRACE System:
  - JUGENE, IBM BlueGene/P, Gauss Center for Supercomputing hosted at Forschungzentrum Jülich (FZJ), Jülich, Germany, peak 1 Pflop/s
- Second PRACE System
  - CURIE funded by GENCI and hosted at CEA, France, is designed and built by BULL. CURIE is based on x86 architecture CPUs with a mix of thin and fat nodes interconnected through a QDR Infiniband interconnect. Available now and the total peak performance will be 1.6 Pflop/s.
- Third PRACE System
  - HLRS based on Cray XE6, > 5 Pflop/s Mid 2012

PRACE Systems

- European HPC-facilities at the top of an HPC provisioning pyramid
  - Tier-0: 3-6 European Centres for Petaflop
  - Tier-0: ? European Centres for Exaflop
  - Tier-1: National Centres
  - Tier-2: Regional/University Centres



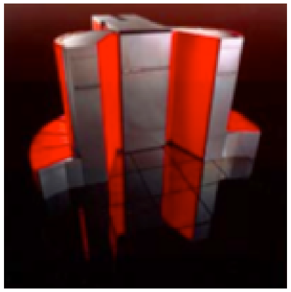
- 1st PRACE System
  - BG/P by Gauss Center for Supercomputing at Juelich
    - 294912 CPU cores, 144 TB memory
    - 1 PFlop/s peak performance
    - 825.5 TFlop/s Linpack
    - 600 I/O nodes (10GigE) > 60 GB/s I/O
    - 2.2 MW power consumption
    - 35% for PRACE
- 2nd PRACE System (2010/2011) - GENCI
  - Intel based system by Bull
    - ~90,000 cores
    - 1.6 PFlop/s peak performance
- 3rd PRACE System 2011 HLRS
- 4th PRACE System 2012 LRZ
- 5th and 6th Systems 2012/2013 – Cineca and BSC

Russia

- T-Platforms will supply Moscow State university with a 1.3 Pflop/s system
  - Roughly $25M
- First stage in place, 510 Tflop/s system and second stage adds 800 Tflop/s = 1.31 Pflop/s
- TB2-TLTM hybrid blade system from T-Platforms equipped with NVIDIA's TeslaTM X2070 GPUs.
  - 32 - Intel Westmere processor 2.13 GHz, 4 core
  - 32 Nvidia Fermi GPUs
  - Infiniband interconnect
  - 32*4*4*2.13 + 32*515 = 17.571 Tflop/s per blade
  - 6 blades per rack = 105.4 Tflop/rack
  - 12 racks system = 1.265 Pflop/s
  - QDR Infiniband
  - 14 PB memory

Looking at the Gordon Bell Prize

(Recognize outstanding achievement in high-performance computing applications and encourage development of parallel processing)

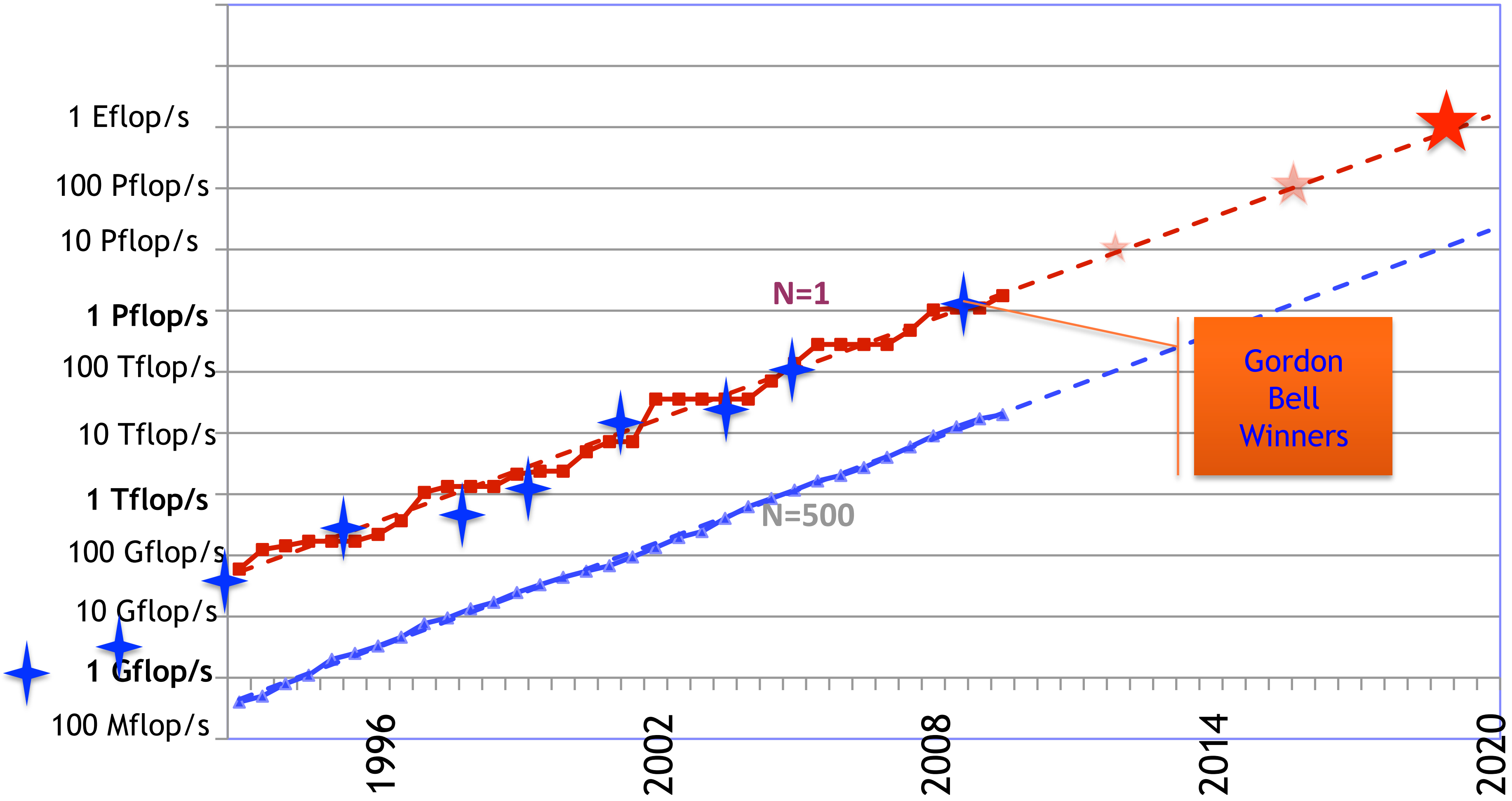


- 1 GFlop/s; 1988; Cray Y-MP; 8 Processors
  - Static finite element analysis
- 1 TFlop/s; 1998; Cray T3E; 1024 Processors
  - Modeling of metallic magnet atoms, using a variation of the locally self-consistent multiple scattering method.

- 1 PFlop/s; 2008; Cray XT5; 1.5x10^5 Processors
  - Superconductive materials

- 1 EFlop/s; ~2018; ?; 1x10^7 Processors (10^9 threads)

Performance Development in Top500



Exascale Applications and Technology

- Town Hall Meetings April-June 2007
- Scientific Grand Challenges Workshops November 2008 – October 2009
  - Climate Science (11/08),
  - High Energy Physics (12/08),
  - Nuclear Physics (1/09),
  - Fusion Energy (3/09),
  - Nuclear Energy (5/09),
  - Biology (8/09),
  - Material Science and Chemistry (8/09),
  - National Security (10/09) (with NNSA)
- Cross-cutting workshops
  - Architecture and Technology (12/09)
  - Architecture, Applied Math and CS (2/10)
- Meetings with industry (8/09, 11/09)
- External Panels
  - ASCAC Exascale Charge (FACA)
  - Trivelpiece Panel

Scientific Grand Challenges

FOREFRONT QUESTIONS IN NUCLEAR SCIENCE AND THE ROLE OF COMPUTING AT THE EXTREME SCALE

· Washington, D.C.

U.S. DEPARTMENT OF ENERGY

Scientific Grand Challenges

CHALLENGES FOR UNDERSTANDING THE QUANTUM UNIVERSE AND THE ROLE OF COMPUTING AT THE EXTREME SCALE

Menlo Park, CA

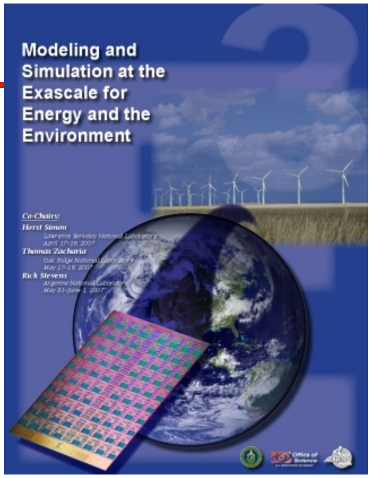
DEPARTMENT OF ENERGY

Scientific Grand Challenges

CHALLENGES IN CLIMATE CHANGE SCIENCE AND THE ROLE OF COMPUTING AT THE EXTREME SCALE

November 6-7, 2008 · Washington D.C.

U.S. DEPARTMENT OF ENERGY



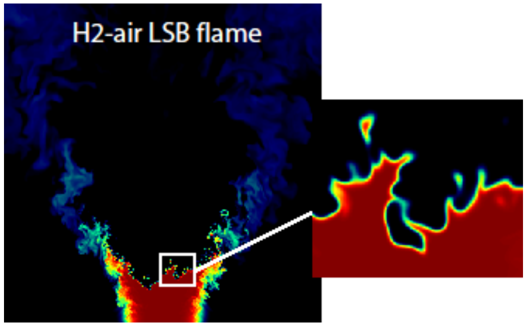
Modeling and Simulation at the Exascale for Energy and the Environment

Co-Chairs:

Horst Simon
Lawrence Berkeley National Laboratory
April 17-18, 2007

Thomas Zacharia
Oak Ridge National Laboratory
May 17-18, 2007

Rick Stevens
Argonne National Laboratory
May 31-June 1, 2007



> "The key finding of the Panel is that there are compelling needs for exascale computing capability to support the DOE's missions in energy, national security, fundamental sciences, and the environment. The DOE has the necessary assets to initiate a program that would accelerate the development of such capability to meet its own needs and by so doing benefit other national interests. Failure to initiate an exascale program could lead to a loss of U. S. competitiveness in several critical technologies."
>
> Trivelpiece Panel Report, January, 2010

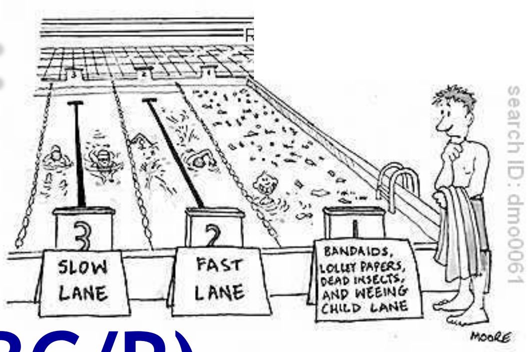
Potential System Architecture

Systems	2011
System peak	**2 Pflop/s**
Power	**7 MW**
System memory	0.3 PB
Node performance	125 GF
Node memory BW	25 GB/s
Node concurrency	12
Total Node Interconnect BW	3.5 GB/s
System size (nodes)	18,700
Total concurrency	225,000
Storage	15 PB
IO	0.2 TB
MTTI	days

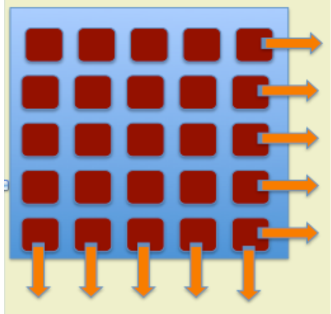
Potential System Architecture with a cap of $200M and 20MW

Systems	2011	2018	Difference Today & 2018
System peak	**2 Pflop/s**	**1 Eflop/s**	**O(1000)**
Power	**7 MW**	**~20 MW**	
System memory	0.3 PB	32 - 64 PB	O(100)
Node performance	125 GF	1,2 or 15TF	O(10) – O(100)
Node memory BW	25 GB/s	2 - 4TB/s	O(100)
Node concurrency	12	O(1k) or 10k	O(100) – O(1000)
Total Node Interconnect BW	3.5 GB/s	200-400GB/s	O(100)
System size (nodes)	18,700	O(100,000) or O(1M)	O(10) – O(100)
Total concurrency	225,000	O(billion)	O(10,000)
Storage	15 PB	500-1000 PB (>10x system memory is min)	O(10) – O(100)
IO	0.2 TB	60 TB/s (how long to drain the machine)	O(100)
MTTI	days	O(1 day)	- O(10)

Exascale (10^{18} Flop/s) Systems: Two Possible Swim Lanes

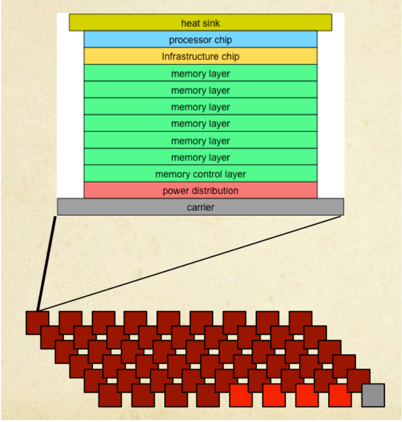


- **Light weight processors (think BG/P)**
  - ~1 GHz processor (10^9)
  - ~1 Kilo cores/socket (10^3)
  - ~1 Mega sockets/system (10^6)



Socket Level
Cores scale-out for planar geometry

- **Hybrid system (think GPU based)**
  - ~1 GHz processor (10^9)
  - ~10 Kilo FPUs/socket (10^4)
  - ~100 Kilo sockets/system (10^5)



Node Level
3D packaging

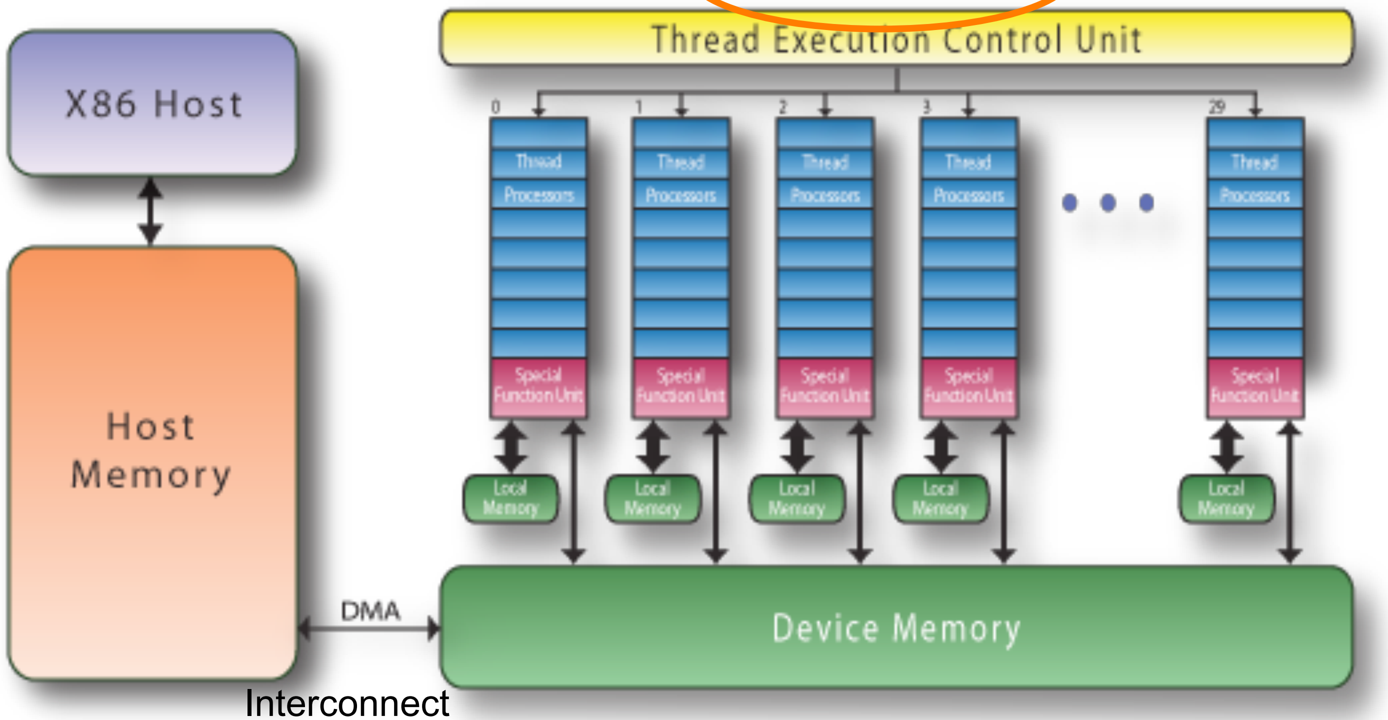
Commodity plus Accelerators

Commodity

Intel Xeon
8 cores
3 GHz
8*4 ops/cycle
96 Gflop/s (DP)

Accelerator (GPU)

Nvidia C2050 “Fermi”
448 “Cuda cores”
1.15 GHz
448 ops/cycle
515 Gflop/s (DP)



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

We Have Seen This Before

- Floating Point Systems FPS-164 Scientific Computer (1976)
- Intel Math Co-processor (1980)
- Weitek Math Co-processor (1981)

The Intel® Math CoProcessor™ is for crunching numbers faster.
intel
Personal Computer Enhancement
There's one for every machine.
80387™ Family. For 386™ based machines.
80287 Family. For 80286-based machines.
80387SX™ For 386SX™ based machines.
8087 Family. For 8086 and 8088-based machines.
intel
Personal Computer Enhancement

It's FAST!

The Intel Math CoProcessor dramatically speeds up the number-crunching that's part of the work you do every day: budgeting, statistical analysis, financial analysis, CAD and other engineering analysis. In fact, the Math CoProcessor is supported by more than 100 commonly used software packages including Lotus 1-2-3, dBase IV, AutoCAD, and most languages and statistical packages.

It's EASY!

Intel makes a variety of math coprocessors. Every PC has a built in socket. Just plug it in and go.

It's SAFE!

Made by Intel, the same people who designed your PC's microprocessor, each and every Math CoProcessor is backed by an industry leading five year warranty and toll-free technical support. You are assured the highest degree of quality, compatibility, reliability and support for your investment.

For more information, or technical support call:
(800) 538-3373 in the U.S. and Canada
(503) 629-7354 for International

[illegible]

1980

Balance Between Data Movement and Floating point

- FPS-164 and VAX (1976)
  - 11 Mflop/s; transfer rate 44 MB/s
  - Ratio of flops to bytes of data movement: 1 flop per 4 bytes transferred
- Nvidia Fermi and PCI-X to host
  - 500 Gflop/s; transfer rate 8 GB/s
  - Ratio of flops to bytes of data movement: 62 flops per 1 byte transferred
- Flop/s are cheap, so are provisioned in excess

Challenges of using GPUs

- **High levels of parallelism**

  Many GPU cores, serial kernel execution

  [e.g. 240 in the Nvidia Tesla; up to 512 in *Fermi* - to have concurrent kernel execution]

- **Hybrid/heterogeneous architectures**

  Match algorithmic requirements to architectural strengths

  [e.g. small, non-parallelizable tasks to run on CPU, large and parallelizable on GPU]

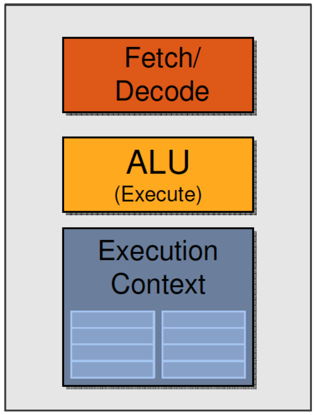
- **Compute *vs* communication gap**

  Exponentially growing gap; persistent challenge

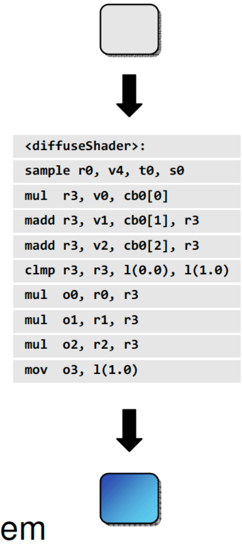
  [Processor speed improves 59%, memory bandwidth 23%, latency 5.5%]
  [on all levels, e.g. a GPU Tesla C1070 (4 x C1060) has compute power of O(1,000) Gflop/s but GPUs communicate through the CPU using O(1) GB/s connection]

How to Count Cores?

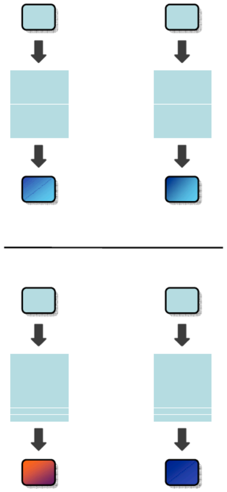
- **CPU Conventional Core** **Quad**

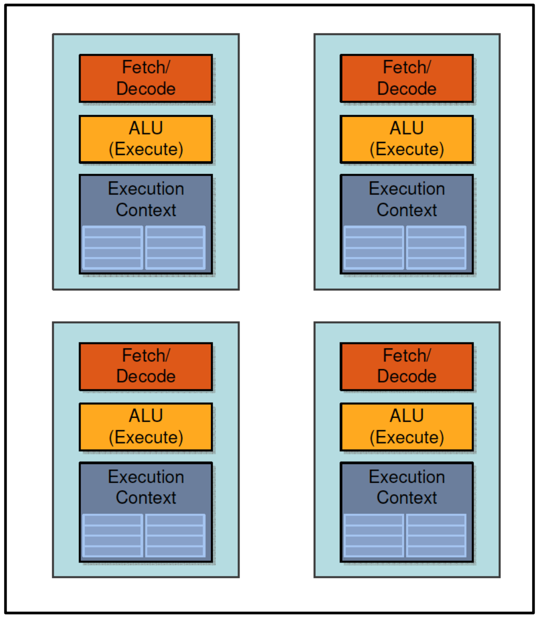


One instruction stream per work-item



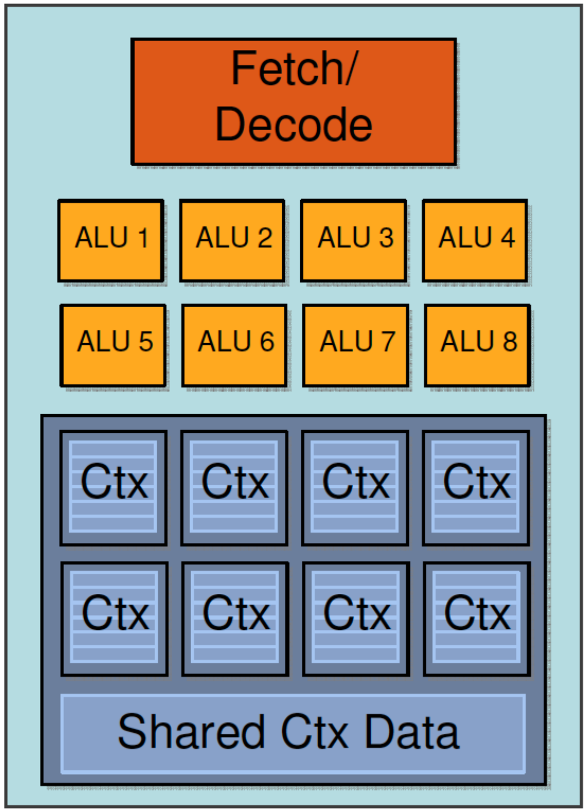
```
<diffuseShader>:
sample r0, v4, t0, s0
mul  r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, l(0.0), l(1.0)
mul  o0, r0, r3
mul  o1, r1, r3
mul  o2, r2, r3
mov  o3, l(1.0)
```



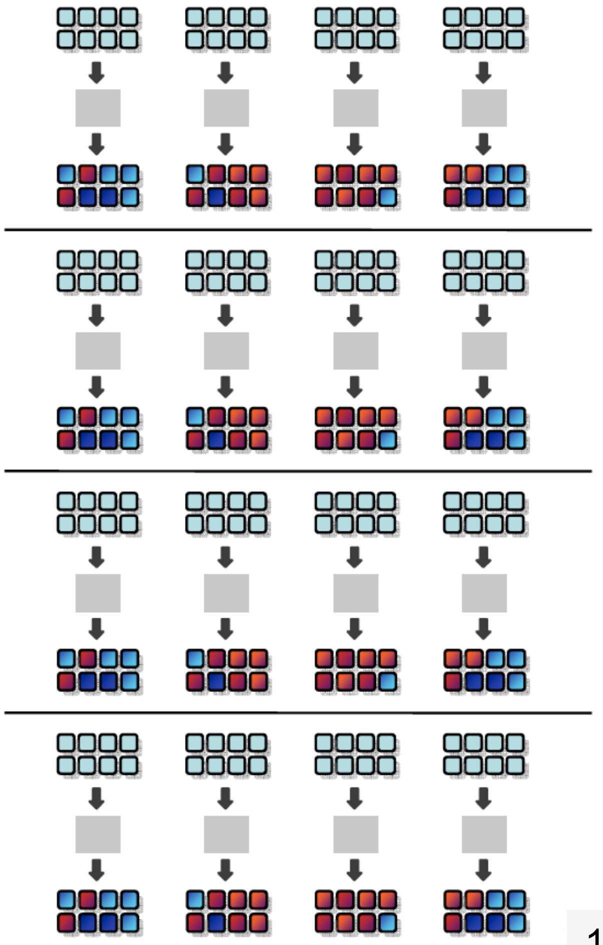


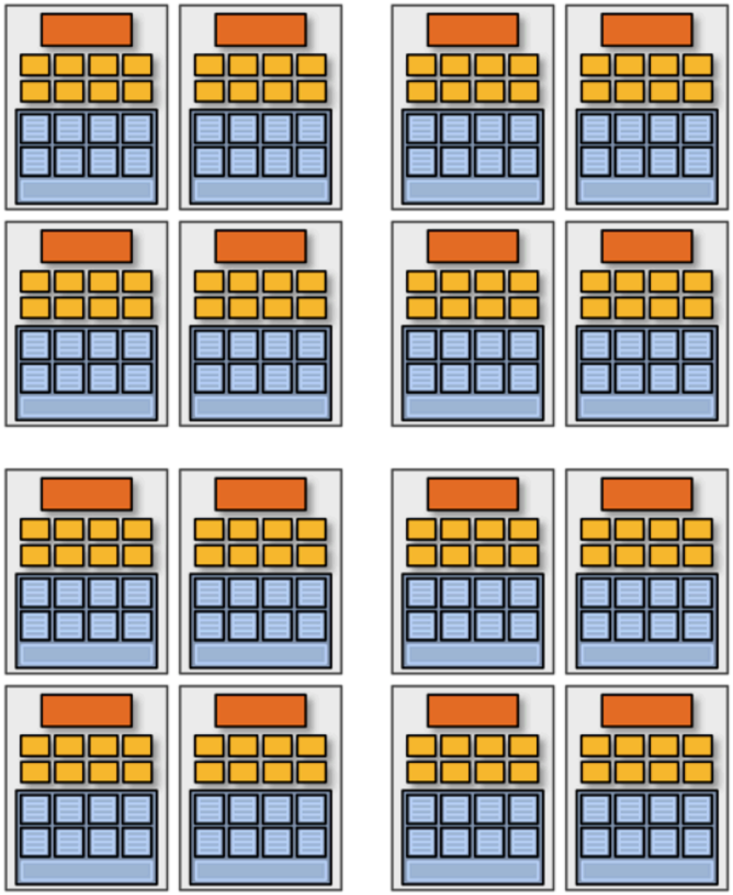
In GPUs - Add ALUs

- SIMD Processing
- Amortize cost /complexity of managing an instruction stream across many ALUs.
- NVIDIA refers to these ALUs as “CUDA Cores” (also streaming processors)



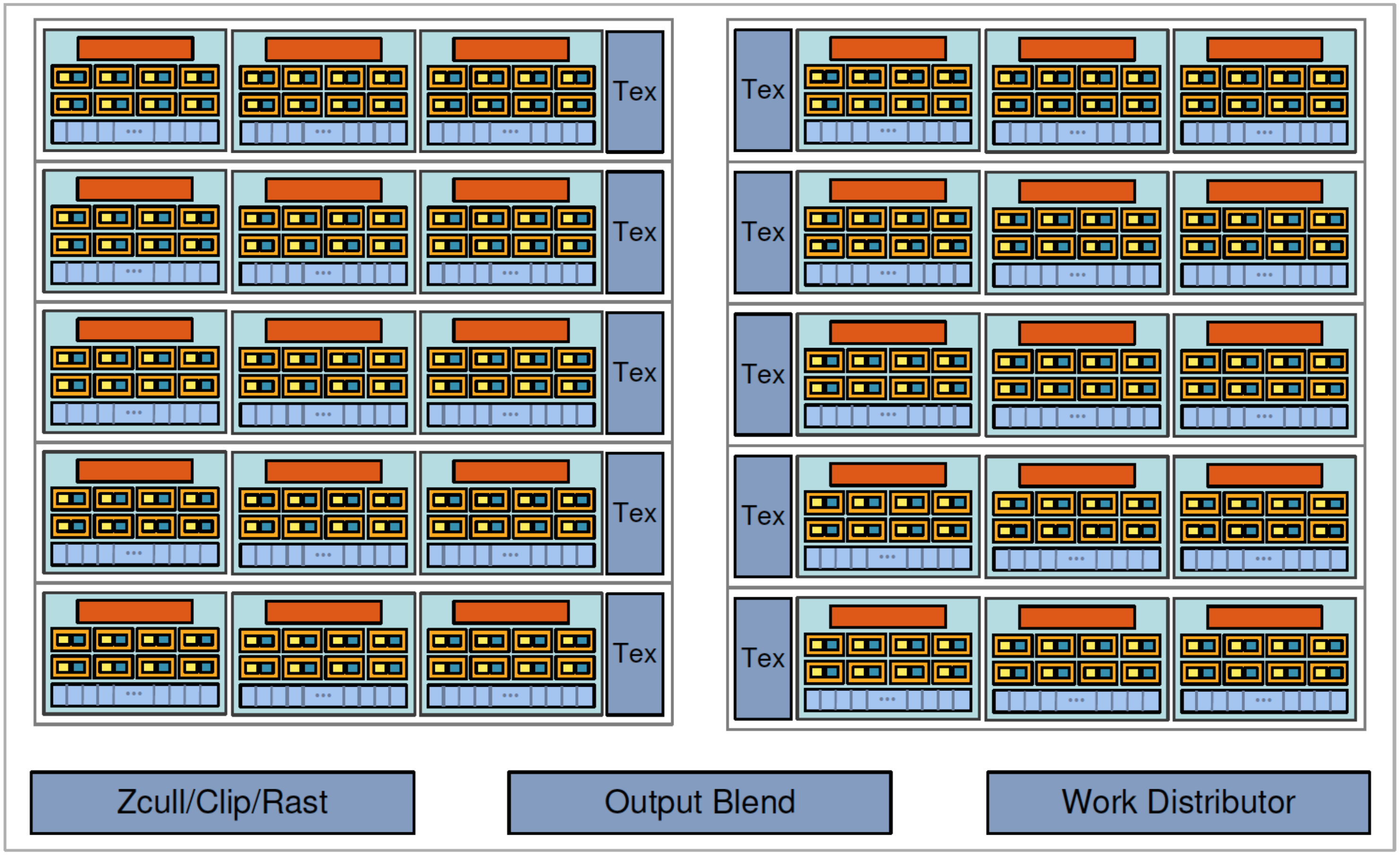
128 Elements in Parallel





16 cores each with 8 ALUs (CUDA Cores)
Total of 16 simultaneous instruction streams with
128 ALUs (CUDA Cores)

NVIDIA GT280 “old Telsa”

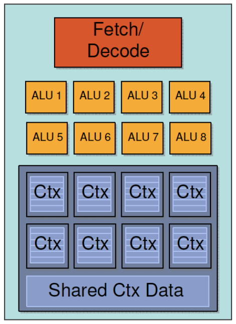


- **240 streaming processors (CUDA Cores) (ALUs)**
- **Equivalent to 30 processing cores, each with 8 “CUDA cores”**

NVIDIA GeForce GTX 280 (Tesla)

- NVIDIA-Speak
  - 240 CUDA cores (ALUs)
- Generic speak
  - 30 processing cores
    - 8 CUDA Cores (SIMD functional units) per core
  - 1 mul-add (2 flops) + 1 mul per functional unit (3 flops/cycle)
  - Best case theoretically: 240 mul-adds + 240 muls per cycle
    - 1.3 GHz clock
    - 30 * 8 * (2 + 1) * 1.33 = 933 Gflop/s peak
  - Best case reality: 240 mul-adds per clock
    - Just able to do the mul-add so 2/3 or 624 Gflop/s
  - All this is single precision
    - Double precision is 78 Gflop/s peak (Factor of 8 from SP; exploit mixed prec)
  - 141 GB/s bus, 1 GB memory
  - 4 GB/s via PCIe (we see: T = 11 us + Bytes/3.3 GB/s)
  - In SP SGEMM performance 375 Gflop/s

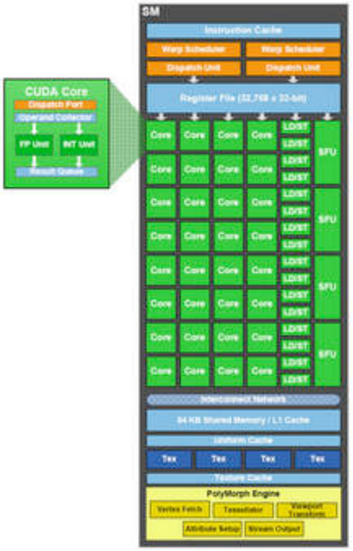
Processing Core

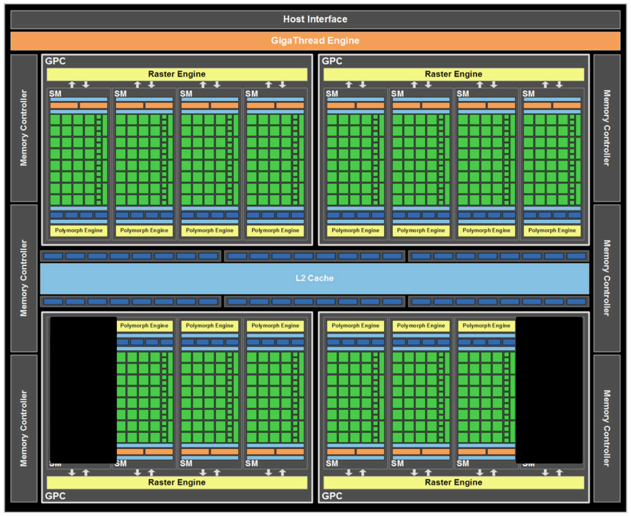


NVIDIA Tesla C2070 (Fermi), GF100 Chip

- **NVIDIA-Speak**
  - 448 CUDA cores (ALUs)
- **Generic speak**
  - 14 processing cores
    - 32 CUDA Cores (SIMD functional units) per core
  - 1 mul-add (2 flops) per ALU (2 flops/cycle)
  - Best case theoretically: 448 mul-adds
    - 1.15 GHz clock
    - 14 * 32 * 2 * 1.15 = 1.03 Tflop/s peak
  - All this is single precision
    - Double precision is half this rate, 515 Gflop/s
  - In SP SGEMM performance 580 Gflop/s
  - In DP DGEMM performance 300 Gflop/s
  - Interface PCI-x16

Processing Core



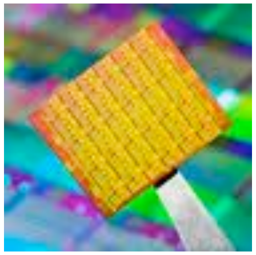


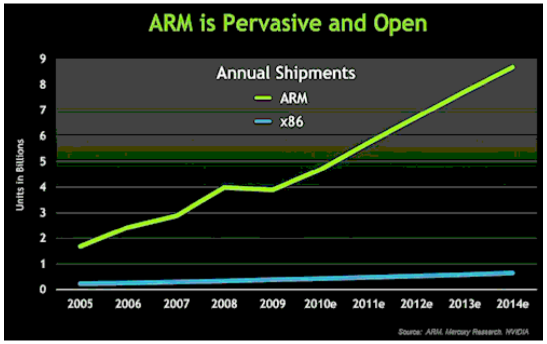
Future Computer Systems

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

AMD

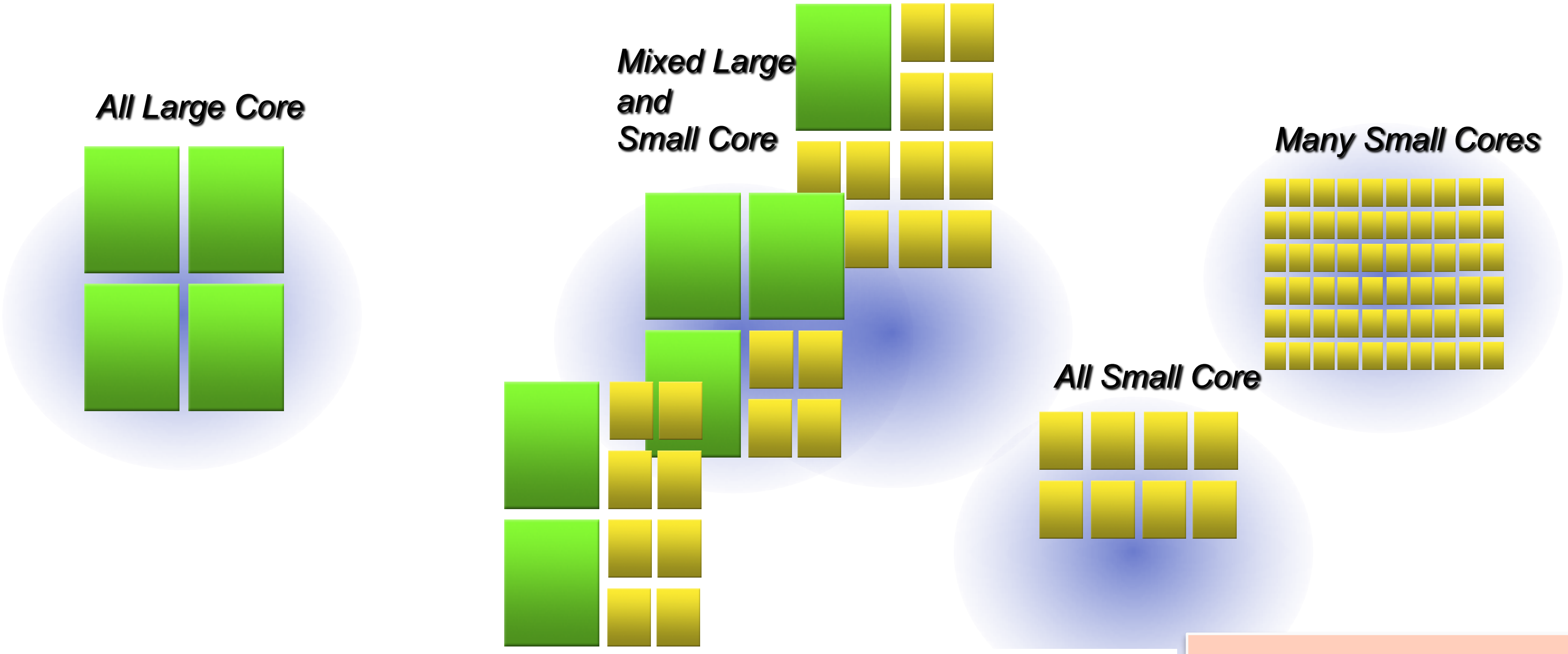
The future is fusion



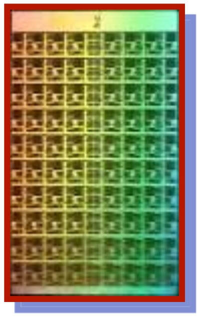


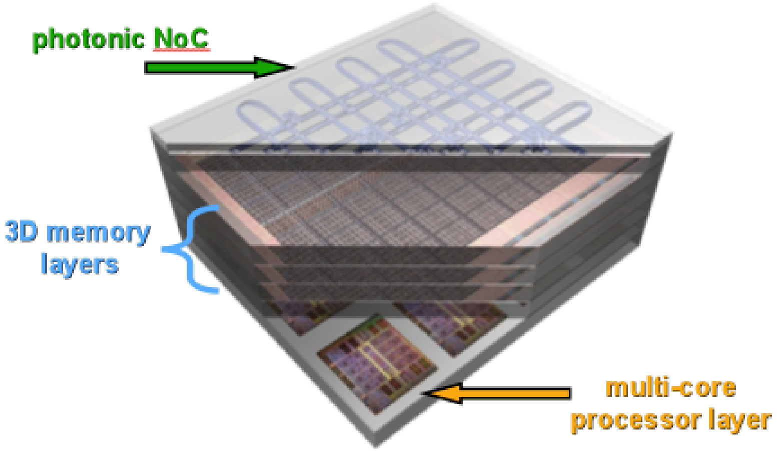
Source: ARM, Mercury Research, NVIDIA

What’s Next?



Many Floating-Point Cores





+ 3D Stacked Memory

Different Classes of Chips
- Home
- Games / Graphics
- Business
- Scientific

The High Cost of Data Movement

- Flop/s or percentage of peak flop/s become much less relevant

Approximate power costs (in picoJoules)

	2011
DP FMADD flop	100 pJ
DP DRAM read	4800 pJ
Local Interconnect	7500 pJ
Cross System	9000 pJ

Source: John Shalf, LBNL

- Algorithms & Software: minimize data movement; perform more work per unit data movement.

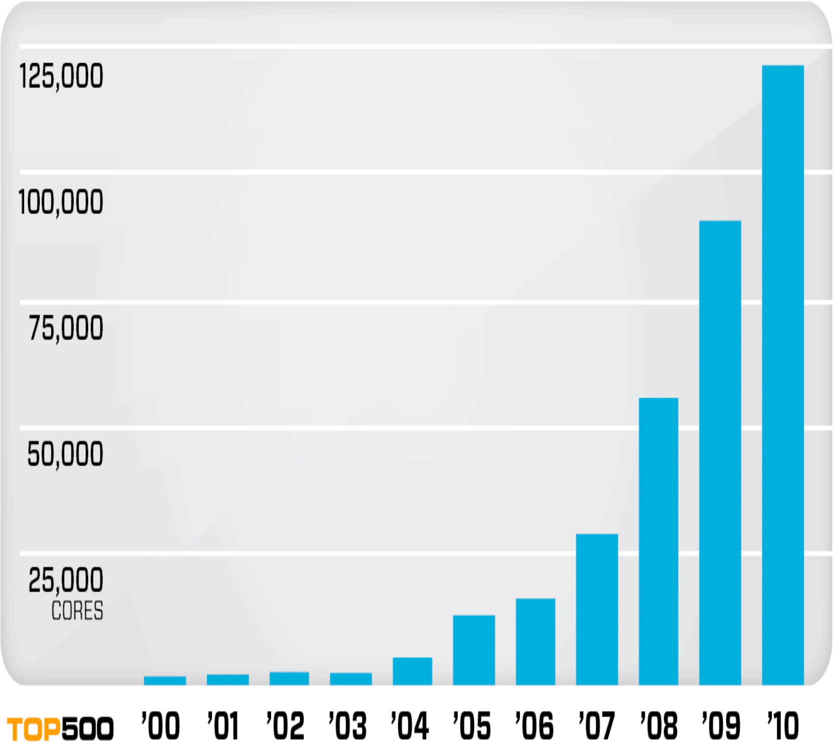
Moore’s Law Reinterpreted

- Number of cores per chip doubles every 2 year, while clock speed decreases (not increases).
  - Need to deal with systems with millions of concurrent threads
    - Future generation will have billions of threads!
  - Need to be able to easily replace inter -chip parallelism with intro-chip parallelism
- Number of threads of execution doubles every 2 year

Factors that Necessitate Redesign of Our Software

- Steepness of the ascent from terascale to petascale to exascale
- Extreme parallelism and hybrid design
  - Preparing for million/billion way parallelism
- Tightening memory/bandwidth bottleneck
  - Limits on power/clock speed implication on multicore
  - Reducing communication will become much more intense
  - Memory per core changes, byte-to-flop ratio will change
- Necessary Fault Tolerance
  - MTTF will drop
  - Checkpoint/restart has limitations
  - shared responsibility

Average Number of Cores per Supercomputer for Top 20 Systems



Software infrastructure does not exist today

Major Changes to Software

- Must rethink the design of our software
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software

Emerging Architectures

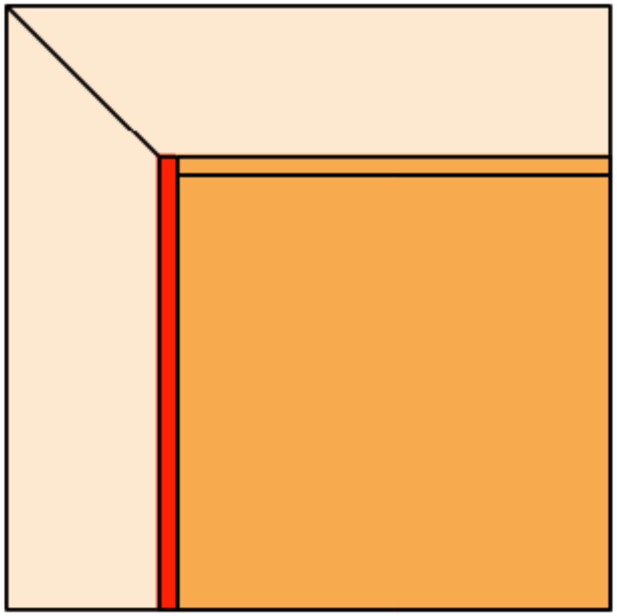
- Are needed by applications

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

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

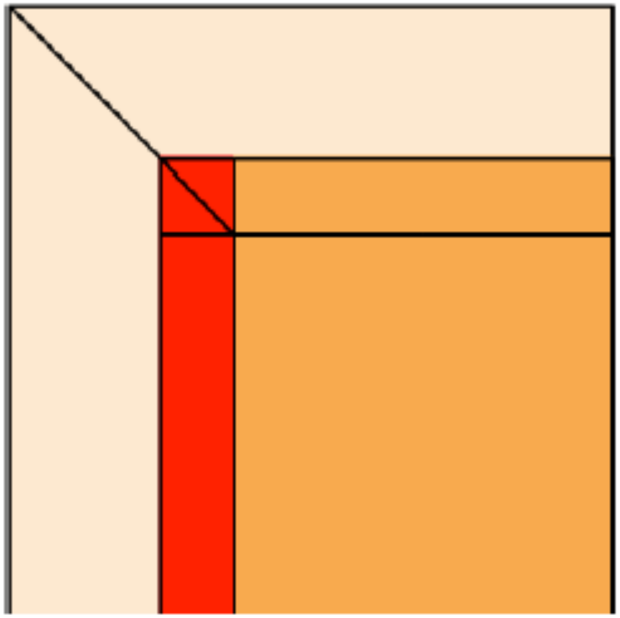
Software/Algorithms follow hardware evolution in time

- **70's - LINPACK, vector operations:**
  - **Vector architectures**
  - **Level-1 BLAS operation**
    - Vector operations



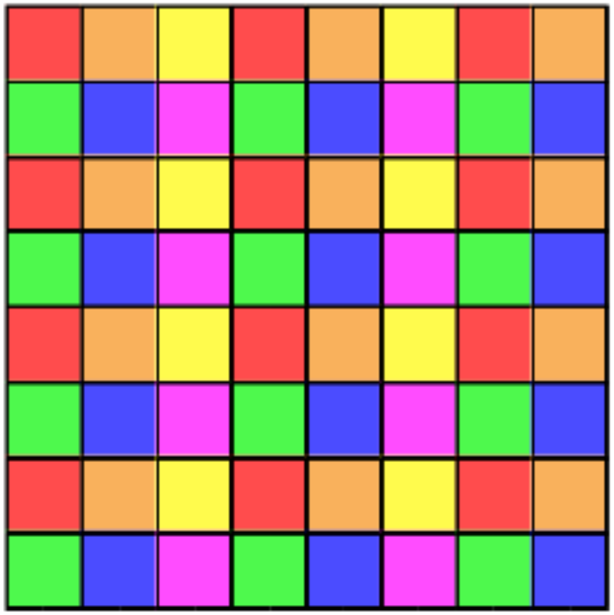
Software/Algorithms follow hardware evolution in time

- 70's - LINPACK, vector operations:
  - Vector architectures
  - Level-1 BLAS operation
    - Vector operations
- 80's - LAPACK, block operations
  - SMP architectures
  - Cache based
    - Matrix operations



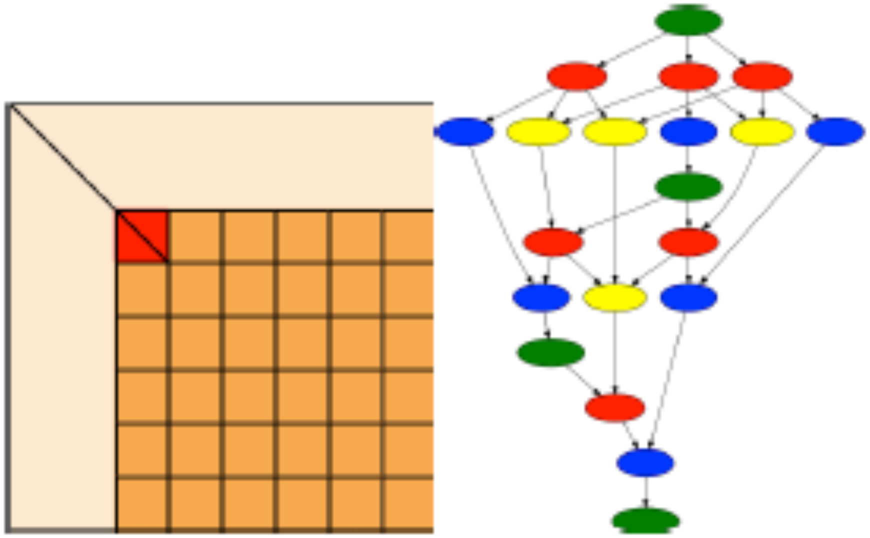
Software/Algorithms follow hardware evolution in time

- 70's - LINPACK, vector operations:
  - Vector architectures
  - Level-1 BLAS operation
    - Vector operations
- 80's - LAPACK, block operations
  - SMP architectures
  - Cache based
    - Matrix operations
- 90's - ScaLAPACK, dist. Memory
  - Message passing
  - Data decomposition



Software/Algorithms follow hardware evolution in time

- **Today - Multicore Heterogeneous Architectures**
  - **PLASMA, many-cores friendly:**
    - Tile based - DAG scheduled algorithms, block data layout
  - **MAGMA, GPU:**
    - GPU BLAS



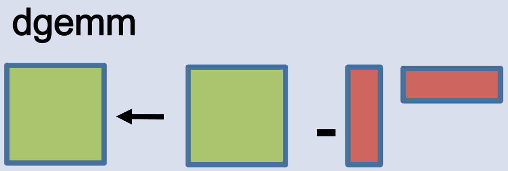
Exascale algorithms that expose and exploit multiple levels of parallelism

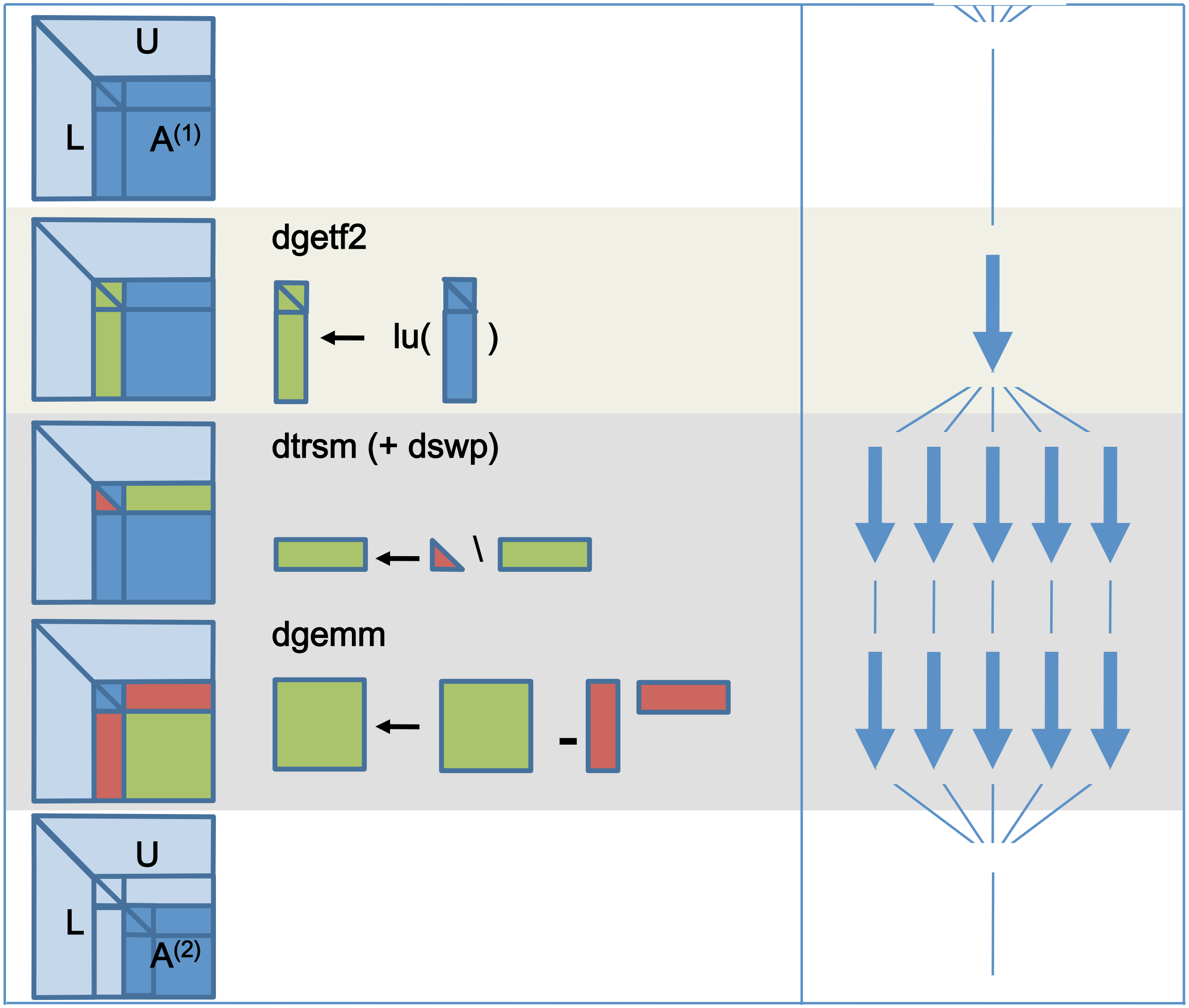
- Synchronization-reducing algorithms
  - Break Fork-Join model
- Communication-reducing algorithms
  - Use methods which have lower bound on communication
- Fault resilient algorithms
  - Implement algorithms that can recover from failures
- Mixed precision methods
  - 2x speed of ops and 2x speed for data movement
- Reproducibility of results
  - Today we can’t guarantee this

Fork-Join Parallelization of LU and QR.

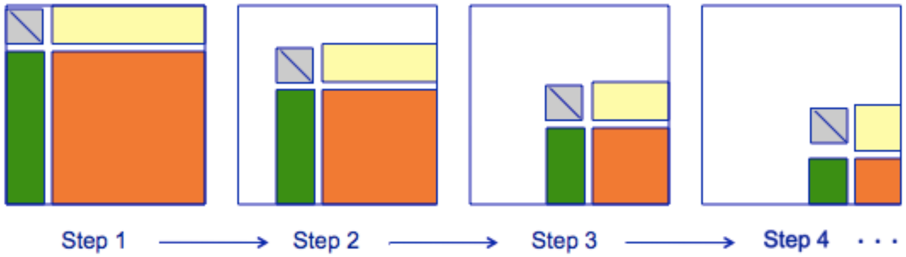
Parallelize the update:

- 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

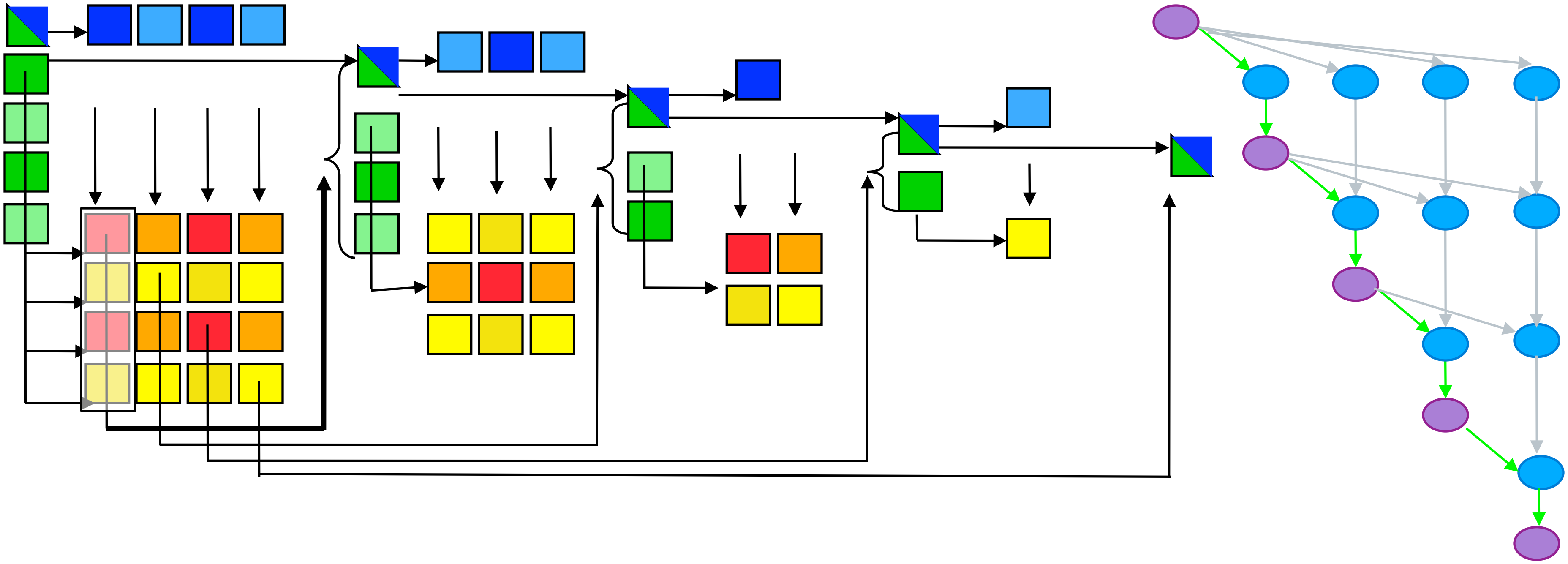




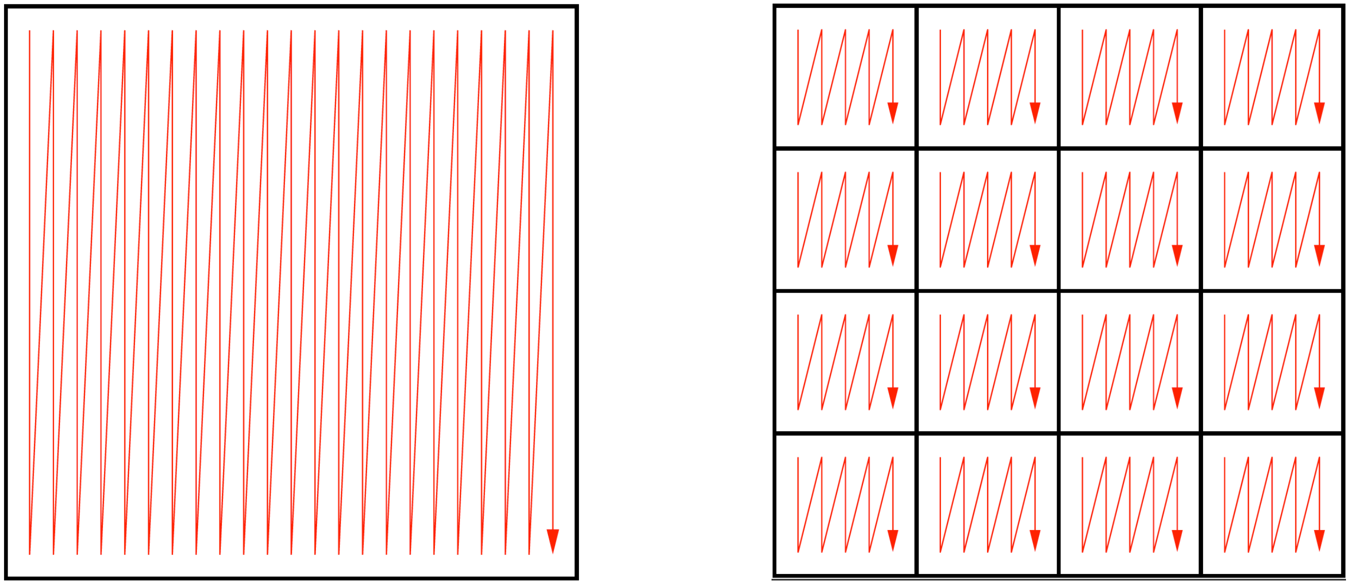
Parallel Tasks in LU/LL^T/QR



- Break into smaller tasks and remove dependencies



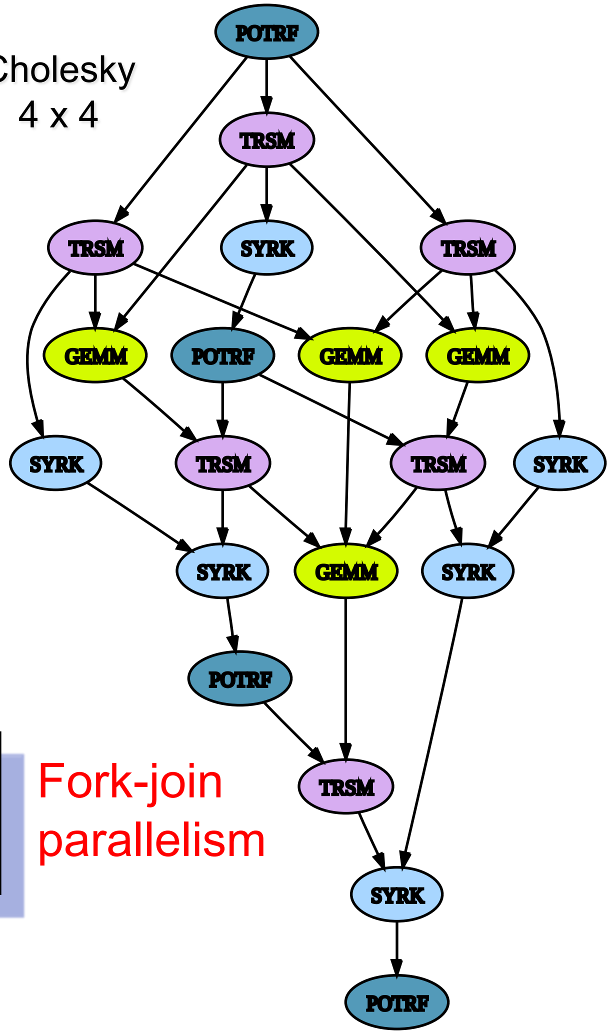
Data Layout is Critical

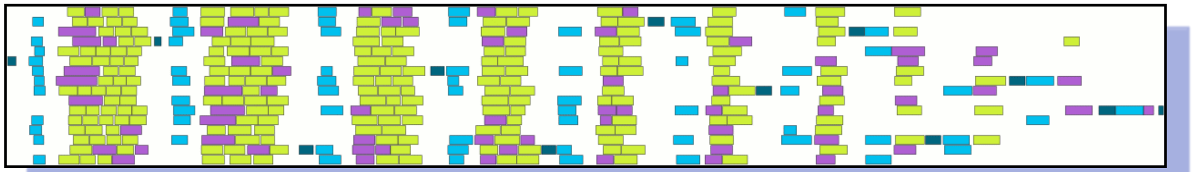


- **Tile data layout where each data tile is contiguous in memory**
- **Decomposed into several fine-grained tasks, which better fit the memory of the small core caches**

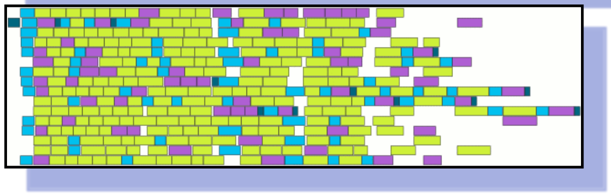
PLASMA: Parallel Linear Algebra s/w for Multicore Architectures

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





Fork-join parallelism



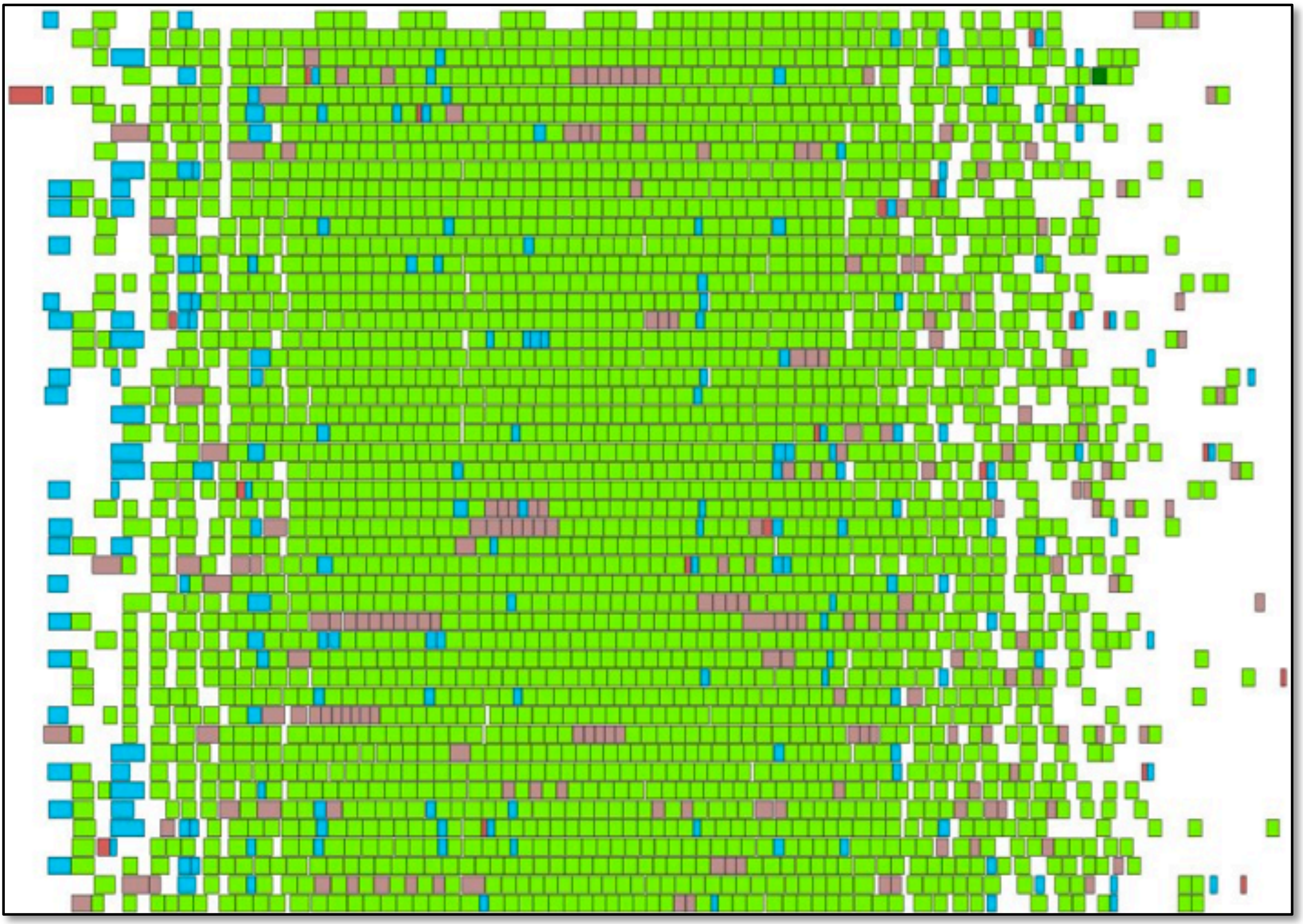
DAG scheduled parallelism

Time

Synchronization Reducing Algorithms

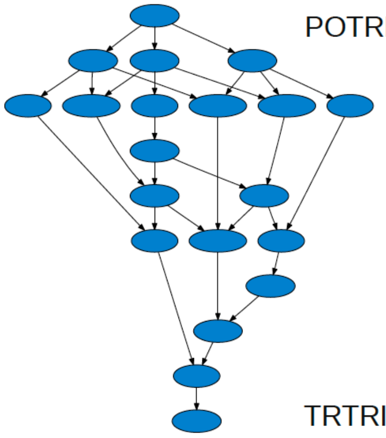
- Regular trace
- Factorization steps pipelined
- Stalling only due to natural load imbalance
- Dynamic
- Out of order execution
- Fine grain tasks
- Independent block operations

The colored area over the rectangle is the efficiency

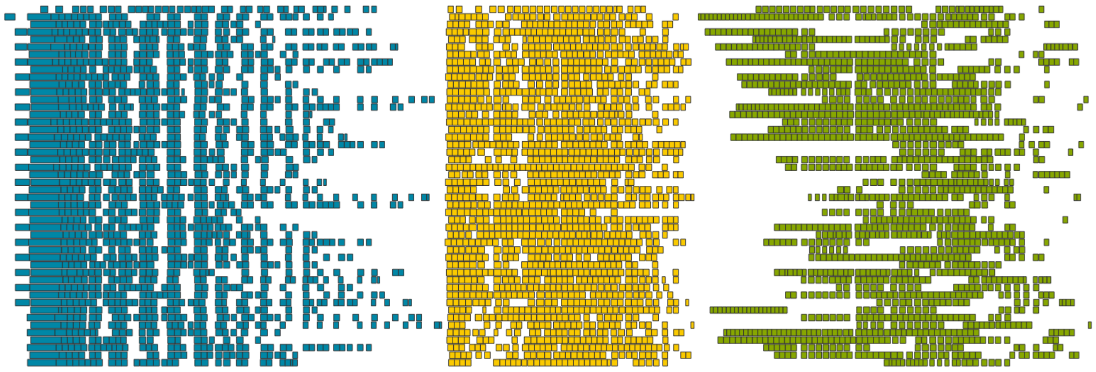


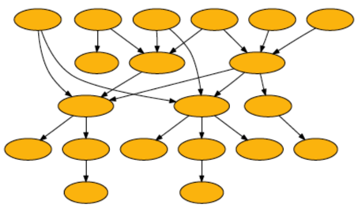
Tile LU factorization; Matrix size 4000x4000, Tile size 200
8-socket, 6-core (48 cores total) AMD Istanbul 2.8 GHz

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

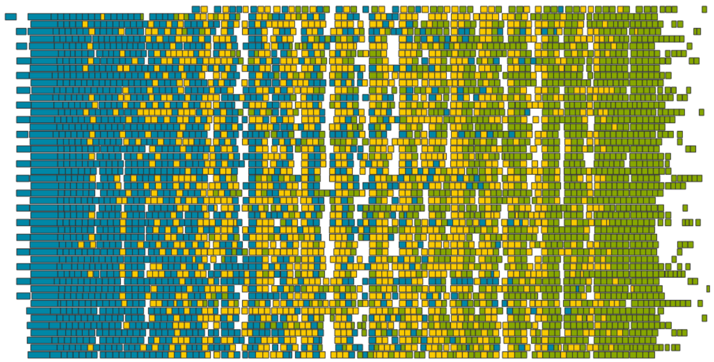


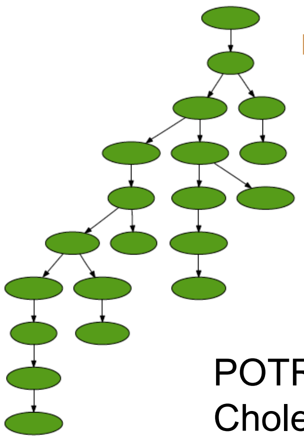
POTRF



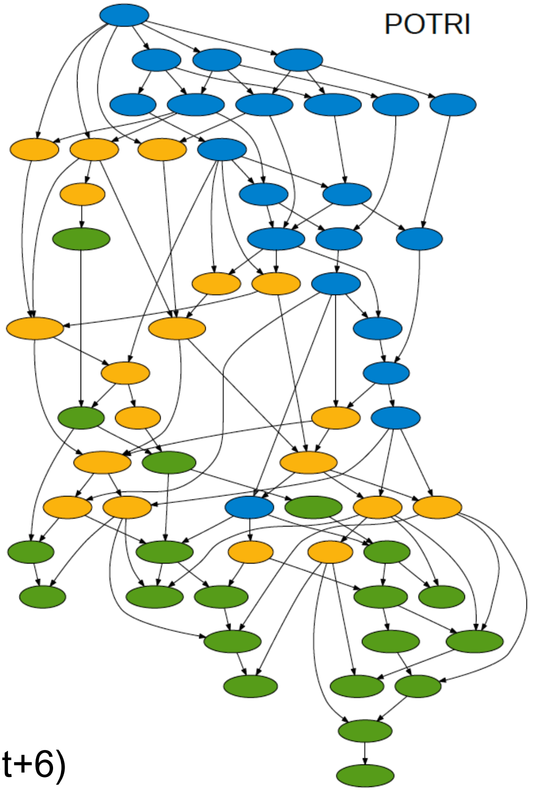


TRTRI





LAUUM



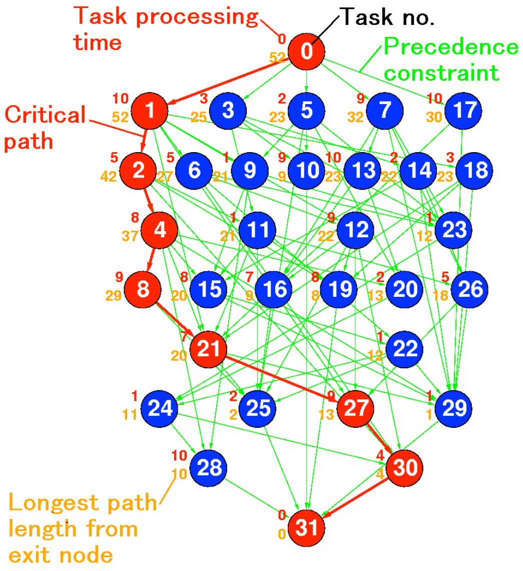
48 cores
POTRF, TRTRI and LAUUM.
The matrix is 4000 x 4000,tile size is 200 x 200,

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

Pipelined: 18 (3t+6)

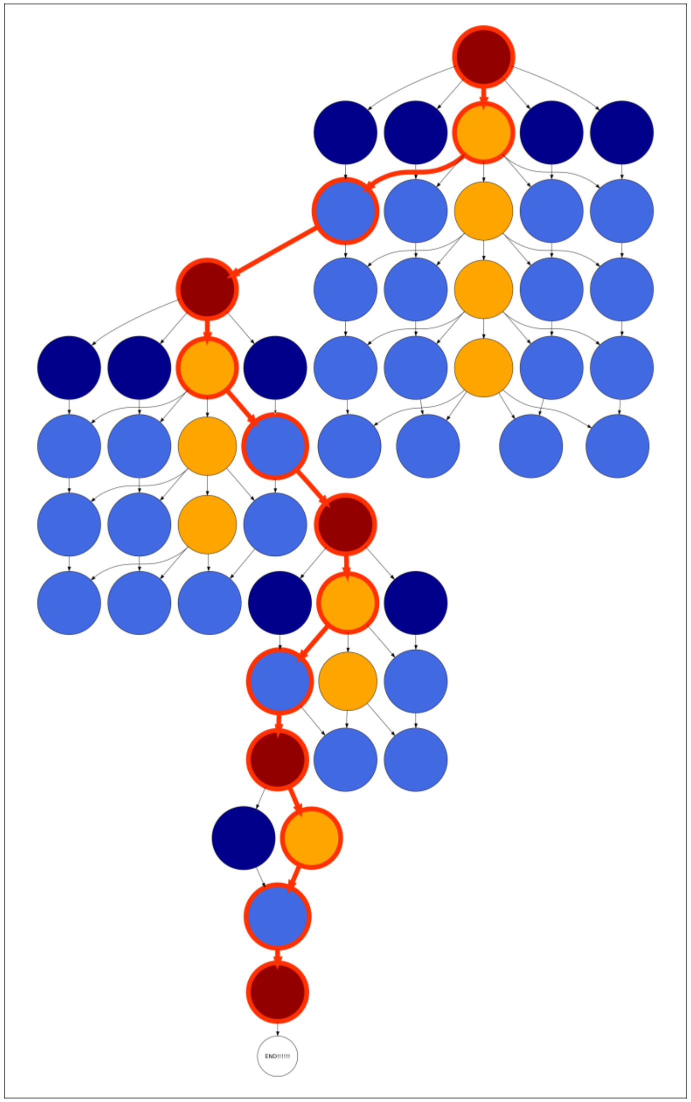
Prioritization of critical path and noncritical tasks

- DAG scheduling of critical path tasks
- Allows taking advantage of asynchronicity between major steps and adaptive load balancing for noncritical tasks



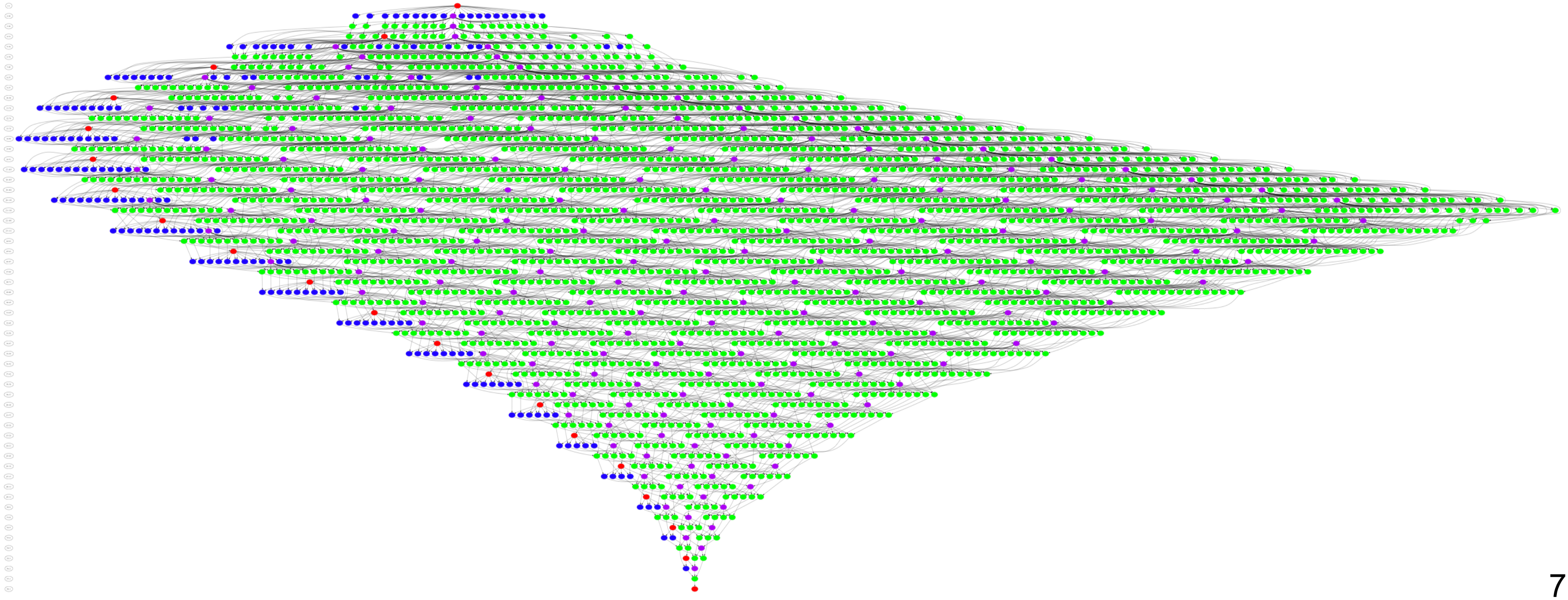
If We Had A Small Matrix Problem

- We would generate the DAG, find the critical path and execute it.
- DAG too large to generate ahead of time
  - Not explicitly generate
  - Dynamically generate the DAG as we go
- Machines will have large number of cores in a distributed fashion
  - Will have to engage in message passing
  - Distributed management
  - Locally have a run time system



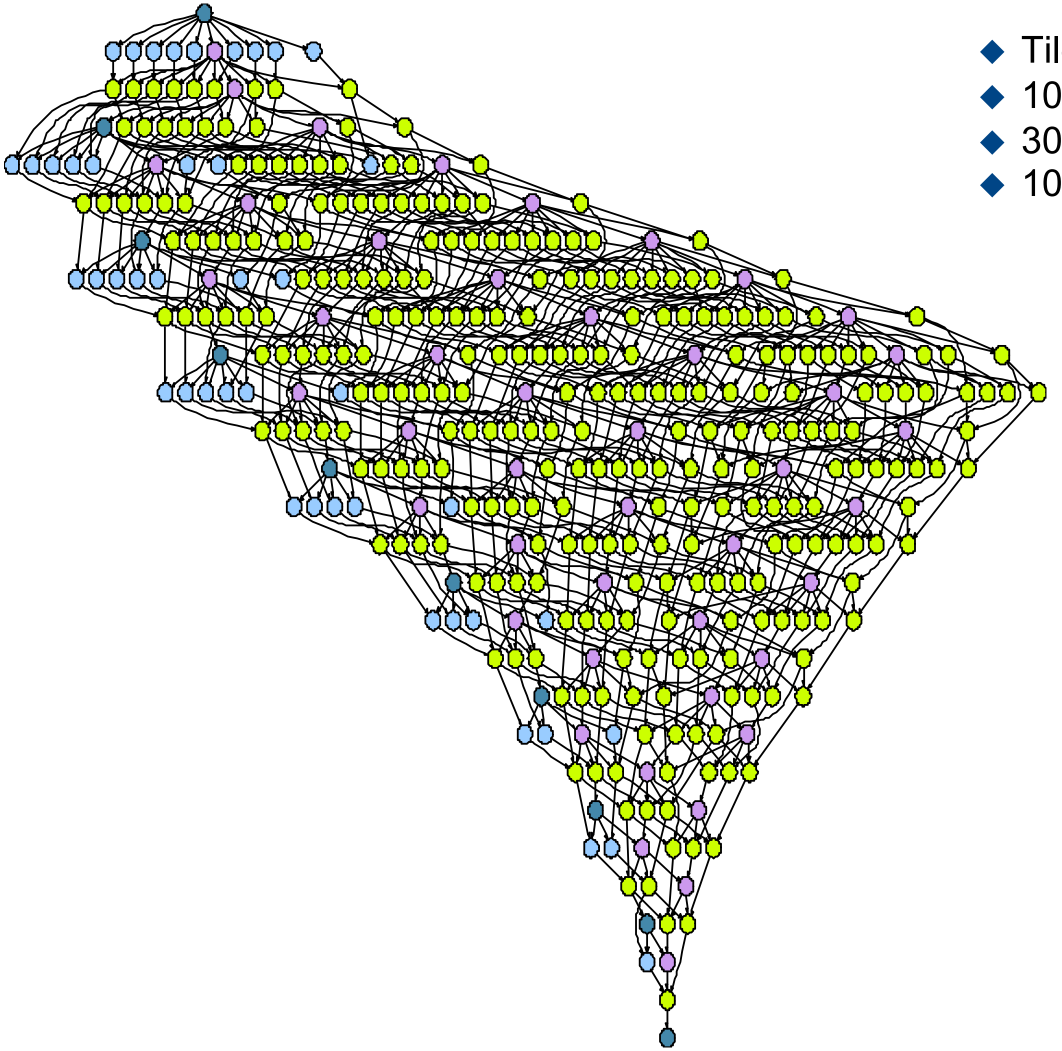
Big DAGs: No Global Critical Path

- DAGs get very big, very fast
  - So windows of active tasks are used; this means no global critical path
  - Matrix of NBxNB tiles; NB^3 operation
    - NB=100 gives 1 million tasks



PLASMA Local Scheduling

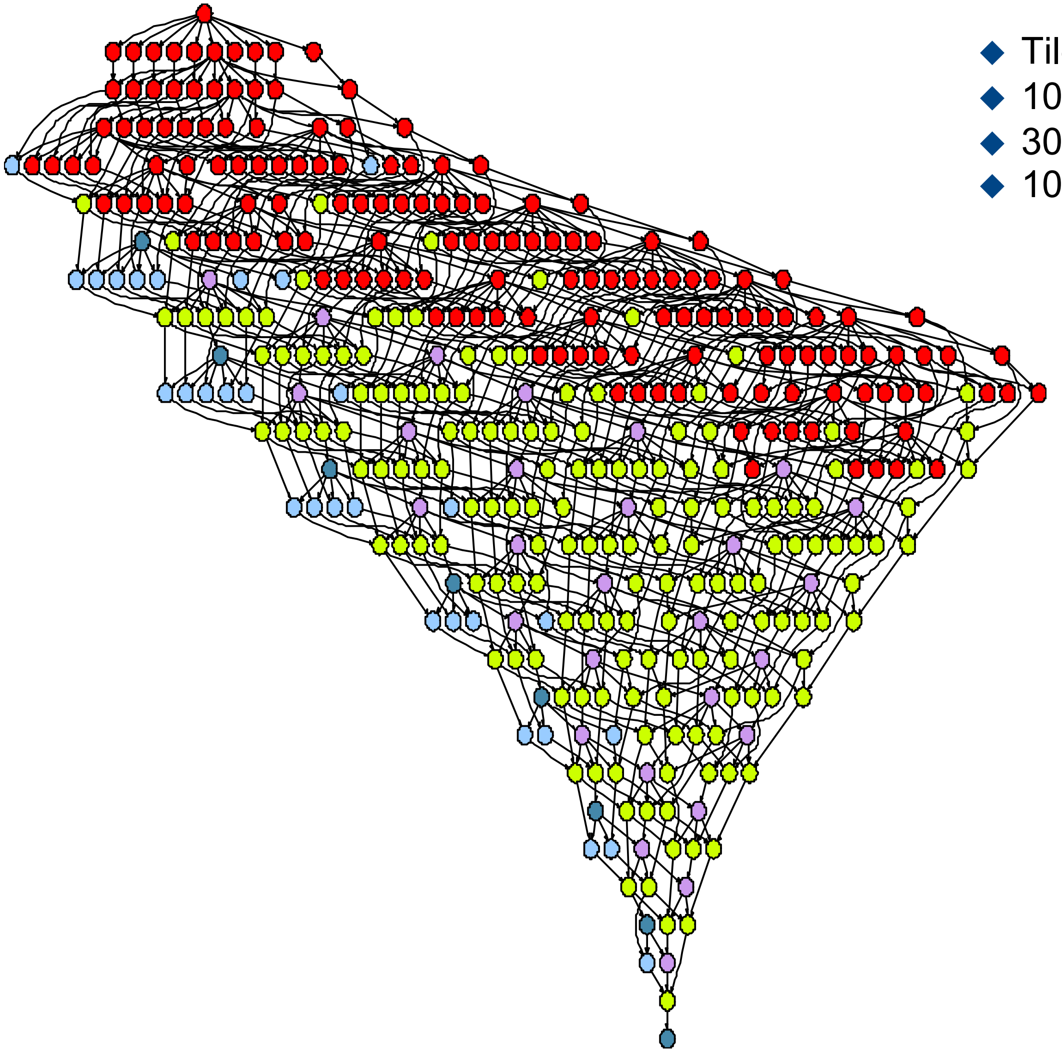
Dynamic Scheduling: Sliding Window



- Tile LU factorization
- 10 x 10 tiles
- 300 tasks
- 100 task window

PLASMA Local Scheduling

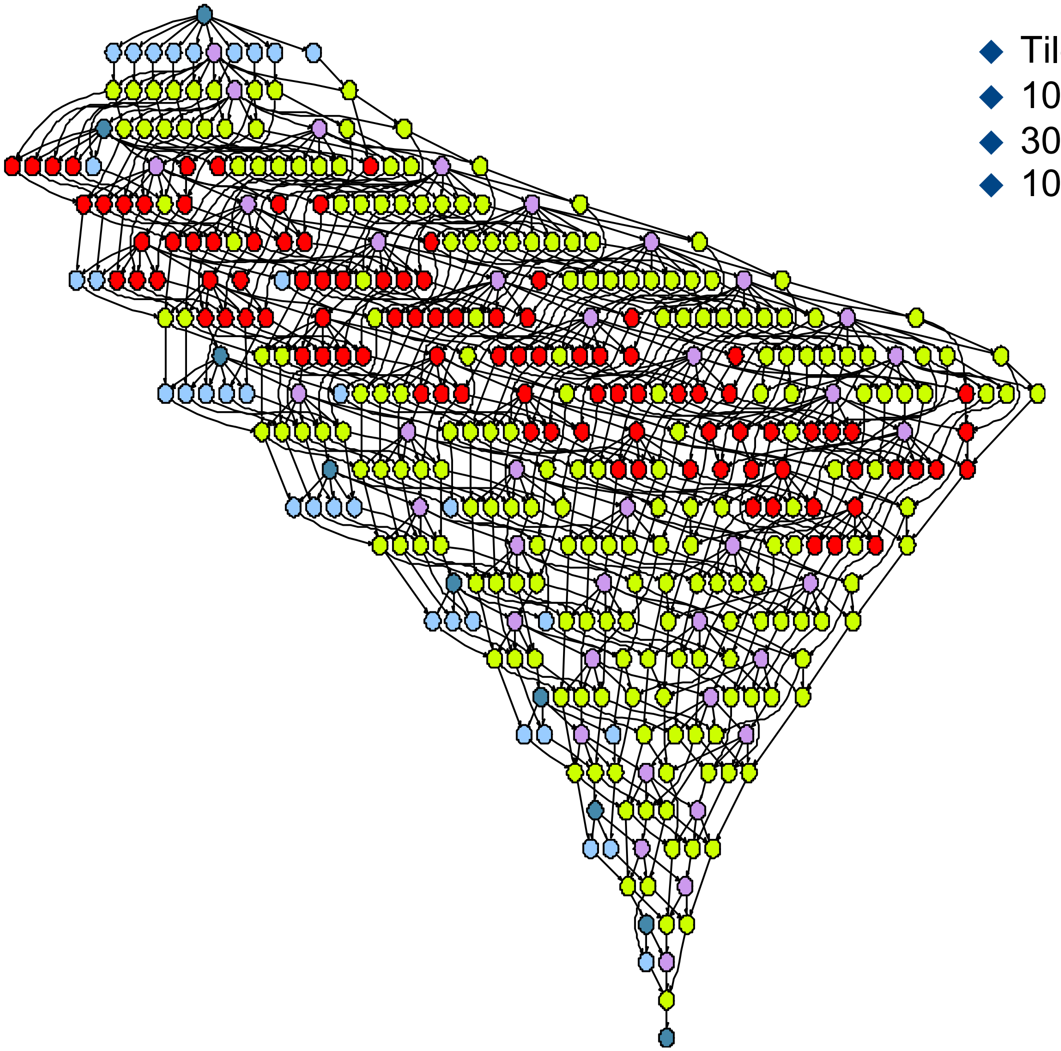
Dynamic Scheduling: Sliding Window



- Tile LU factorization
- 10 x 10 tiles
- 300 tasks
- 100 task window

PLASMA Local Scheduling

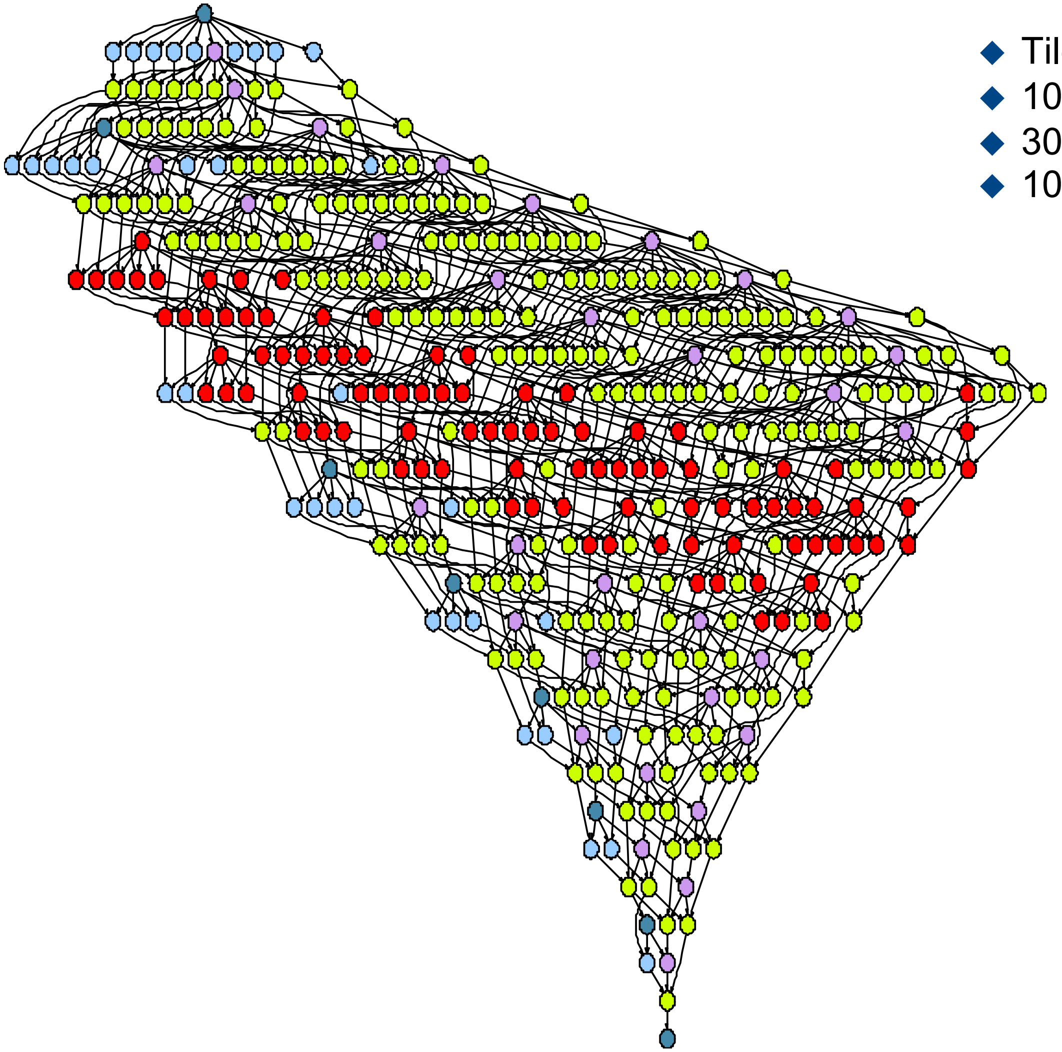
Dynamic Scheduling: Sliding Window



- Tile LU factorization
- 10 x 10 tiles
- 300 tasks
- 100 task window

PLASMA Local Scheduling

Dynamic Scheduling: Sliding Window



- Tile LU factorization
- 10 x 10 tiles
- 300 tasks
- 100 task window

Some Questions

- What’s the best way to represent the DAG?
- What’s the best approach to dynamically generating the DAG?
- What run time system should we use?
  - We will probably build something that we would target to the underlying system’s RTS.
- What about work stealing?
  - Can we do better than nearest neighbor work stealing?
- What does the program look like?
  - Experimenting with Cilk, Charm++, UPC, Intel Threads
  - I would like to reuse as much of the existing software as possible

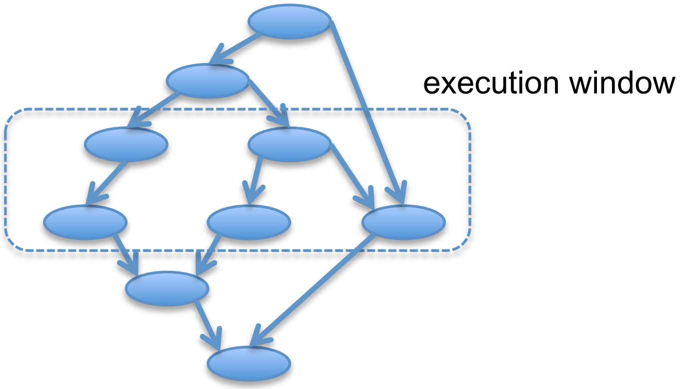
PLASMA Scheduling

Dynamic Scheduling with QUARK

- Sequential algorithm definition
- Side-effect-free tasks
- Directions of arguments (IN, OUT, INOUT)
- Runtime resolution of data hazards (RaW, WaR, WaW)
- Implicit construction of the DAG
- Processing of the tasks by a sliding window

- Old concept
  - Jade (Stanford University)
  - SMP Superscalar (Barcelona Supercomputer Center)
  - StarPU (INRIA)

PLASMA
(On Node)



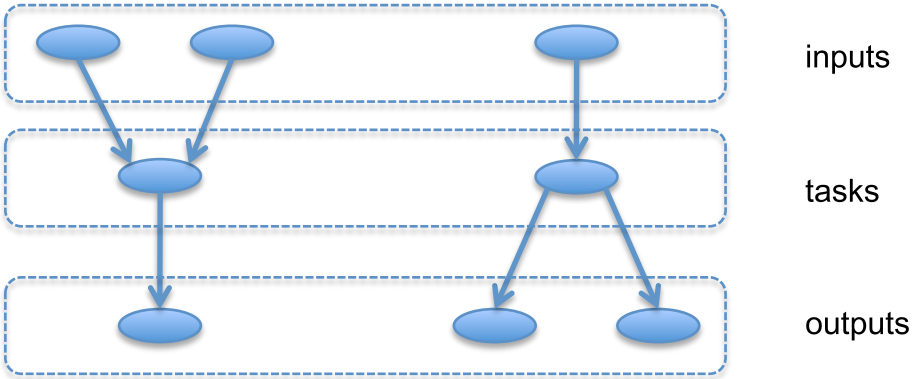
QUARK

Number of tasks in DAG:

$O(n^3)$

Cholesky: $1/3\ n^3$
LU: $2/3\ n^3$
QR: $4/3\ n^3$

DPLASMA
(Distributed System)



DAGuE

Number of tasks in parameterized DAG:

O(1)

Cholesky: 4 (POTRF, SYRK, GEMM, TRSM)
LU: 4 (GETRF, GESSM, TSTRF, SSSSM)
QR: 4 (GEQRT, LARFB, TSQRT, SSRFB)

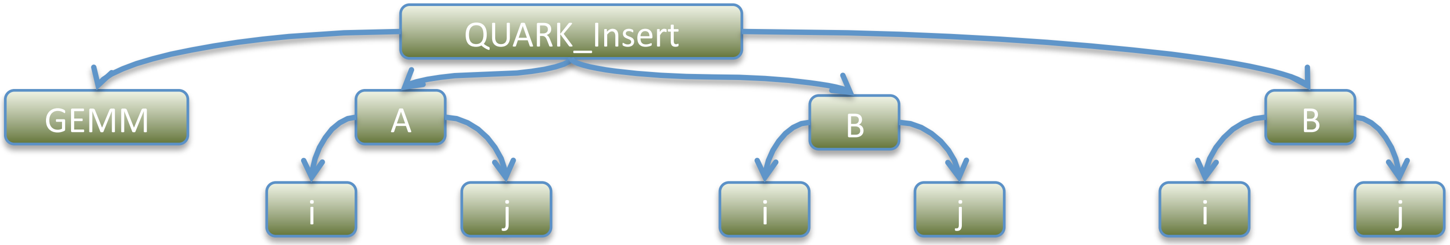
DAG: Conceptualized & Parameterized

small enough to store on each core in every node = Scalable

Start with PLASMA

```
for i,j = 0..N
    QUARK_Insert( GEMM,  A[i, j],INPUT,   B[j, i],INPUT,  C[i,i],INOUT )
    QUARK_Insert( TRSM,  A[i, j],INPUT,   B[j, i],INOUT )
```

Parse the C source code to Abstract Syntax Tree

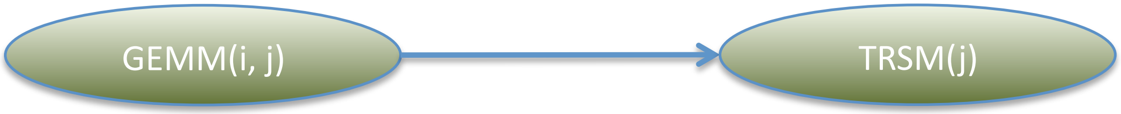


Analyze dependencies with Omega Test

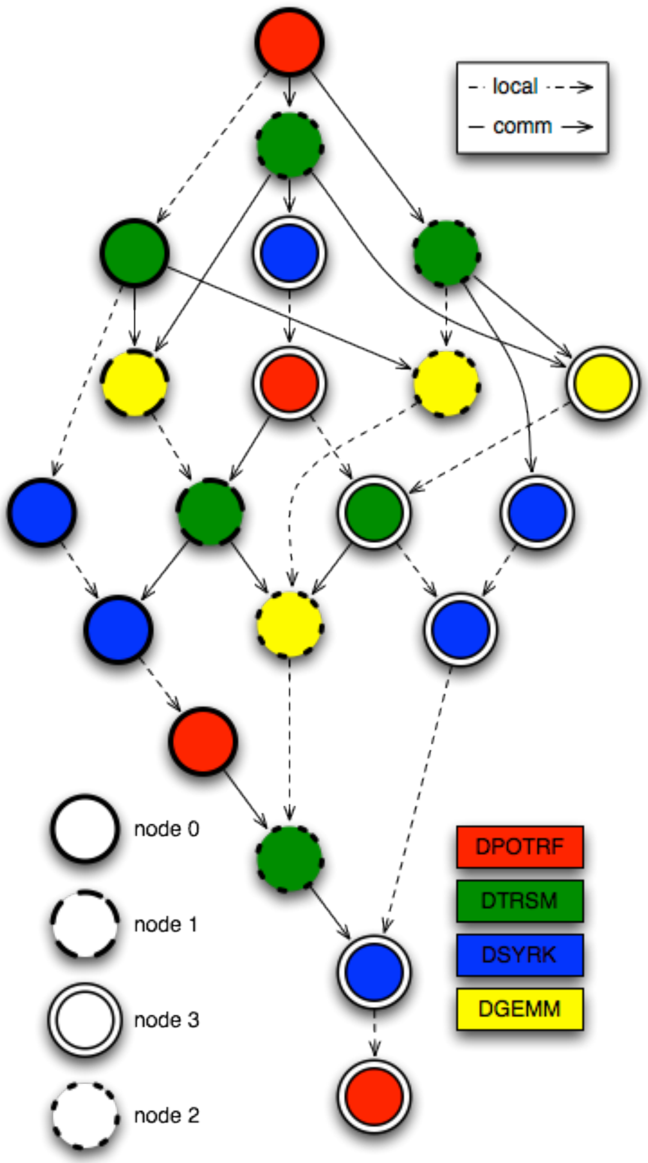
```
{ 1 < i < N : GEMM(i, j) => TRSM(j) }
```

Loops & array references have to be affine

Generate Code which has the Parameterized DAG

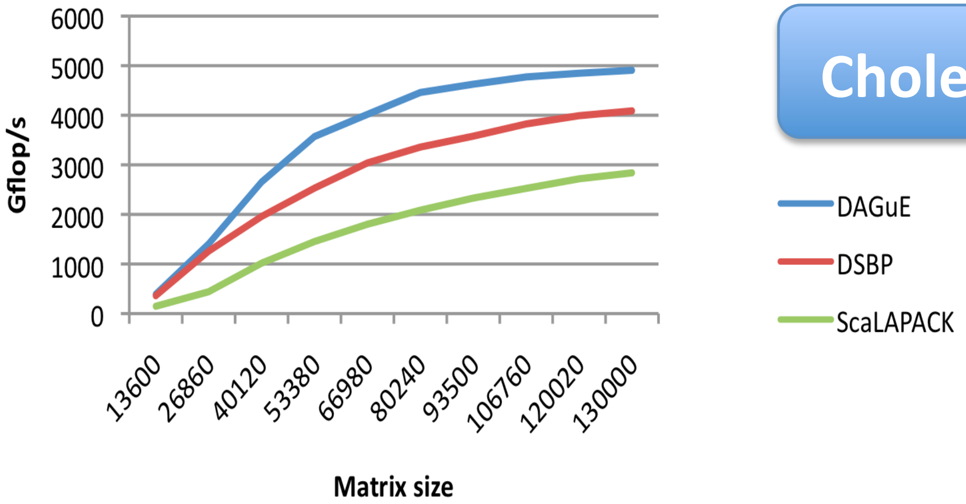


Example: Cholesky 4x4



- RT is using the symbolic information from the compiler to make scheduling, message passing, & RT decisions
- Data distribution: regular, irregular
- Task priorities
- No left looking or right looking, more adaptive or opportunistic

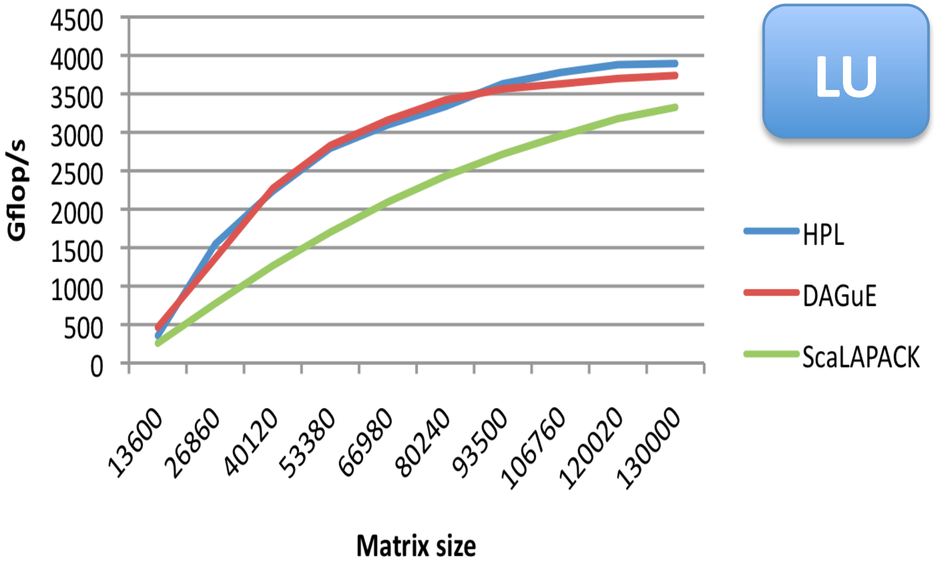
Cholesky



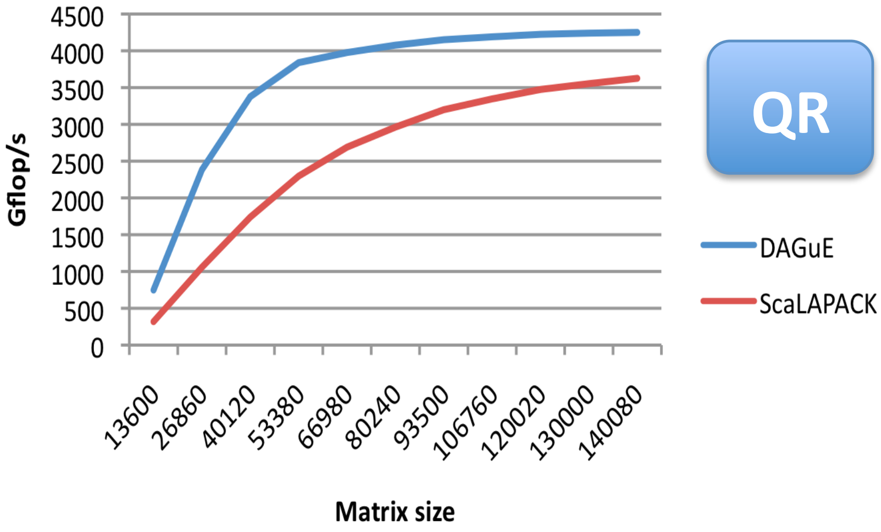
DSBP =
Distributed Square
Block Packed

81 nodes
Dual socket nodes
Quad core Xeon L5420
Total 648 cores at 2.5 GHz
ConnectX InfiniBand DDR 4x

LU

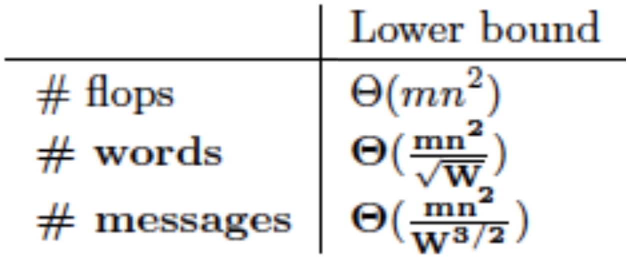


QR



Communication Avoiding Algorithms

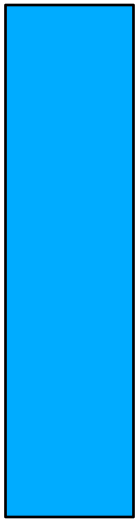
- Goal: Algorithms that communicate as little as possible
- Jim Demmel and company have been working on algorithms that obtain a provable minimum communication.
- Direct methods (BLAS, LU, QR, SVD, other decompositions)
  - Communication lower bounds for *all* these problems
  - Algorithms that attain them (*all* dense linear algebra, some sparse)
- Iterative methods – Krylov subspace methods for Ax=b, Ax=λx
  - Communication lower bounds, and algorithms that attain them (depending on sparsity structure)
- For QR Factorization they can show:



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

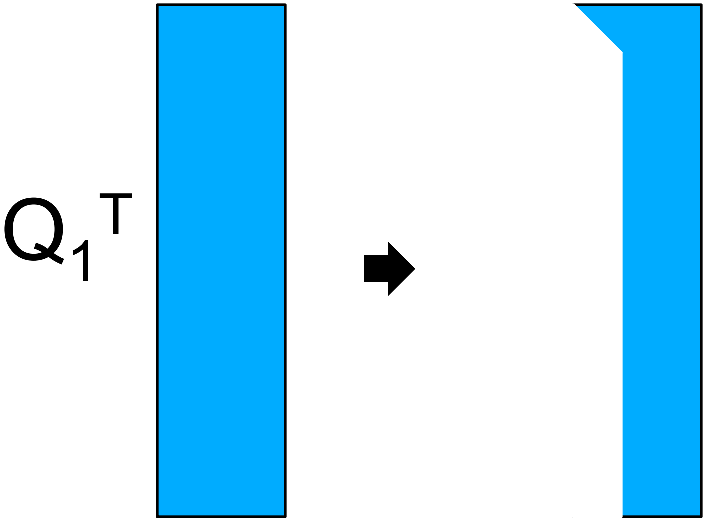
Standard QR Block Reduction

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



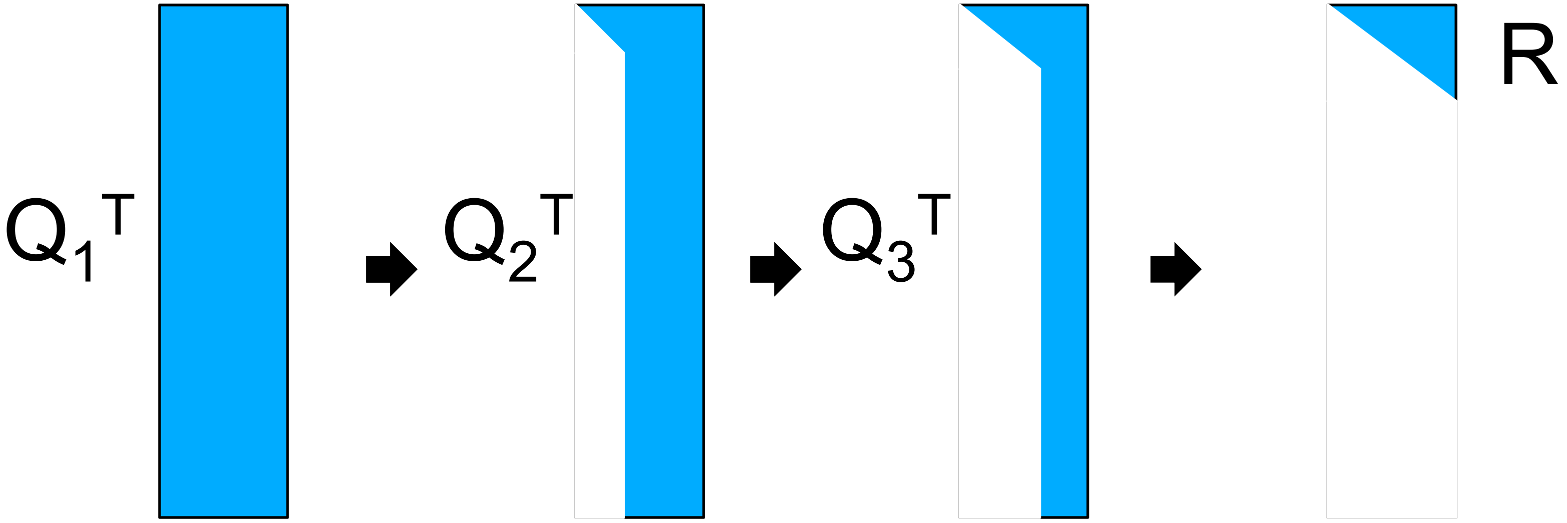
Standard QR Block Reduction

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



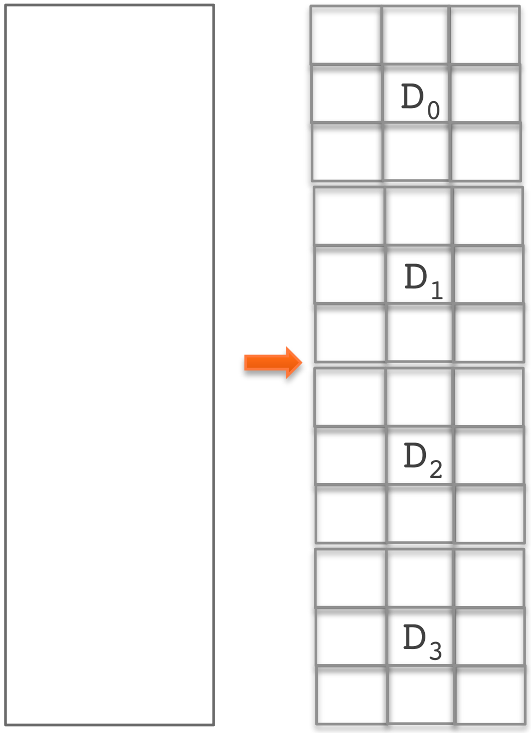
Standard QR Block Reduction

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



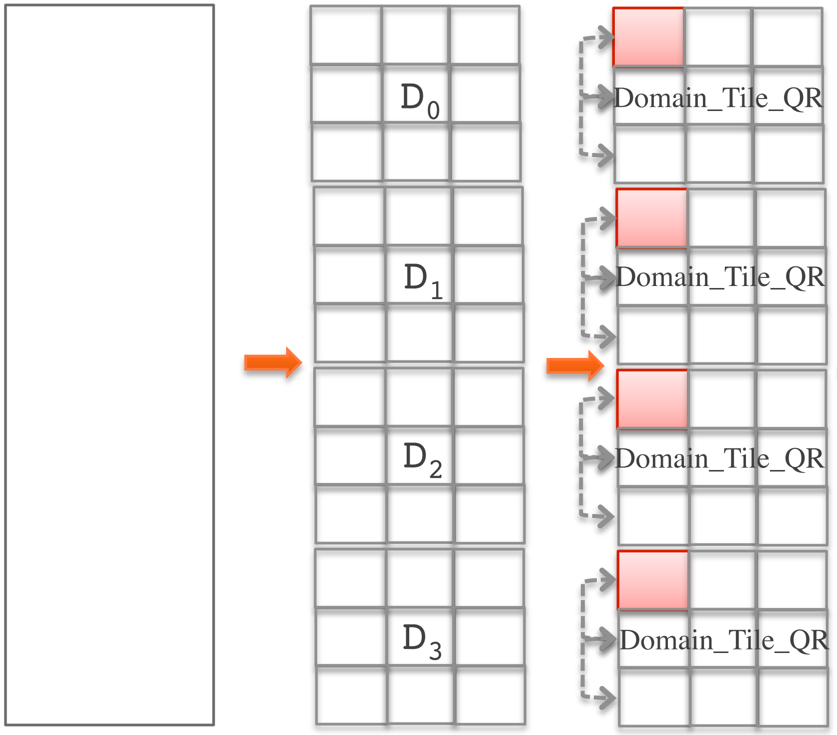
$$A = Q_1 Q_2 Q_3 R = QR$$

Communication Avoiding QR Example



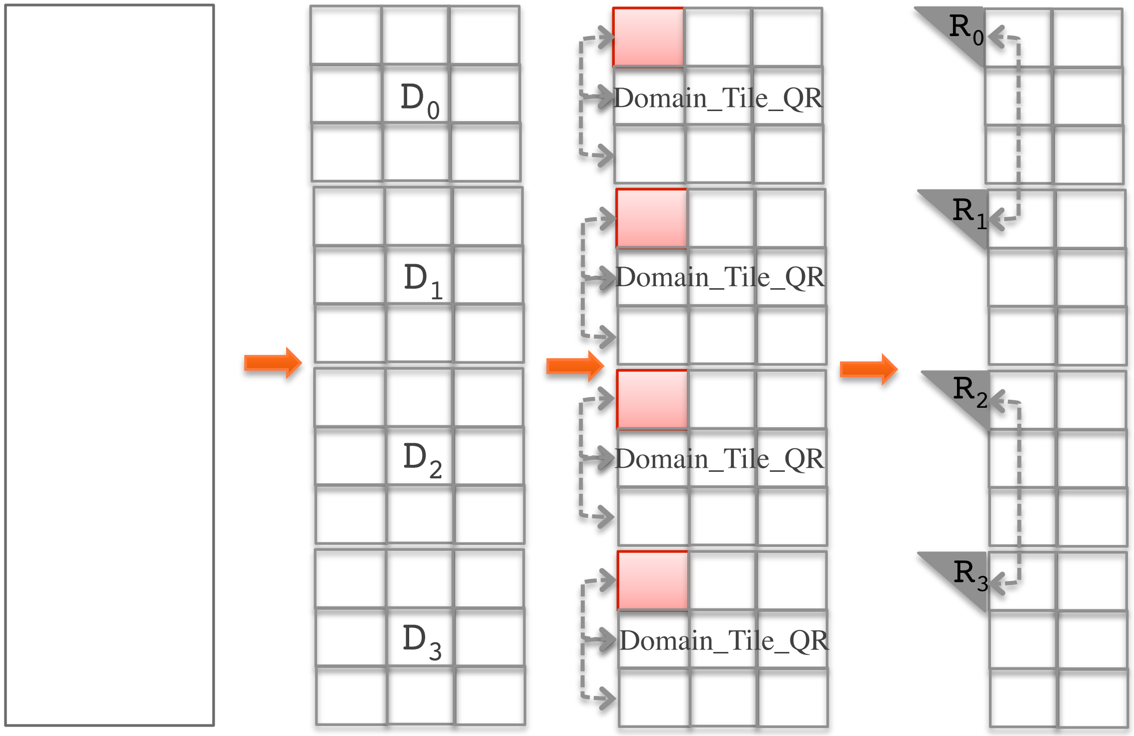
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Avoiding QR Example



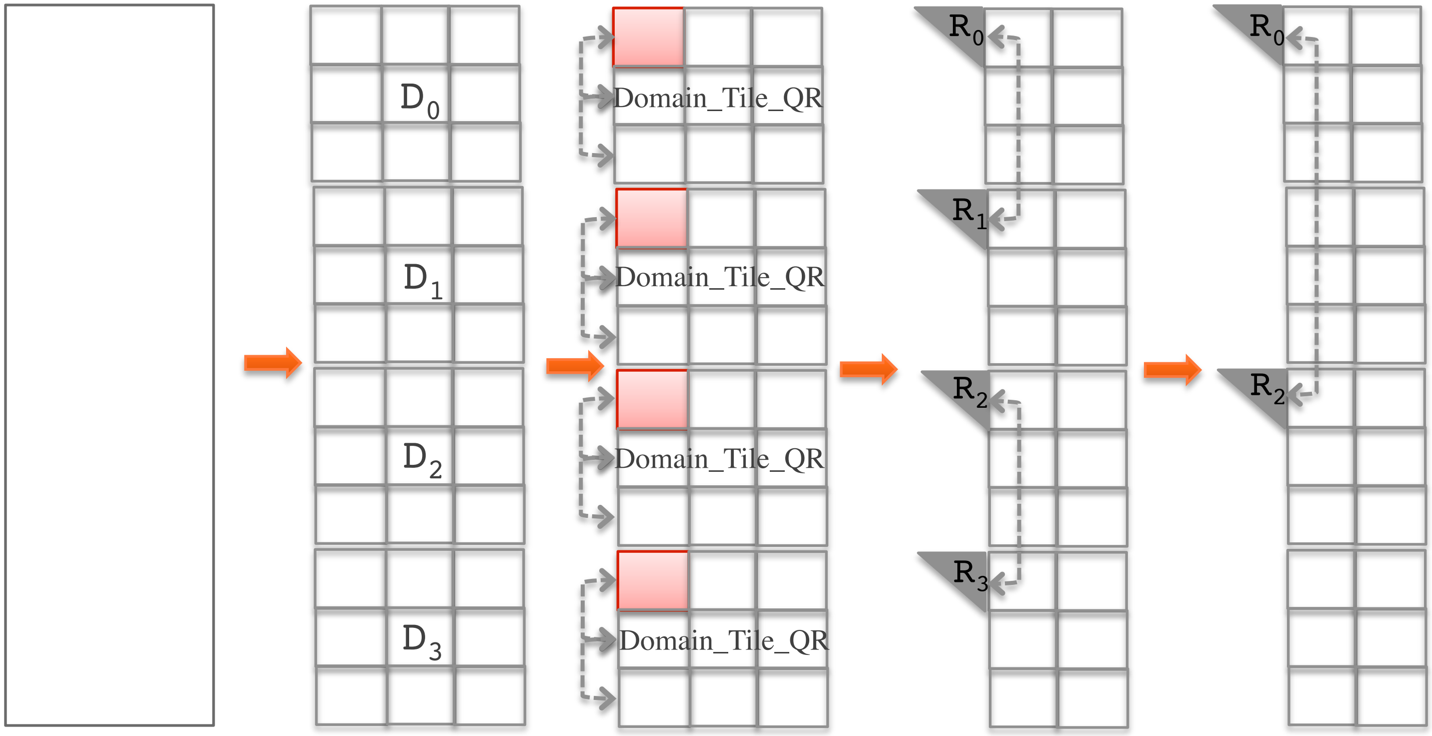
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Avoiding QR Example



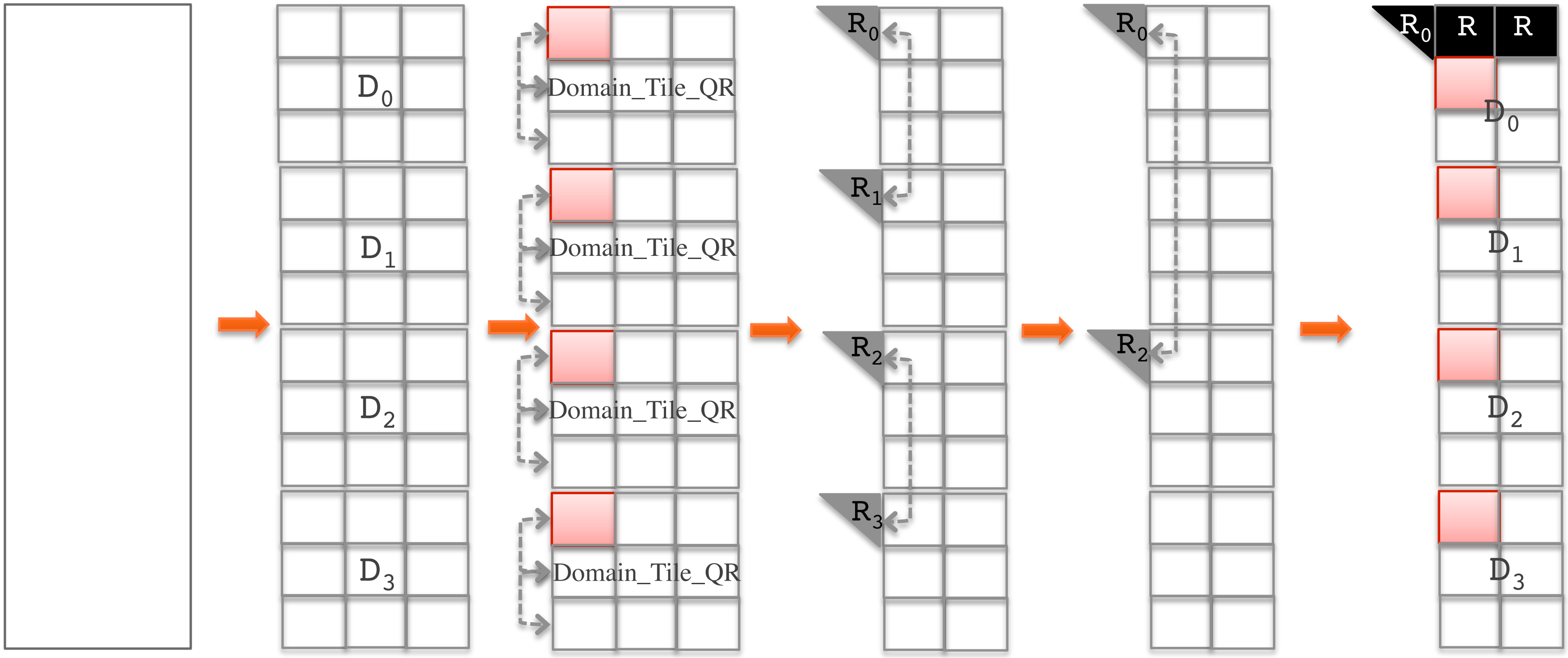
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Avoiding QR Example



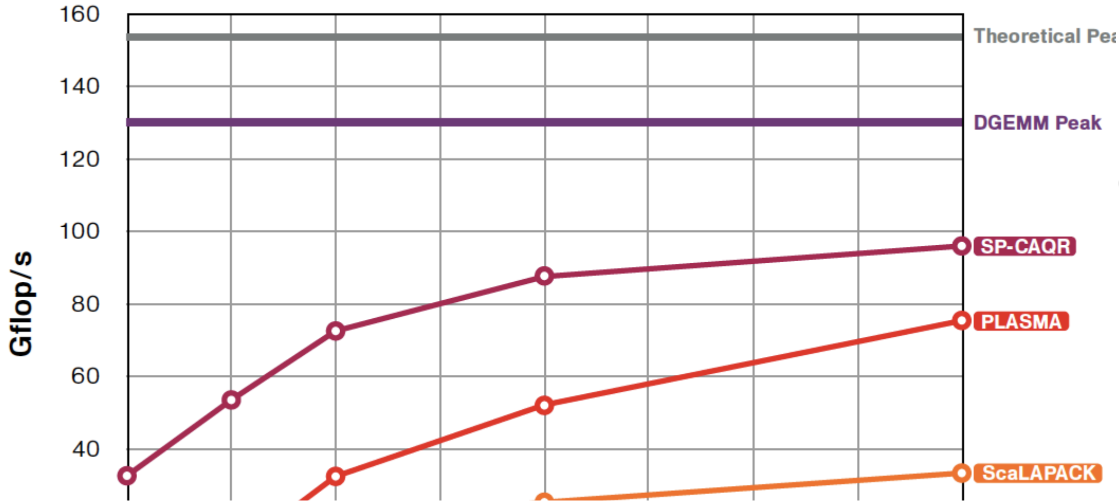
A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

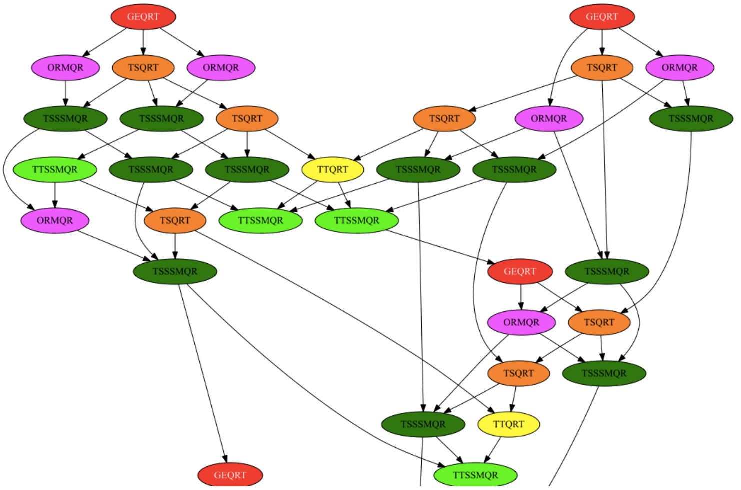
Communication Avoiding QR Example

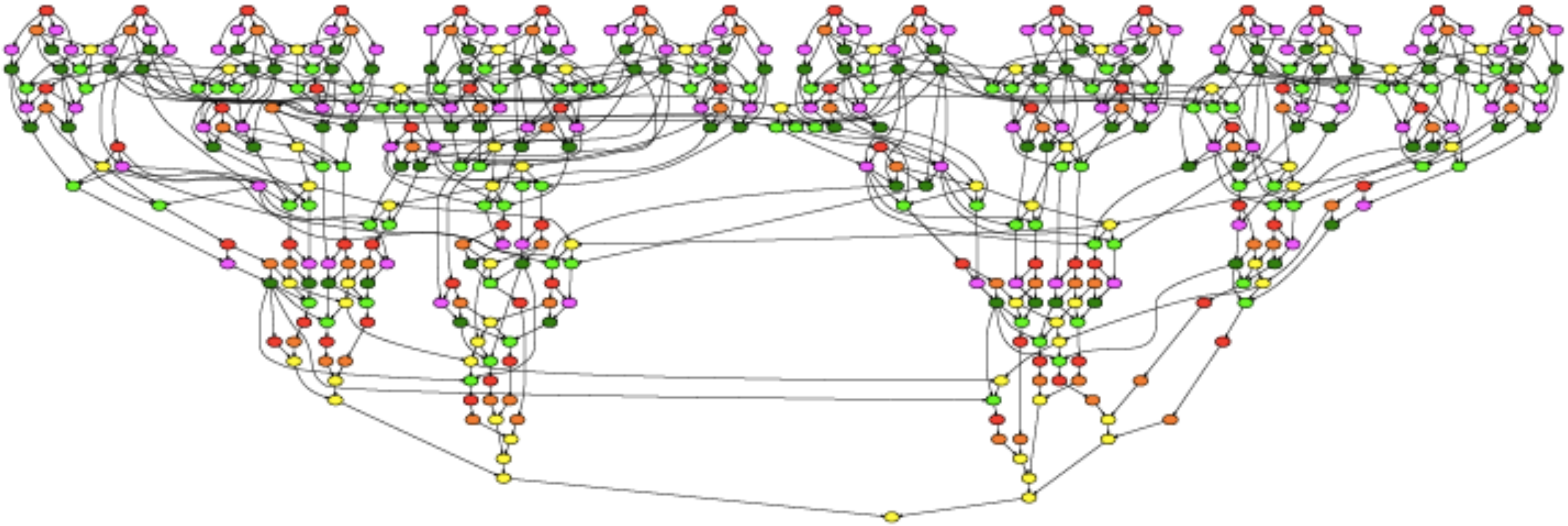


A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

Communication Reducing QR Factorization







Mixed Precision Methods

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

Mixed Precision

- Single Precision is 2X faster than Double Precision
- With some GP-GPUs a factor of 8x
- Power saving issues
- Reduced data motion
  - 32 bit data instead of 64 bit data
- Higher locality in cache
  - More data items in cache

Idea Goes Something Like This…

- Exploit 32 bit floating point as much as possible.
  - Especially for the bulk of the computation
- Correct or update the solution with selective use of 64 bit floating point to provide a refined results
- Intuitively:
  - Compute a 32 bit result,
  - Calculate a correction to 32 bit result using selected higher precision and,
  - Perform the update of the 32 bit results with the correction using high precision.

Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, *Ax* = *b*, can work this way.

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

- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.

Mixed-Precision Iterative Refinement

- Iterative refinement for dense systems, *Ax = b*, can work this way.

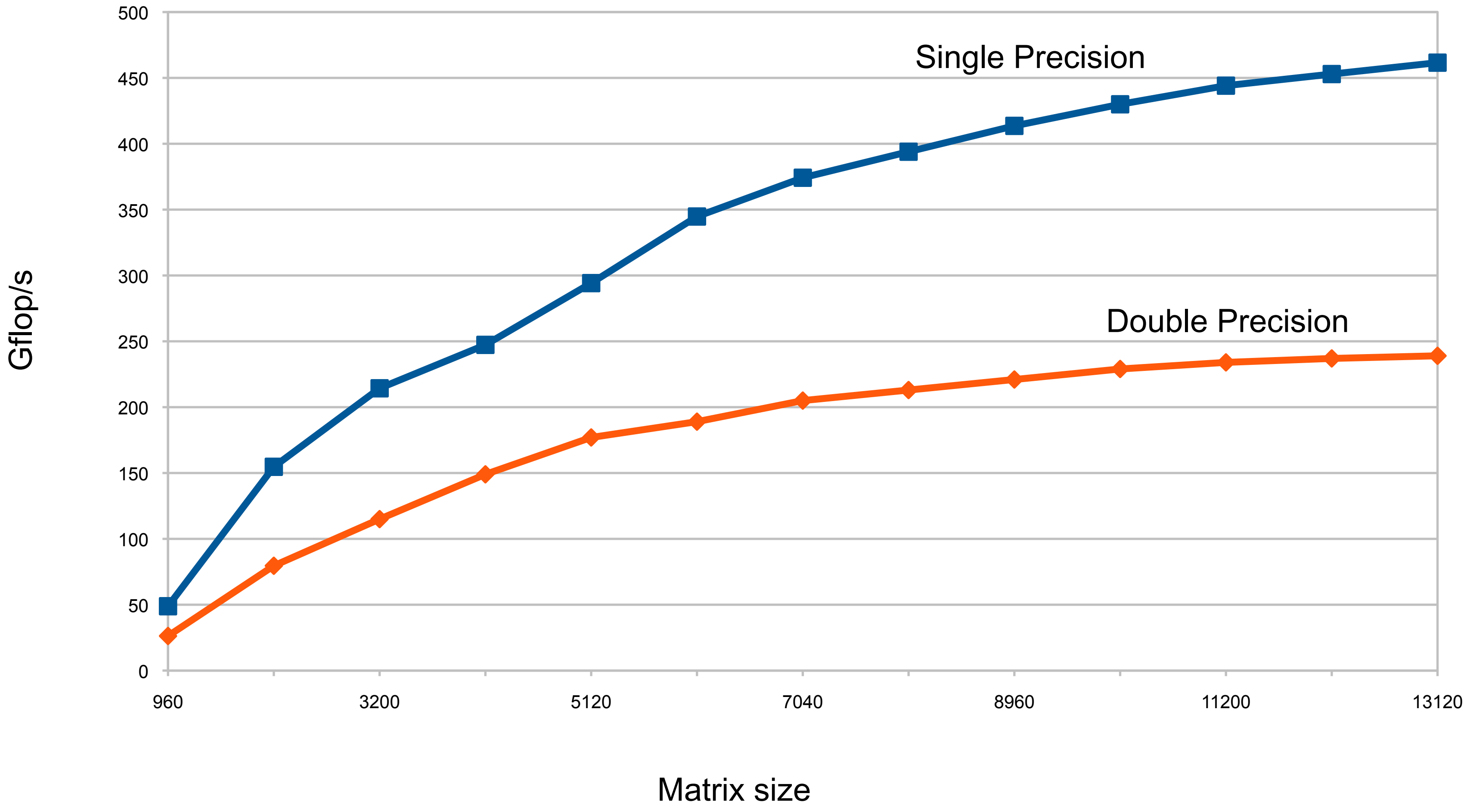
```
L U = lu(A)                      SINGLE     $O(n^3)$
x = L\(U\b)                      SINGLE     $O(n^2)$
r = b – Ax                       DOUBLE     $O(n^2)$
WHILE || r || not small enough
        z = L\(U\r)              SINGLE     $O(n^2)$
        x = x + z                DOUBLE     $O(n^1)$
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

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

- Requires extra storage, total is 1.5 times normal;
- $O(n^3)$ work is done in lower precision
- $O(n^2)$ work is done in high precision
- Problems if the matrix is ill-conditioned in sp; $O(10^8)$

Ax = b

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

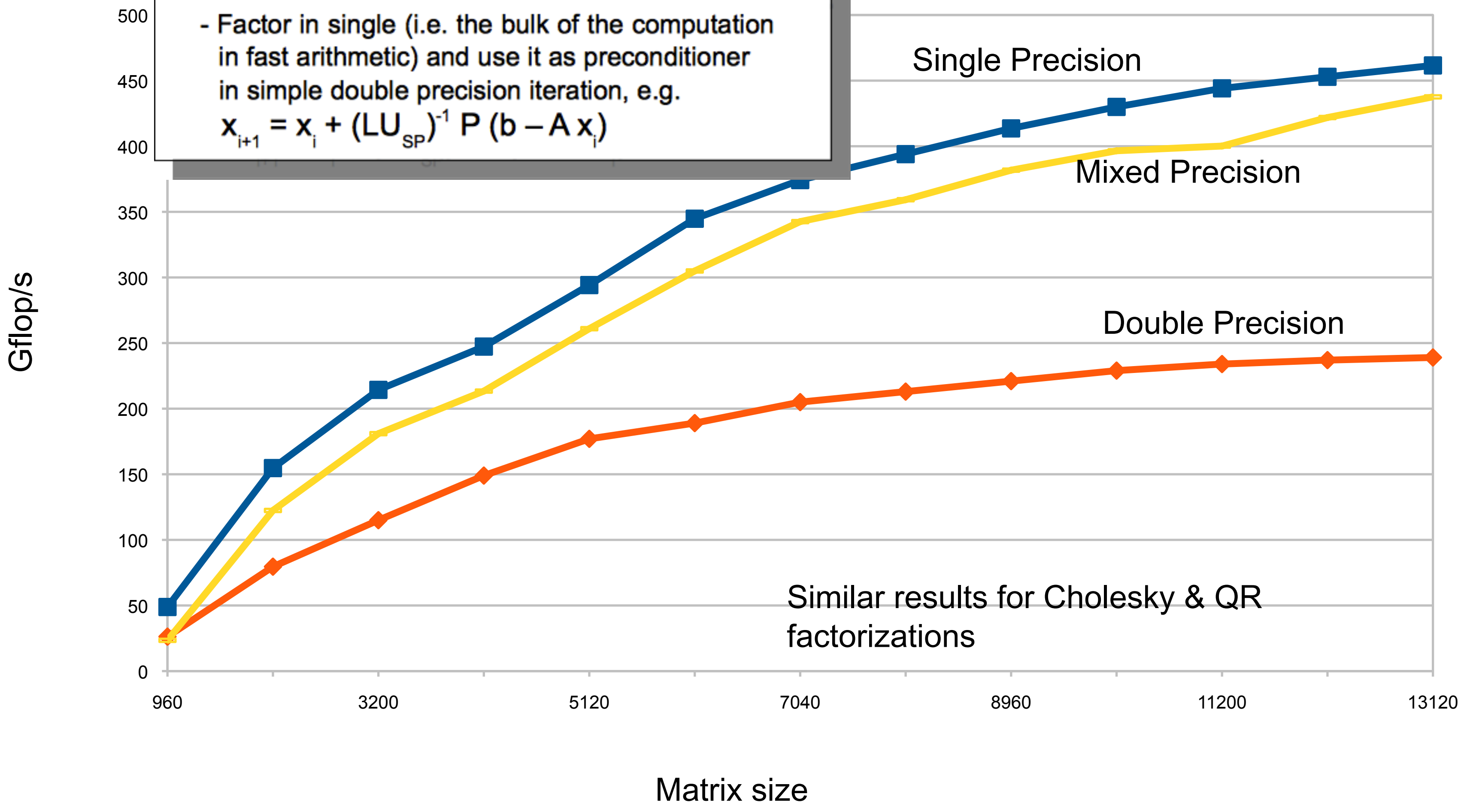


Ax = b

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

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

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



Similar results for Cholesky & QR factorizations

Exploiting Mixed Precision Computations

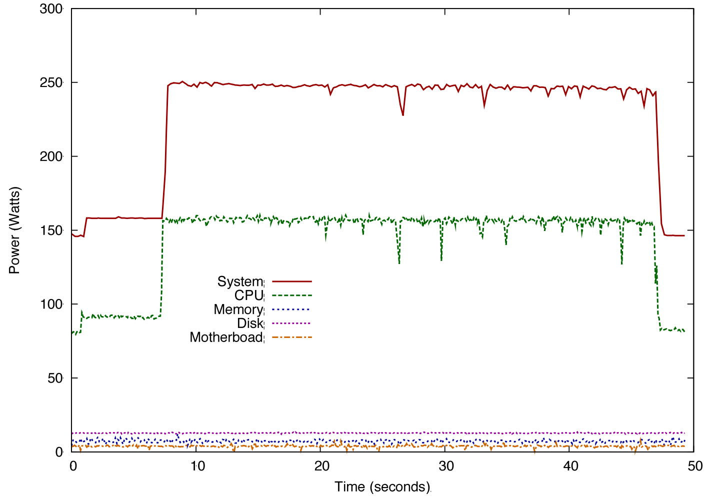
- Single precision is faster than DP because:
  - **Higher parallelism within floating point units**
    - 4 ops/cycle (usually) instead of 2 ops /cycle
  - **Reduced data motion**
    - 32 bit data instead of 64 bit data
  - **Higher locality in cache**
    - More data items in cache

Power Profiles

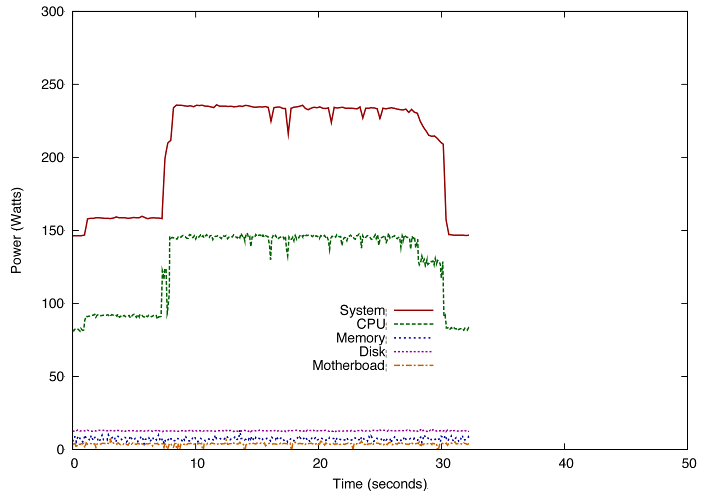
Two dual-core 1.8 GHz AMD Opteron processors
Theoretical peak: 14.4 Gflops per node
DGEMM using 4 threads: 12.94 Gflops
PLASMA 2.3.1, GotoBLAS2
Experiments:
PLASMA LU solver in double precision
PLASMA LU solver in mixed precision

N = 8400, using 4 cores	PLASMA DP	PLASMA Mixed
Time to Solution (s)	39.5	22.8
GFLOPS	10.01	17.37
Accuracy $\frac{\| Ax-b \|}{(\| A \|\| X \| + \| b \|)N\varepsilon}$	2.0E-02	1.3E-01
Iterations		7
System Energy (KJ)	10852.8	6314.8

PLASMA DP

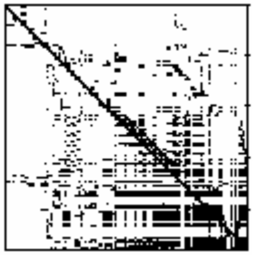


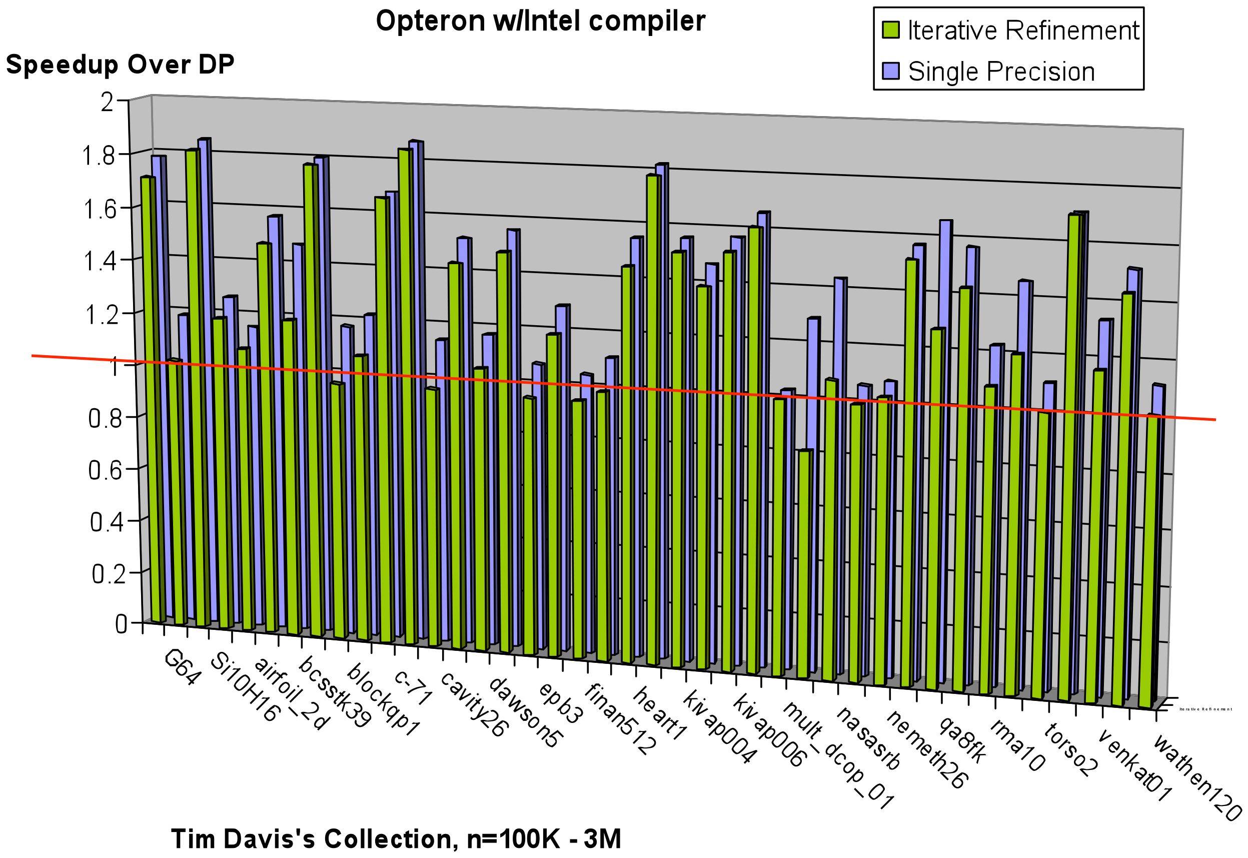
PLASMA Mixed Precision



Sparse Direct Solver and Iterative Refinement

MUMPS package based on multifrontal approach which generates small dense matrix multiplies



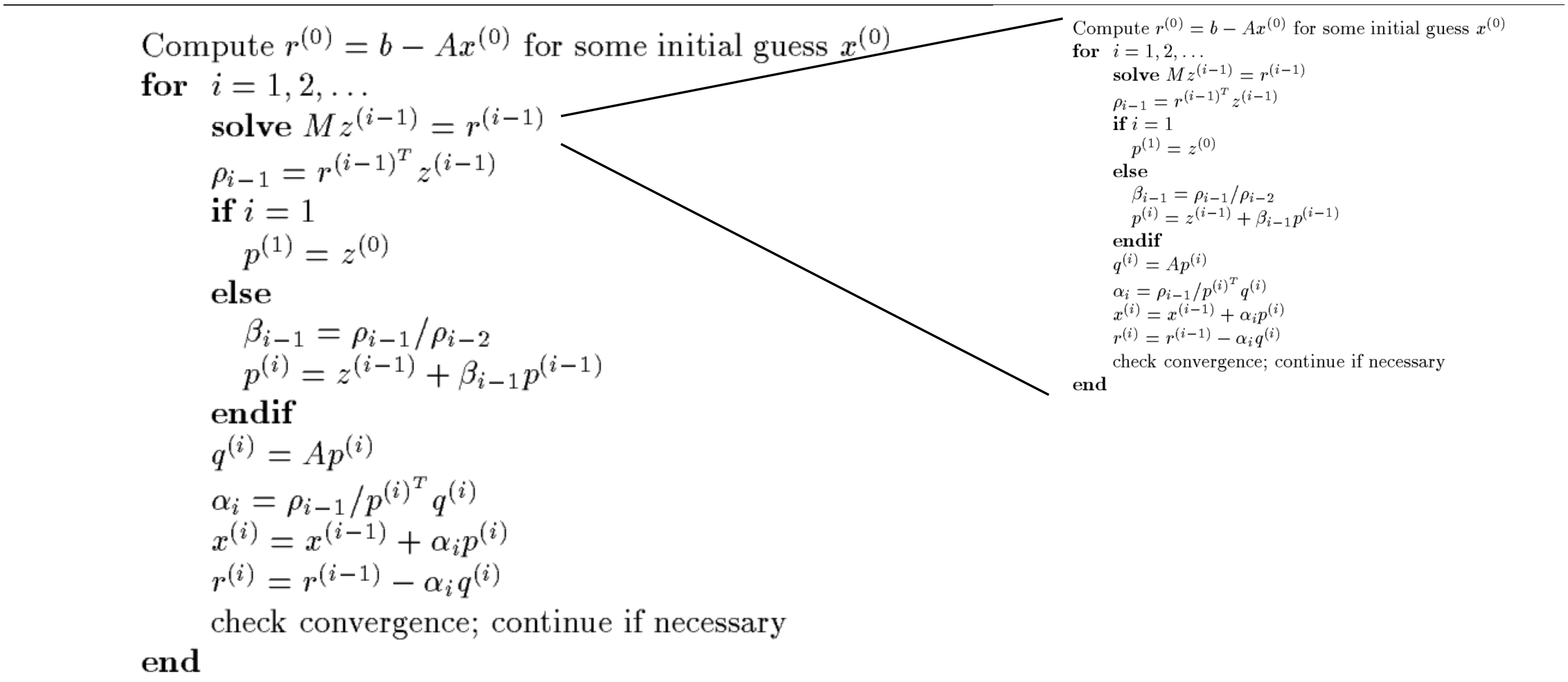


Sparse Iterative Methods (PCG)

- **Outer/Inner Iteration**

Outer iterations using 64 bit floating point

Inner iteration:
In 32 bit floating point

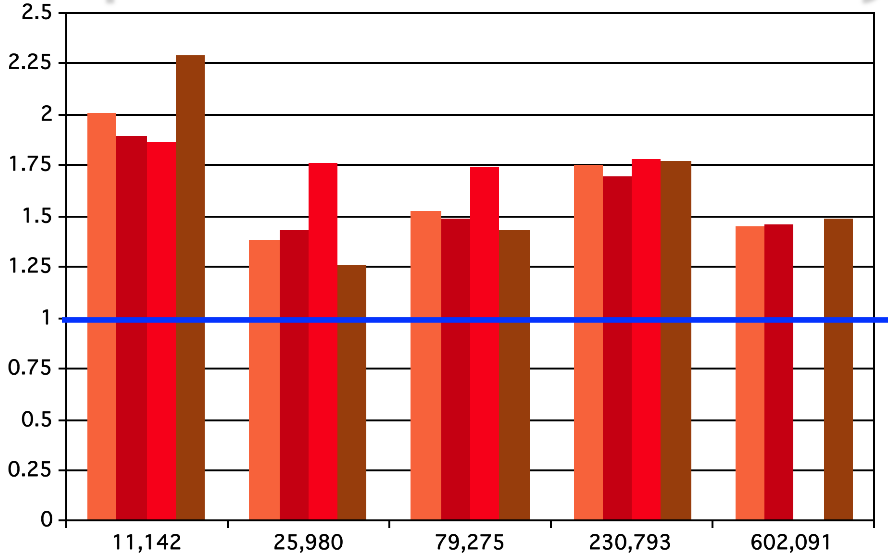


```
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for  $i = 1, 2, \ldots$
    solve $M z^{(i-1)} = r^{(i-1)}$
    $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
    if $i = 1$
        $p^{(1)} = z^{(0)}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
    $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence; continue if necessary
end
```

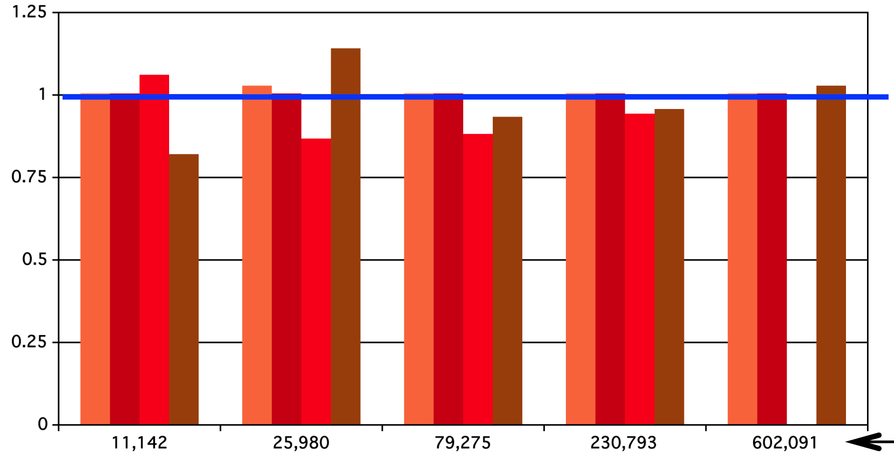
```
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for  $i = 1, 2, \ldots$
    solve $M z^{(i-1)} = r^{(i-1)}$
    $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
    if $i = 1$
        $p^{(1)} = z^{(0)}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
    $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence; continue if necessary
end
```

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

Mixed Precision Computations for Sparse Inner/Outer-type Iterative Solvers



Speedups for mixed precision
Inner SP/Outer DP (SP/DP) iter. methods *vs* DP/DP
(CG^2, $GMRES^2$, PCG^2, and $PGMRES^2$ with diagonal prec.)
(***Higher is better***)



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

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

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

← Matrix size

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

Matrix Algebra on GPU and Multicore Architectures

- MAGMA: **a new generation linear algebra (LA) libraries** to achieve the fastest possible time to an accurate solution **on hybrid/heterogeneous architectures**, starting with current multicore+MultiGPU systems
  Homepage: http://icl.cs.utk.edu/magma/

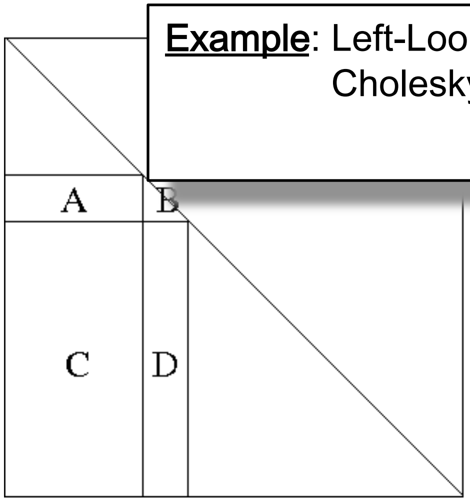
- MAGMA & LAPACK
  - MAGMA - based on LAPACK and extended for hybrid systems (multi-GPUs + multicore systems);
  - MAGMA - designed to be similar to LAPACK in functionality, data storage and interface, in order to allow scientists to effortlessly port any of their LAPACK-relying software components to take advantage of the new architectures
  - MAGMA - to leverage years of experience in developing open source LA software packages and systems like LAPACK, ScaLAPACK, BLAS, ATLAS as well as the newest LA developments (e.g. communication avoiding algorithms) and experiences on homogeneous multicores (e.g. PLASMA)

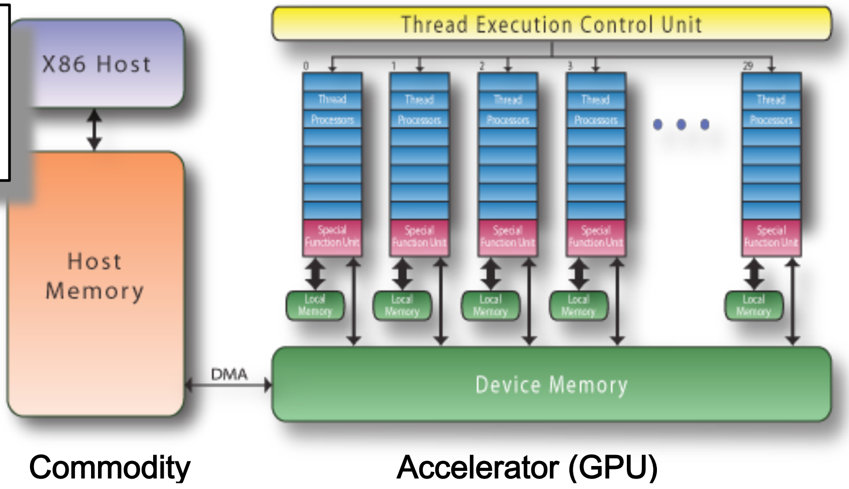
- Support
  - NSF, NVIDIA [CUDA Center of Excellence **at UTK** on the development of **Linear Algebra Libraries for CUDA-based Hybrid Architectures**]

- MAGMA developers
  - University of Tennessee, **Knoxville**; University of California, **Berkeley**; University of Colorado, **Denver**

One-Sided Dense Matrix Factorizations (LU, QR, and Cholesky) from MAGMA

Example: Left-Looking Hybrid
Cholesky factorization





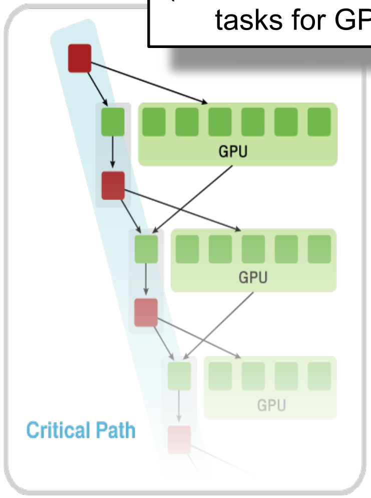
MATLAB code	LAPACK code	Hybrid code
(1) B = B – A*A'	ssyrk_(“L”, “N”, &nb, &j, &mone, hA(j,0), ...)	cublasSsyrk('L', 'N', nb, j. mone, dA(j,0), ...)
		cublasGetMatrix(nb, nb, 4, dA(j, j), *lda, hwork, nb)
(2) B = chol(B, ‘lower’)	spotrf_(“L”, &nb, hA(j, j), lda, info)	cublasSgemm(‘N’, ‘T’, j, …)
(3) D = D – C*A’	sgemm_(“N”, “T”, &j, …)	spotrf_(“L”, &nb, hwork, &nb, info)
		cublasSetMatrix(nb, nb, 4, hwork, nb, dA(j, j), *lda)
(4) D = B\D	strsm_(“R”, “L”, “T”, “N”, &j, …)	cublasStrsm(‘R’, ‘L’, ‘T’, ‘N’, j, …)

CUDA implementation:

- a_ref points to the GPU memory
- GPU kernels are started asynchronously which results in overlapping the GPU sgemm with transferring T to the CPU, factoring it, and sending the result back to the GPU

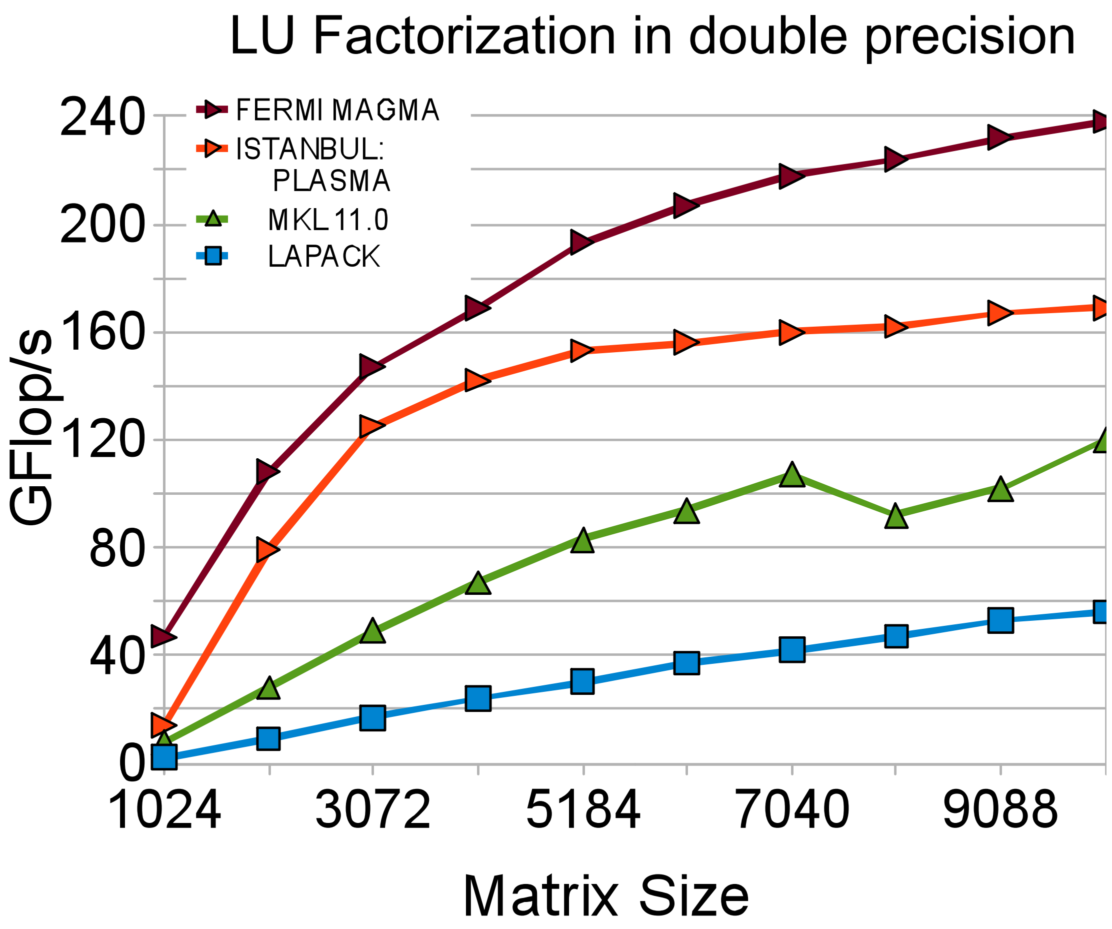
Hybridization Methodology

- MAGMA uses HYBRIDIZATION methodology based on
  - Representing linear algebra algorithms as collections of TASKS and DATA DEPENDENCIES among them
  - Properly SCHEDULING the tasks' execution over the multicore and the GPU hardware components
- Successfully applied to fundamental linear algebra algorithms
  - One and two-sided factorizations and solvers
  - Iterative linear and eigen-solvers
- Faster, cheaper, better ?
  - High-level
  - Leveraging prior developments
  - Exceeding in performance homogeneous solutions



Hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)

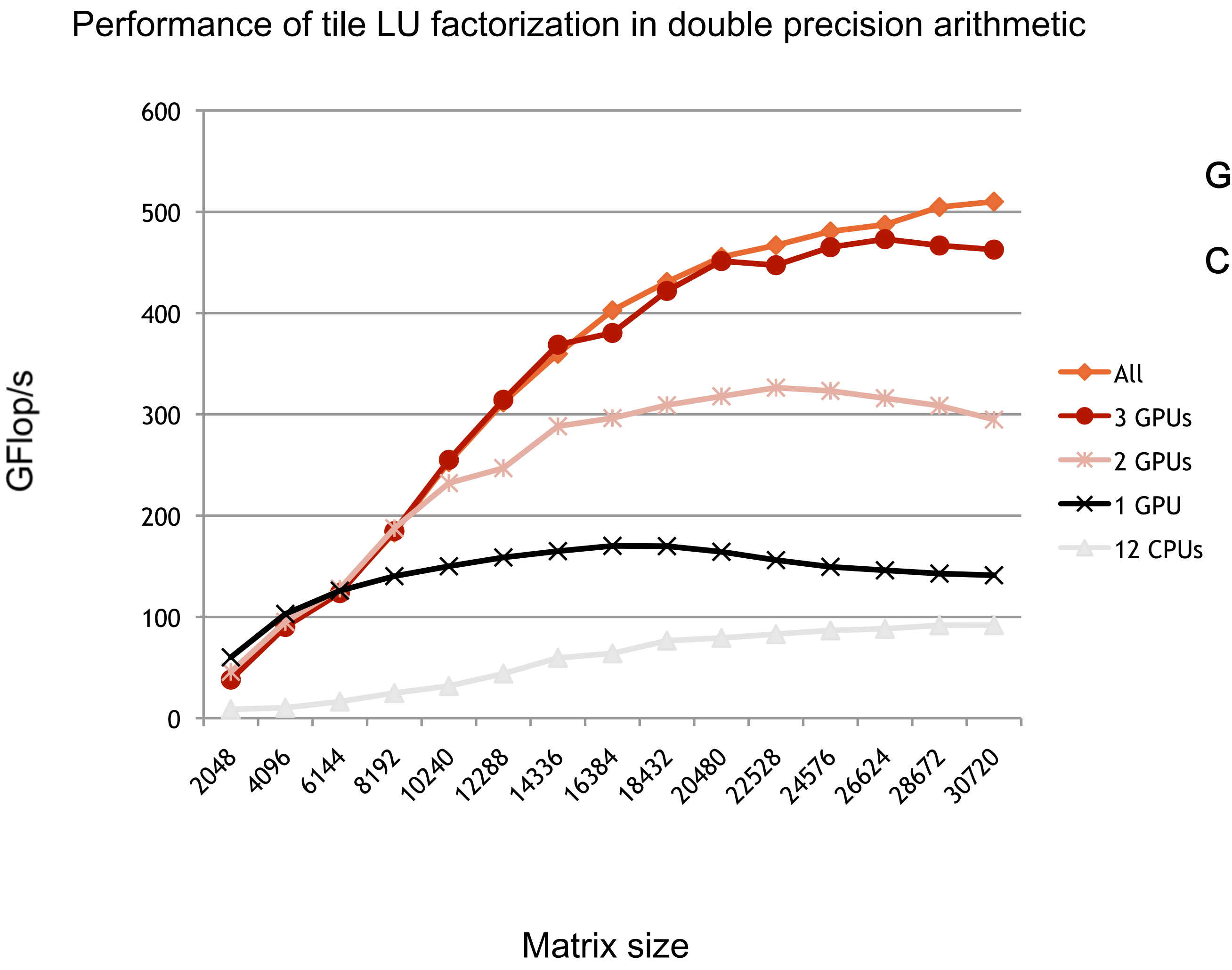
Results – one sided factorizations



FERMI	Tesla C2050: 448 CUDA cores @ 1.15GHz SP/DP peak is 1030 / 515 GFlop/s
ISTANBUL	AMD 8 socket 6 core (48 cores) @2.8GHz SP/DP peak is 1075 / 538 Gflop/s

- Similar results for Cholesky & QR
- 60% faster than the commercially available CULA library for GPUs

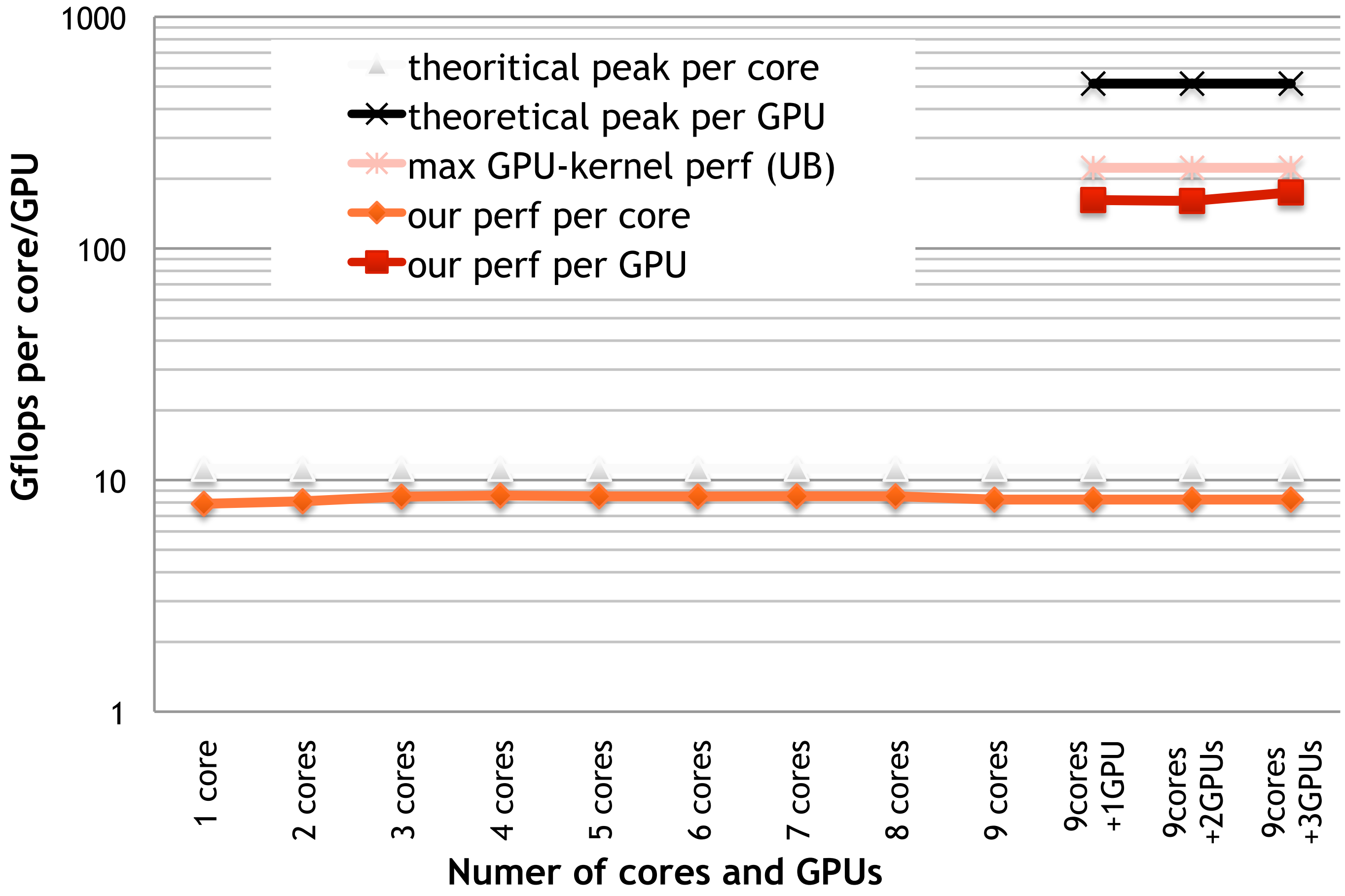
Tile LU factorization on multi-GPUs



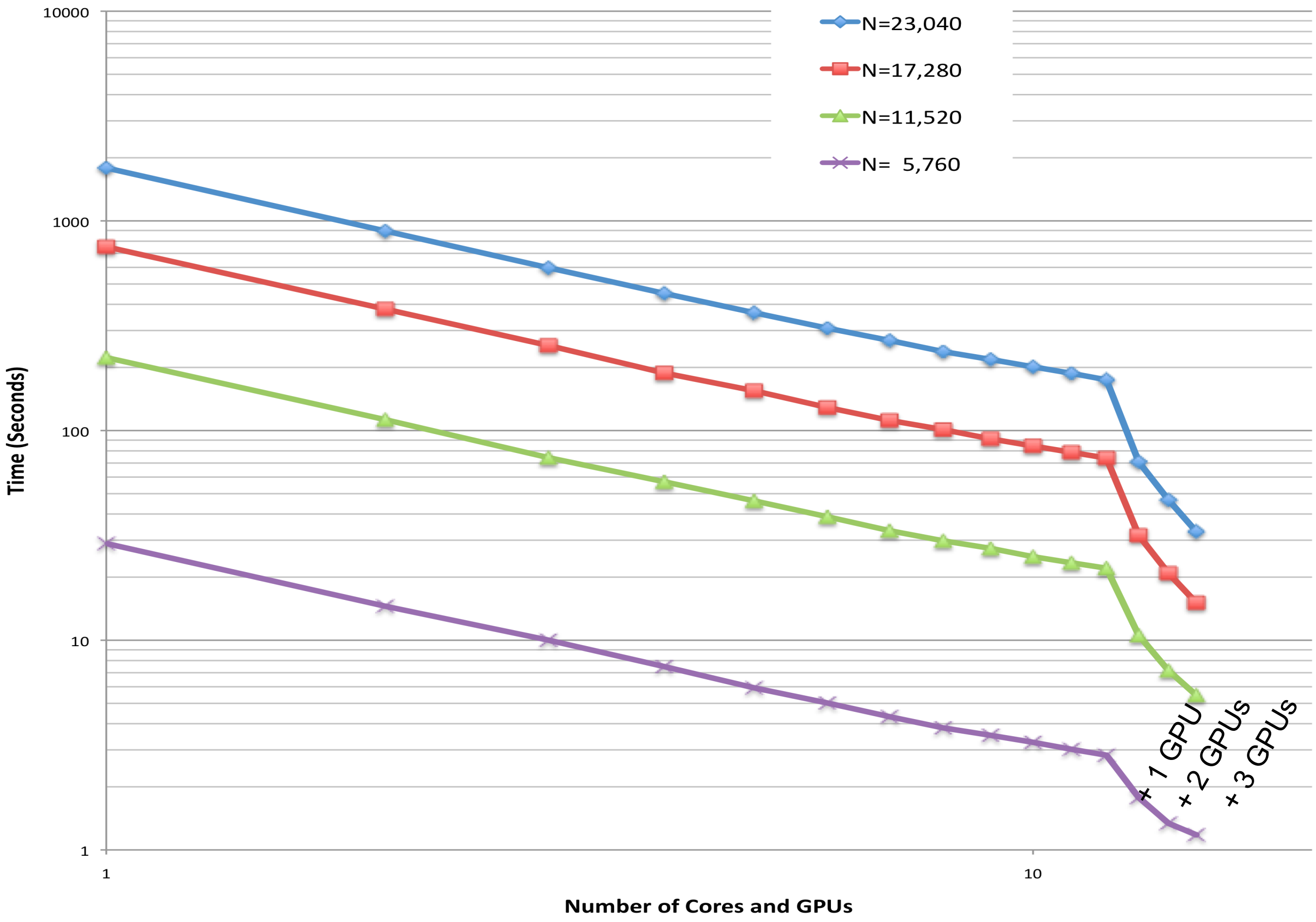
GPU : Fermi C2050
(14 processors @1.14GHz)

CPU : dual-socket hexa-core
Intel Nehalem @2.67GHz

QR (DP): Weak Scalability

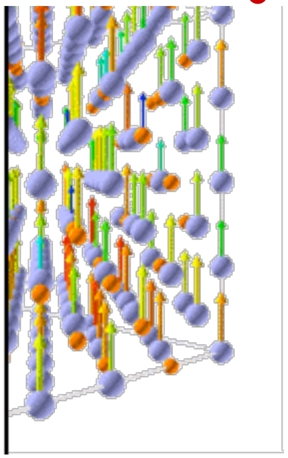


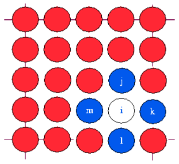
QR (DP): Strong Scalability



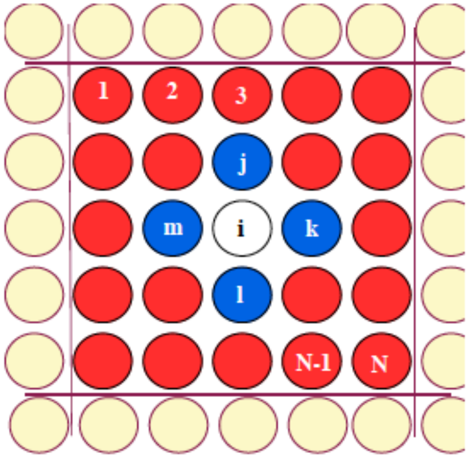
Locally-Self-Consistent Multiple -Scattering

- The LSMS Code is a first-principles computer model that simulates the interactions between electrons and atoms in magnetic materials.
- LSMS is a real-space multiple scattering, Green-function-based method.
- First app to reach TeraFlop and PetaFlop

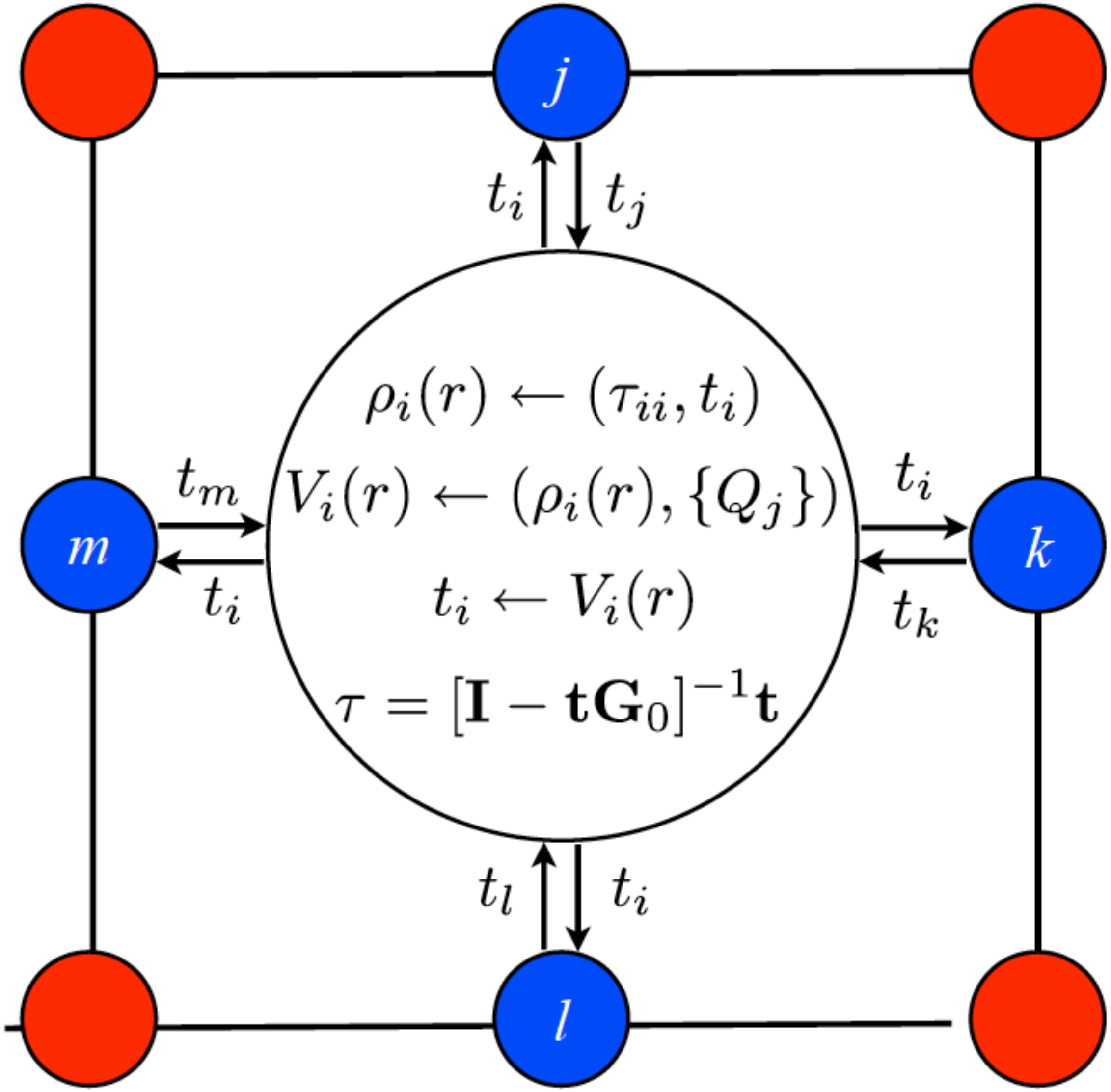




A parallel implementation and scaling of the LSMS method: perfectly scalable at high performance



- Need only block i of τ
- $\left(\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right)^{-1} = \left(\begin{array}{c|c} (A - BD^{-1}C)^{-1} & * \\ \hline * & * \end{array}\right)$
- Calculation dominated by ZGEMM
- Sustained performance similar to Linpack



$$\rho_i(r) \leftarrow (\tau_{ii}, t_i)$$

$$V_i(r) \leftarrow (\rho_i(r), \{Q_j\})$$

$$t_i \leftarrow V_i(r)$$

$$\tau = [\mathbf{I} - \mathbf{t}\mathbf{G}_0]^{-1}\mathbf{t}$$

Complex Multiplication

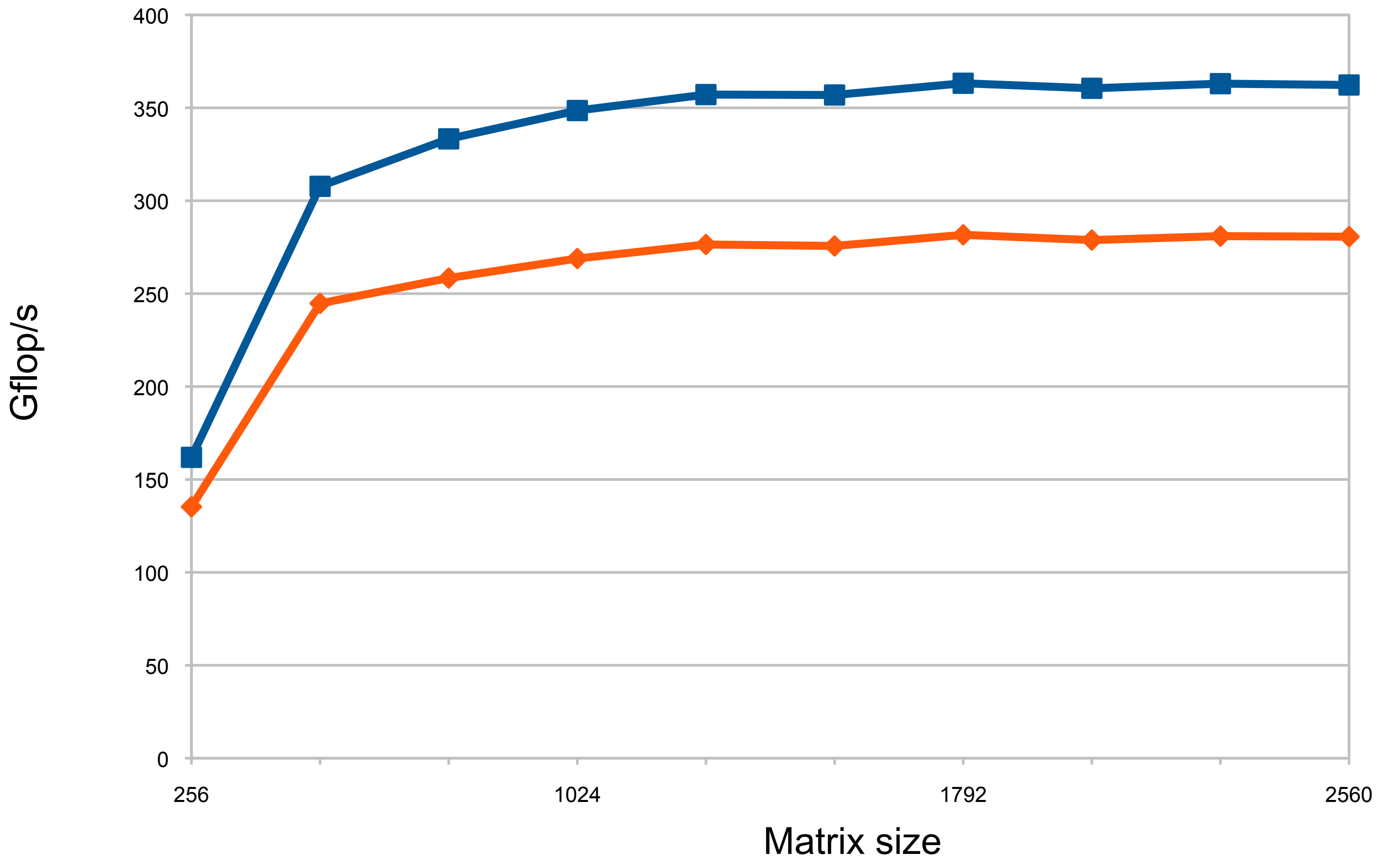
- The product (a + b*i*) · (c + d*i*) normally requires 4 multiplications and 2 additions
  - Real part = a · c - b · d
  - Imaginary part = a · d + b · c
- But it can be calculated in the following way, using 3 multiplications and 5 additions.
  - k1 = c · (a + b)
  - k2 = a · (d – c)
  - k3 = b · (c + d)
  - Real part = k1 – k3
  - Imaginary part = k1 + k2
- Resulting in 1 less multiplication and 3 more additions
- Can be applied to matrices resulting in a 25% reduction in operation count for ZGEMM.
  - Remove $2 \cdot n^3$ operations in exchange for adding $3 \cdot n^2$ operations.

No Free Lunch

- Need extra storage, $2n^2$
- The imaginary part may be contaminated by relative errors much larger than those for conventional multiplication.
  - However if the errors are measured relative to

    $||A||*||B||$ then they are just as small as for conventional multiplication.

ZGEMM on Nvidia Fermi



3 mults & 5 adds
Version of
ZGEMM

Conventional call
To ZGEMM
4 mults & 2 adds

Nvidia C2050 (Fermi): 448 CUDA cores @ 1.15GHz,
theoretical SP peak is 1.03 Tflop/s, DP peak 515 GFlop/s)

PLASMA 3.0 (June 6)

Functionality	Coverage
linear systems and least squares	LU (LAPACK pivoting)★ , LDLT (no pivoting)★ Cholesky, QR & LQ
mixed-precision linear systems	LU, Cholesky, QR
tall and skinny factorization	QR & LQ
Q matrix generation & application	QR & LQ, tall and skinny QR & LQ★
explicit matrix inversion	Cholesky
symmetric EVP	eigenvalues only (no vectors)★
SVD	singular values only (no vectors)★
Level 3 BLAS (tile layout)	GEMM, HEMM, HER2K, HERK, SYMM, SYR2K, SYRK, TRMM, TRSM (complete set)
In-place layout translations	CM, RM, CCRB, CRRB, RCRB, RRRB (all combinations)

Features
Covering four precisions: Z, C, D, S (and mixed-precision: ZC, DS)
Static scheduling and dynamic scheduling with QUARK
Support for Linux, MS Windows, Mac OS and AIX

PLASMA 3.x (November 15)

Functionality	Coverage
symmetric EVP	eigenvalues & eigenvectors
SVD	singular values & singular vectors
LDLT	pivoting

Features
GPU acceleration (factorizations, reductions to band forms, BLAS 3)

MAGMA 1.0 (June 6)

Functionality	Coverage
linear systems and least squares	LU, Cholesky, QR & LQ
mixed-precision linear systems	LU, Cholesky, QR
Q matrix generation & application	QR & LQ, tall and skinny QR & LQ
general EVP	1st and 3rd stage have GPU acceleration; 2nd stage is LAPACK on multicore
symmetric EVP	1st and 3rd stage have GPU acceleration; 2nd stage is LAPACK on multicore
SVD	1st and 3rd stage have GPU acceleration; 2nd stage is LAPACK on multicore
BLAS	GEMM, TRSM, GEMV, SYMV, auxiliary routines, etc., for both Tesla and Fermi

Features
Covering four precisions: Z, C, D, S (and mixed-precision: ZC, DS)
Static scheduling for single NVIDIA GPU
Support for Linux, MS Windows, and Mac OS

MAGMA 1.x (November 15)

Functionality	Coverage
SpMV	Different matrix formats
Krylov space methods	PCG, GMRES, BiCG, LOBPCG, etc.
LAPACK for Multicore	LU, QR, and Cholesky w/ QUARK
Multicore + GPU algorithms	LU, QR, and Cholesky w/ QUARK
Multicore + multiGPU algorithms	LU, QR, and Cholesky w/ StarPU

Futures

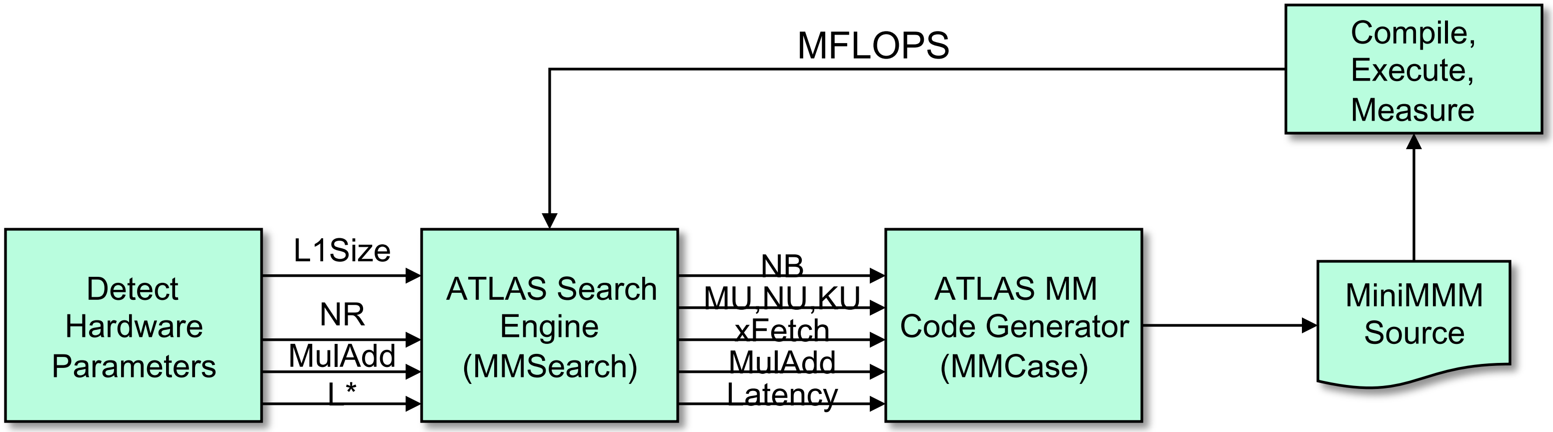
- Develop implementations for two-sided factorizations: bi-diagonalization, Hessenberg reduction, etc.
- Develop accurate performance models
- Develop auto-tuning framework
- Propose mechanisms for static and dynamic scheduling of work in a multi-socket multi-blade hybrid system
- Investigate impact of memory architectures on different algorithmic approaches to common linear algebra workloads

Automatic Performance Tuning

- Writing high performance software is hard
- Ideal: get high fraction of peak performance from one algorithm
- Reality: Best algorithm (and its implementation) can depend strongly on the problem, computer architecture, compiler,…
  - Best choice can depend on knowing a lot of applied mathematics and computer science
  - Changes with each new hardware, compiler release
- Automatic performance tuning
  - Use machine time in place of human time for tuning
  - Search over possible implementations
  - Use performance models to restrict search space
  - Past successes: ATLAS, FFTW, Spiral, Open-MPI

How to Deal with Complexity?

- Many parameters in the code needs to be optimized.
- Software adaptivity is the key for applications to effectively use available resources whose complexity is exponentially increasing


MFLOPS
Compile, Execute, Measure
Detect Hardware Parameters
L1Size
NR
MulAdd
L*
ATLAS Search Engine (MMSearch)
NB
MU,NU,KU
xFetch
MulAdd
Latency
ATLAS MM Code Generator (MMCase)
MiniMMM Source

Auto-Tuning

- Best algorithm implementation can depend strongly on the problem, computer architecture, compiler,…
- There are 2 main approaches
  - Model-driven optimization
    [Analytical models for various parameters;
    Heavily used in the compilers community;
    May not give optimal results]
  - Empirical optimization
    [Generate large number of code versions and runs them on a given platform to determine the best performing one;
    Effectiveness depends on the chosen parameters to optimize and the search heuristics used]
- Natural approach is to combine them in a hybrid approach
  [1st model-driven to limit the search space for a 2nd empirical part]
  [Another aspect is adaptivity – to treat cases where tuning can not be restricted to optimizations at design, installation, or compile time]

Reproducibility

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

A Call to Action: Exascale is a Global Challenge

- Hardware has changed dramatically while software ecosystem has remained stagnant
- Community codes unprepared for sea change in architectures
- No global evaluation of key missing components
- The IESP was Formed in 2008
- Goal to engage international computer science community to address common software challenges for Exascale
- Focus on open source systems software that would enable multiple platforms
- Shared risk and investment
- Leverage international talent base

International Exascale Software Program

Improve the world’s simulation and modeling capability by improving the coordination and development of the HPC software environment

Workshops:

Build an international plan for coordinating research for the next generation open source software for scientific high-performance computing

www.exascale.org

Roadmap Components

www.exascale.org

4.1 Systems Software

- 4.1.1 Operating systems
- 4.1.2 Runtime Systems
- 4.1.2 I/O systems
- 4.1.3 External Environments
- 4.1.4 Systems Management

4.2 Development Environments

- 4.2.1 Programming Models
- 4.2.2 Frameworks
- 4.2.3 Compilers
- 4.2.4 Numerical Libraries
- 4.2.5 Debugging tools

4.3 Applications

- 4.3.1 Application Element: Algorithms
- 4.3.2 Application Support: Data Analysis and Visualization
- 4.3.3 Application Support: Scientific Data Management

4.4 Crosscutting Dimensions

- 4.4.1 Resilience
- 4.4.2 Power Management
- 4.4.3 Performance Optimization
- 4.4.4 Programmability

Roadmap Components

4.1 Systems Software

- 4.1.1 Operating systems
- 4.1.2 Runtime Systems
- 4.1.3 I/O systems
- 4.1.4 Systems Management
- 4.1.5 External Environments

4.2 Development Environments

- 4.2.1 Programming Models
- 4.2.2 Frameworks
- 4.2.3 Compilers
- 4.2.4 Numerical Libraries
- 4.2.5 Debugging Tools

4.3 Applications

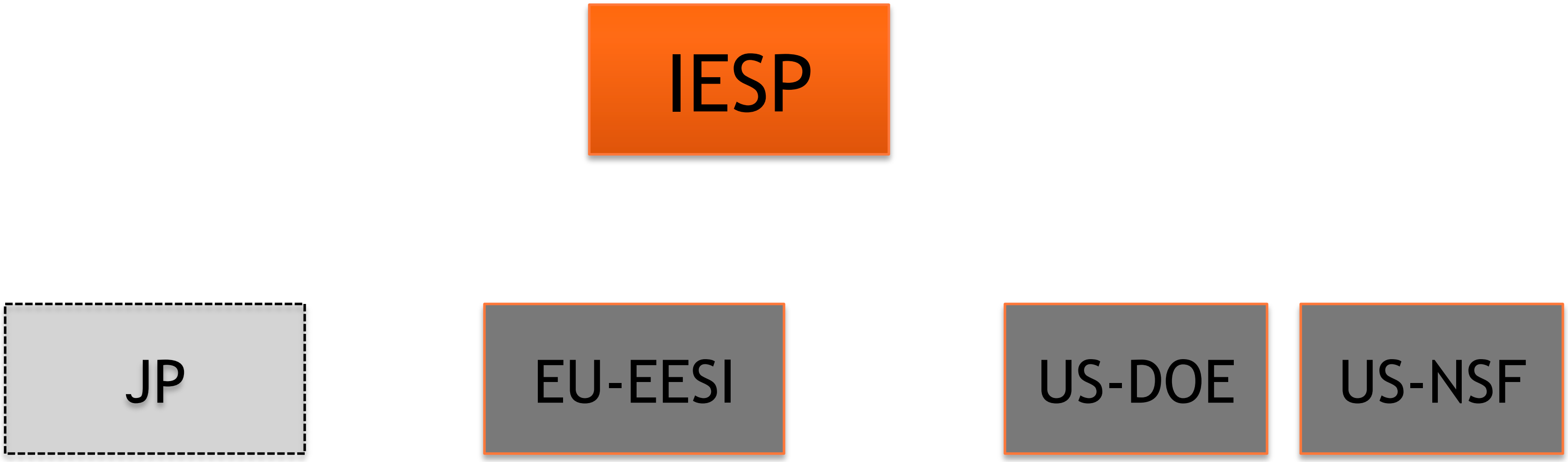
- 4.3.1 Application Element: Algorithms
- 4.3.2 Application Support: Data Analysis and Visualization
- 4.3.3 Application Support: Scientific Data Management

4.4 Crosscutting Dimensions

- 4.4.1 Resilience
- 4.4.2 Power Management
- 4.4.3 Performance Optimization
- 4.4.4 Programmability

see IJHPCA, Feb 2011, http://hpc.sagepub.com/content/25/1/3

Example Organizational Structure: Incubation Period (today):



- IESP provides coordination internationally, while regional groups have well managed R&D plans and milestones

INTERNATIONAL
EXASCALE
SOFTWARE PROJECT

ROADMAP

Jack Dongarra
Pete Beckman
Terry Moore
Patrick Aerts
Giovanni Aloisio
Jean-Claude Andre
David Barkai
Jean-Yves Berthou
Taisuke Boku
Bertrand Braunschweig
Franck Cappello
Barbara Chapman
Xuebin Chi

Alok Choudhary
Sudip Dosanjh
Thom Dunning
Sandro Fiore
Al Geist
Bill Gropp
Robert Harrison
Mark Hereld
Michael Heroux
Adolfy Hoisie
Koh Hotta
Yutaka Ishikawa
Fred Johnson

Sanjay Kale
Richard Kenway
David Keyes
Bill Kramer
Jesus Labarta
Alain Lichnewsky
Thomas Lippert
Bob Lucas
Barney Maccabe
Satoshi Matsuoka
Paul Messina
Peter Michielse
Bernd Mohr

Matthias Mueller
Wolfgang Nagel
Hiroshi Nakashima
Michael E. Papka
Dan Reed
Mitsuhisa Sato
Ed Seidel
John Shalf
David Skinner
Marc Snir
Thomas Sterling
Rick Stevens
Fred Streitz

Bob Sugar
Shinji Sumimoto
William Tang
John Taylor
Rajeev Thakur
Anne Trefethen
Mateo Valero
Aad van der Steen
Jeffrey Vetter
Peg Williams
Robert Wisniewski
Kathy Yelick

SPONSORS

Office of Science
NSF
ANR
cea
CERFACS
CRAY
eDF
EPSRC
FUJITSU
INRIA
GENCI
NVIDIA
RIKEN
東京大学

Published in the January 2011 issue of The International Journal of High Performance Computing Applications

> "We can only see a short distance ahead, but we can see plenty there that needs to be done."
>
> - *Alan Turing (1912 —1954)*

- www.exascale.org

Conclusions

- For the last decade or more, the research investment strategy has been overwhelmingly biased in favor of hardware.
- This strategy needs to be rebalanced - barriers to progress are increasingly on the software side.
- Moreover, the return on investment is more favorable to software.
  - Hardware has a half-life measured in years, while software has a half-life measured in decades.
- High Performance Ecosystem out of balance
  - Hardware, OS, Compilers, Software, Algorithms, Applications
    - No Moore's Law for software, algorithms and applications

PLASMA and Magma

Institutions

University of Tennessee

Knoxville

University of Colorado

Denver

University of California

Berkeley

Sponsors

NSF

U.S. DEPARTMENT OF ENERGY

intel

NVIDIA

The MathWorks

AMD

Microsoft

PLASMA and MAGMA

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

names listed alphabetically

Last …

- Thank a number of people who have helped with this work
  - Emmanuel Agullo, George Bosilca, Aurelien Bouteiller, Anthony Danalis, Jim Demmel, Tingxing "Tim" Dong, Mathieu Faverge, Azzam Haidar, Thomas Herault, Mitch Horton, Jakub Kurzak, Julien Langou, Julie Langou, Pierre Lemarinier, Piotr Luszczek, Hatem Ltaief, Fengguang Song, Stanimire Tomov, Asim YarKhan