ARCHITECTURE-AWARE ALGORITHMS FOR SCALABLE PERFORMANCE AND RESILIENCE ON HETEROGENEOUS ARCHITECTURES

THE EXTREME-SCALE ALGORITHMS & SOFTWARE INSTITUTE (EASI)

Jack Dongarra

University of Tennessee and
Oak Ridge National Laboratory

ASCAC Meeting

August 2010

Research supported by DOE ASCR

Four DOE Math/CS Institutes

- CACHE - Communication Avoidance and Communication Hiding at the Extreme Scale
  - Goal: simplify algorithm specification, orchestration of data movements, mapping to complex computer architectures, portable performance
  - Lead by Erich Strohmaier – LBNL, ANL, UCB, & Colorado SU
- Nonlinear Algorithms to Circumvent the Memory Bandwidth Limitations of Implicit PDE Simulations
  - Goal: Efficient, scalable implicit solution of nonlinear PDEs
  - Lead by Barry Smith – ANL, BNL, ORNL, U of Chicago & U of Kansas
- I/O Coordination to Improve HEC System Performance: A Marriage of Analytical Modeling, Control Theory
  - Goal: Extend the scalability of checkpoint/restart and reduce the stress on the I/O system and resultant failures
  - Lead by Pat Teller – U Texas El Paso
- EASI

Extreme-scale Algorithms & Software Institute - EASI

- Architecture-aware Algorithms for Scalable Performance and Resilience on Heterogeneous Architectures
- EASI Team
  - Lead PI: Al Geist (ORNL)
  - Ron Brightwell (SNL)
  - Jim Demmel (UC Berkeley)
  - Jack Dongarra (UTK/ORNL)
  - George Fann (ORNL)
  - Bill Gropp (UIUC)
  - Michael Heroux (SNL)

EASI Goals:

- Study and characterize the application-architecture performance gaps that we can address in the near-term and identify architecture features that future systems may want to incorporate.
- Develop multi-precision and architecture-aware implementations of Krylov, Poisson, Helmholtz solvers, and dense factorizations for heterogeneous multi-core systems.
- Explore new methods of algorithm resilience, and develop new algorithms with these capabilities.
- Develop runtime support for adaptable algorithms that are dealing with resilience, scalability, and performance.
- Demonstrate architecture-aware algorithms in full DOE applications on large-scale DOE architectures
- Distribute the new algorithms and runtime support through widely used software packages.
- Establish a strong outreach program to disseminate results, interact with colleagues and train students and junior members of our community.

EASI uses co-design to provide both near and long-term Impact:

Integrated team of math, CS, and application experts working together to create new:

Architecture-aware algorithms and associated runtime to enable many science applications to better exploit the architectural features of DOE's petascale systems.

Applications team members immediately incorporate new algorithms providing **Near-term high impact on science**

Numerical libraries used to disseminate the new algorithms to the wider community providing **broader and longer-term impact.**

EASI Project Overview

Addressing Heterogeneity and Resilience

Applications

HOMME
MADNESS
Charon

Algorithms

Krylov
Poisson
Helmholtz
Dense NLA
BLAS

Runtime

MPI
Shared-memory
Processor affinity
Memory affinity

Architecture **Heterogeneous, multi-core, extreme-scale**

Research Areas in Institute

Heterogeneous programming API
Robust multi-precision algorithms
Hybrid programming
Resilient algorithms
Communication optimal algorithms
Auto-tuned BLAS (API)
New parallelization methods

Task placement and scheduling
Memory management
Architecture-aware MPI

Community Outreach

Deliver codes to community through:

ScaLAPACK
Trilinos
Open MPI
MPICH2
MADNESS
HOMME

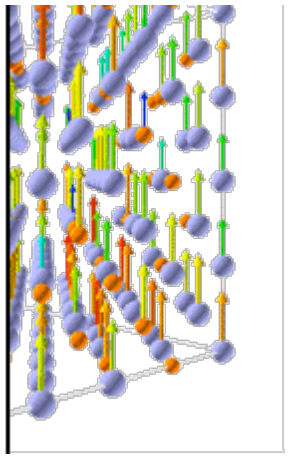
Workshops
Training
Publications

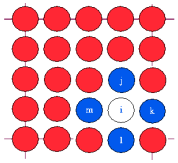
EASI Budget

- Duration: 3 years.
  - Started in Fall of 2009 for Labs
  - Spring/Summer 2010 for Universities (no idea why the lag)
- Total funding over 3 years $7.425M
  - ORNL $1M/year
  - SNL $1M/year
  - UTK $150K/year
  - UCB $150K/year
  - UIUC $150K/year

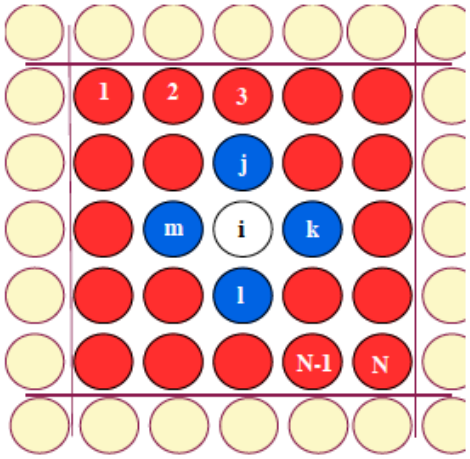
LSMS

- The Locally-Self-Consistent Multiple-Scattering (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

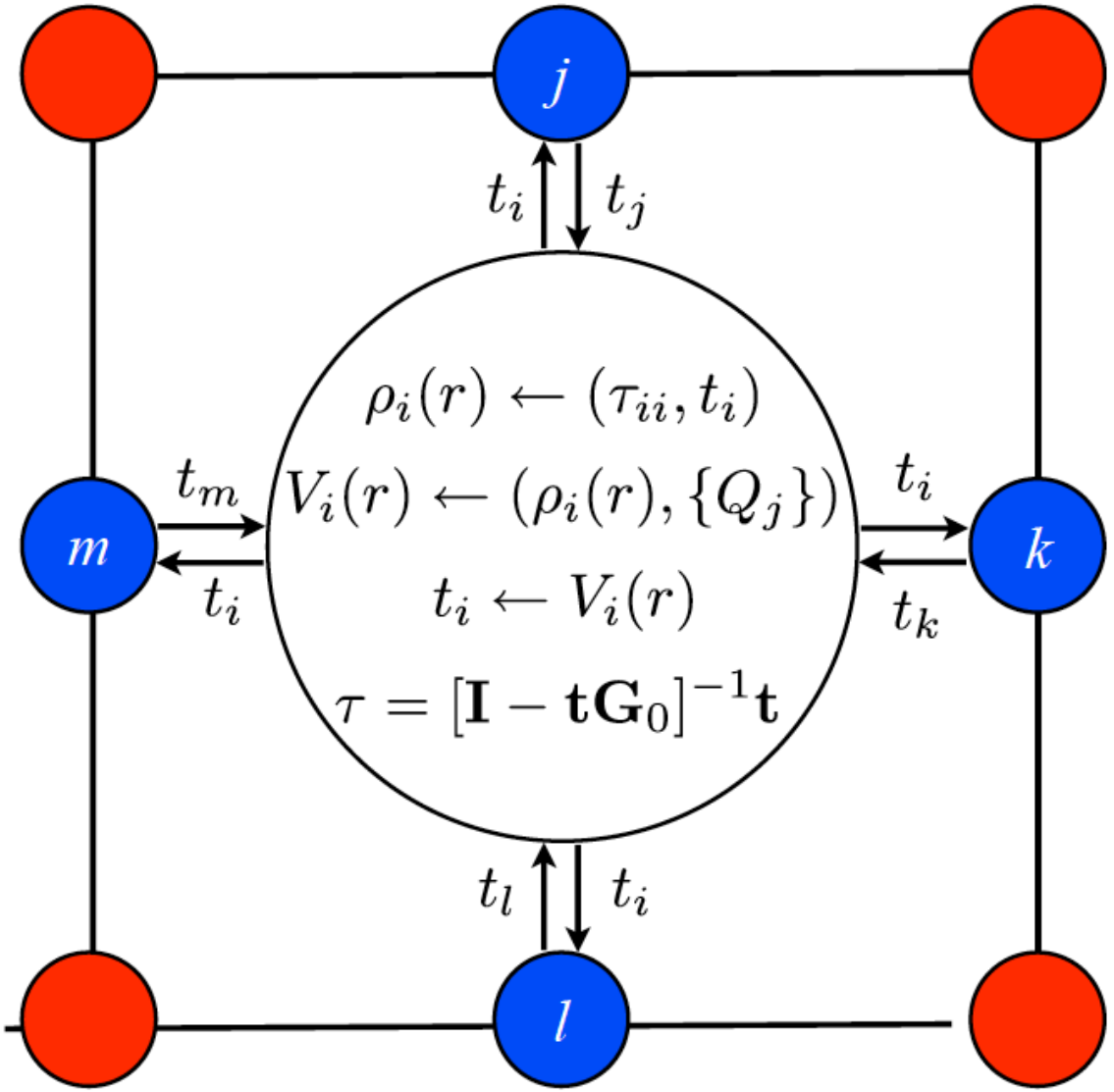




A parallel implementation and scaling of the LSMS method: perfectly scalable at high performance



- 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



Complex Multiplication

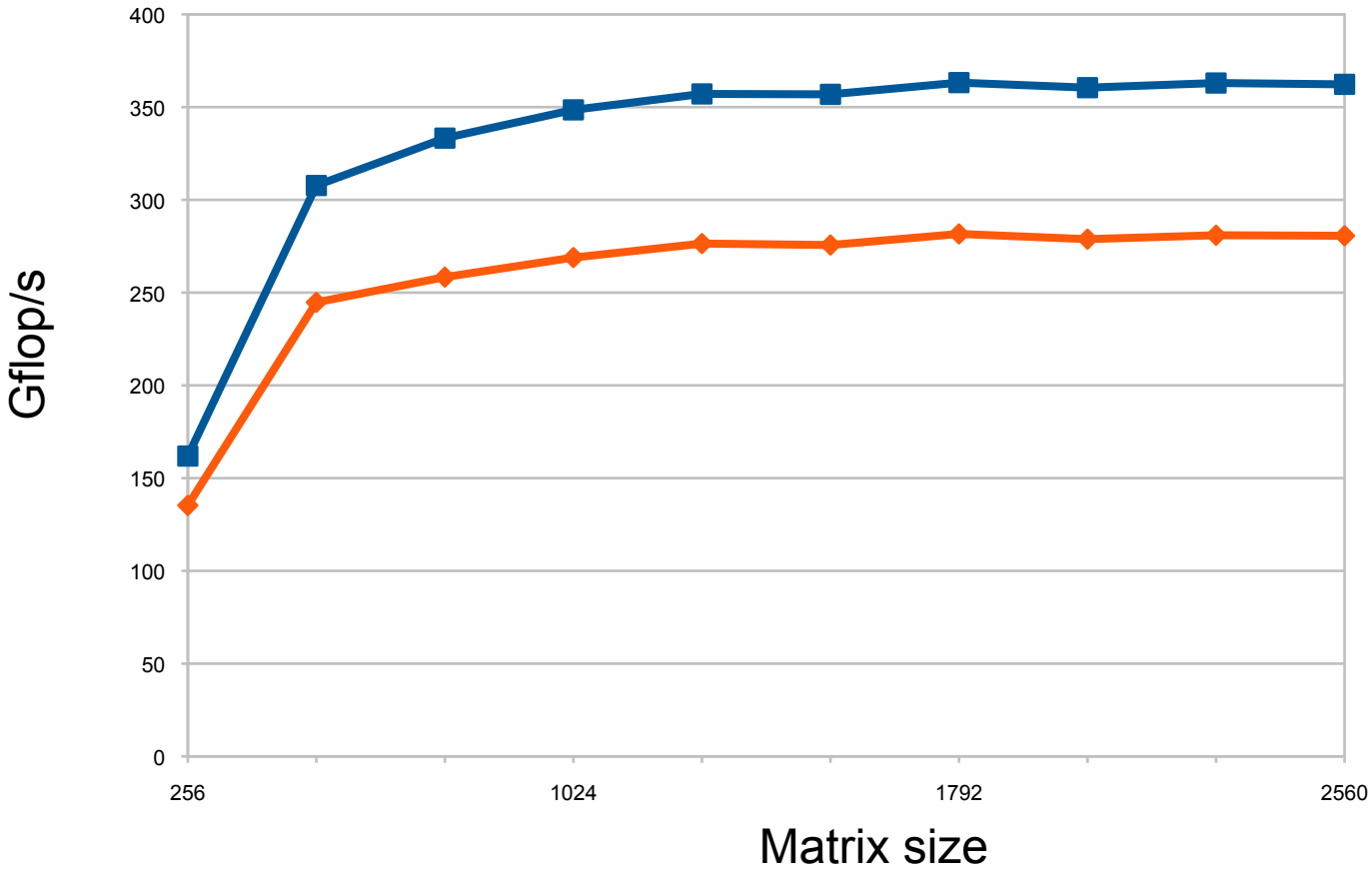
- The product $(a + bi) \cdot (c + di)$ normally requires 4 multiplications and 2 additions
  - Real part $= a \cdot c - b \cdot d$
  - Imaginary part $= a \cdot d + b \cdot c$
- But it can be calculated in the following way.
  - $k1 = c \cdot (a + b)$
  - $k2 = a \cdot (d - c)$
  - $k3 = b \cdot (c + d)$
  - 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. – N. Higham

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)

Develop robust multi-precision algorithms

- 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)	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		

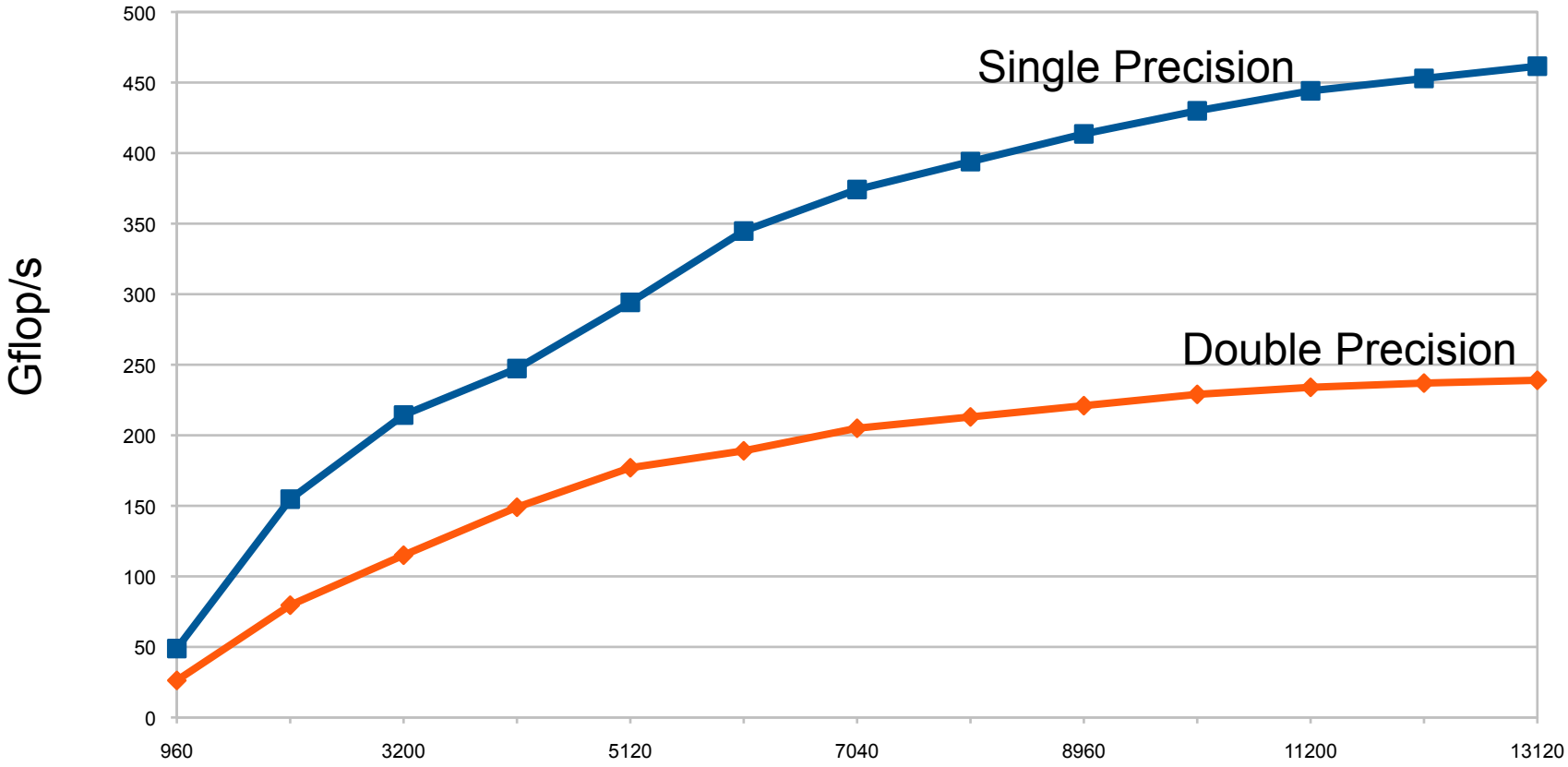
- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

- 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; $O(10^8)$

Results for Mixed Precision Iterative Refinement for Dense $Ax = b$

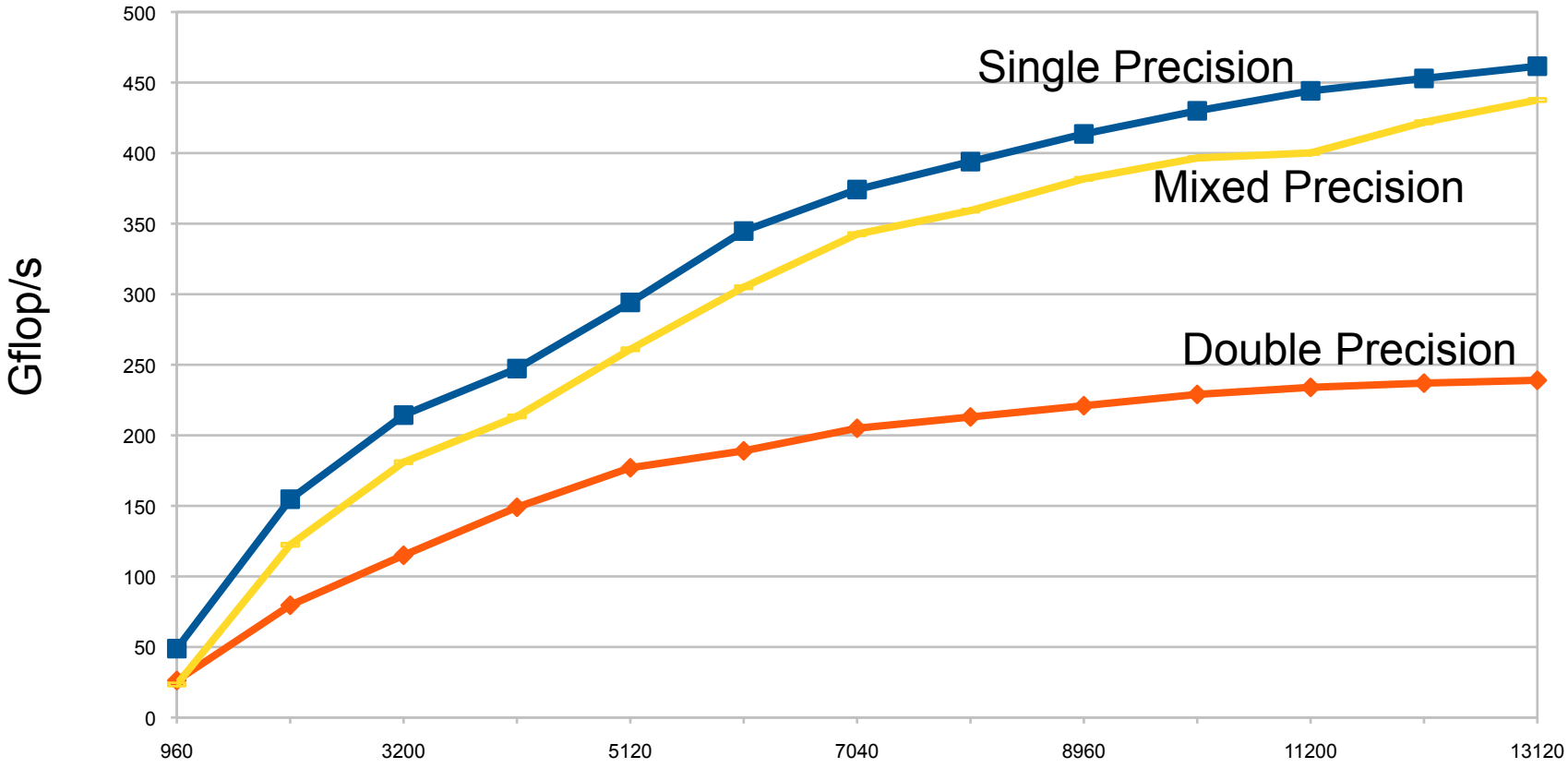
- Single precision is faster than DP because:
  - **Higher parallelism within floating point units**
    - 4 ops/cycle (usually) instead of 2 ops/cycle
  - **Reduced data motion**
    - 32 bit data instead of 64 bit data
  - **Higher locality in cache**
    - More data items in cache

Ax = b



Tesla C2050, 448 CUDA cores (14 multiprocessors x 32) @ 1.15 GHz.,
3 GB memory, connected through PCIe to a quad-core Intel @2.5 GHz.

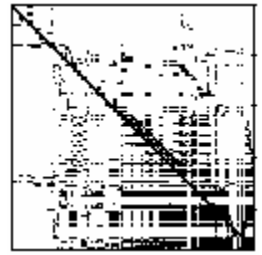
Ax = b

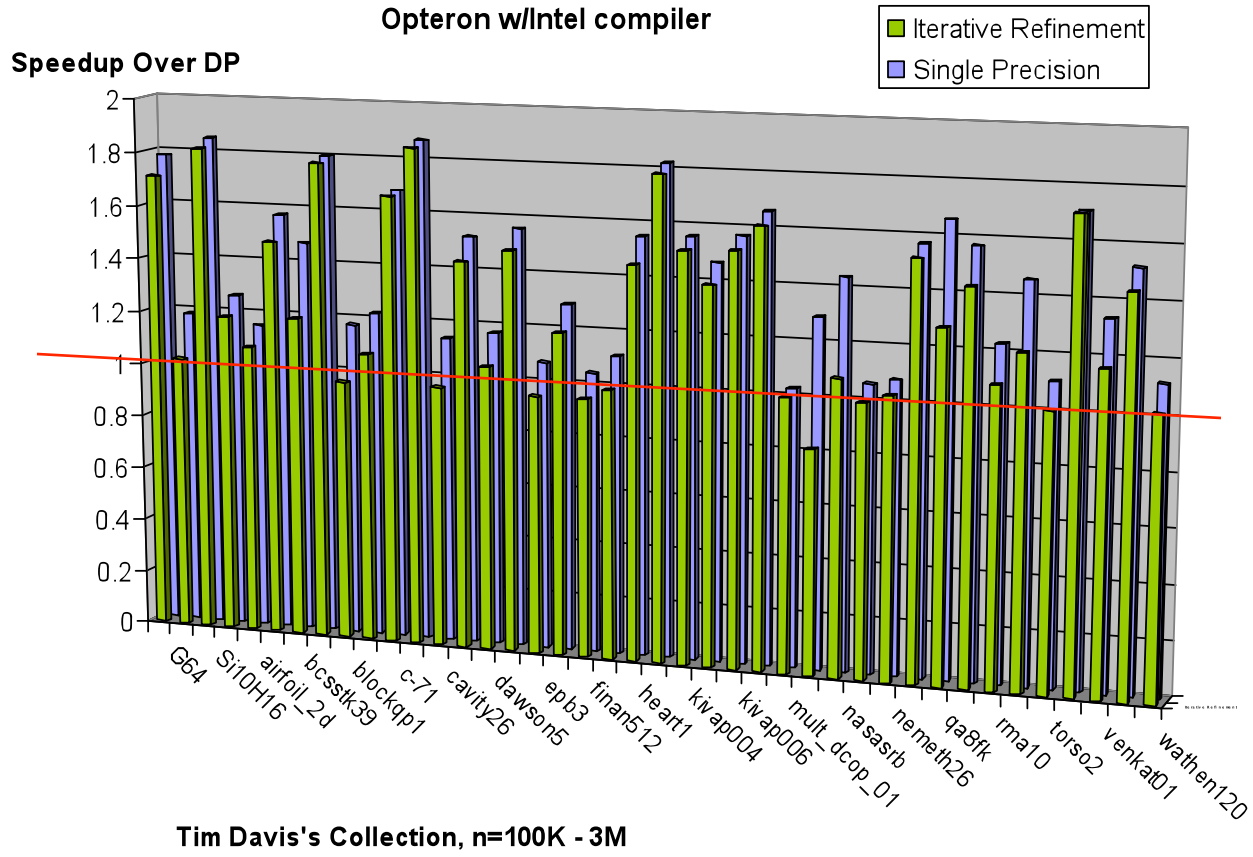


Tesla C2050, 448 CUDA cores (14 multiprocessors x 32) @ 1.15 GHz.,
3 GB memory, connected through PCIe to a quad-core Intel @2.5 GHz.

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

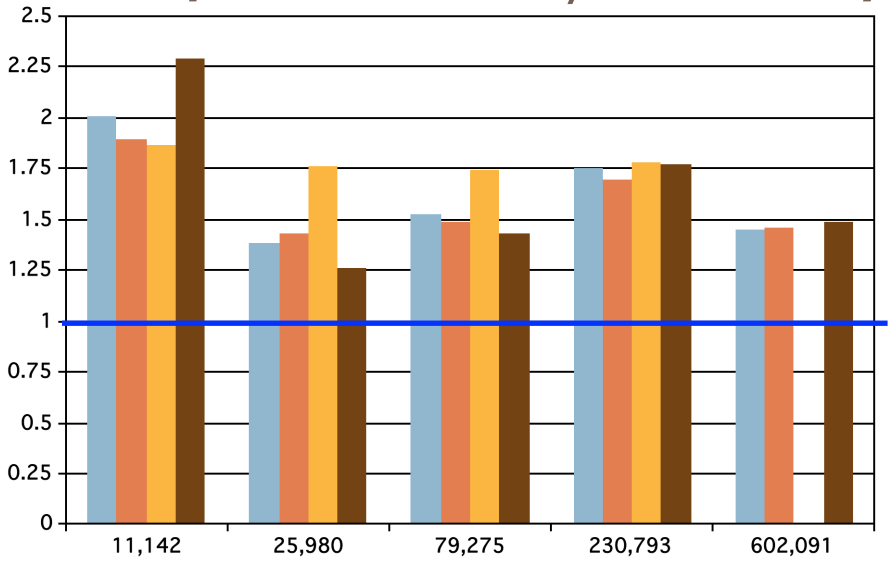
Inner iteration:
In 32 bit floating point

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  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

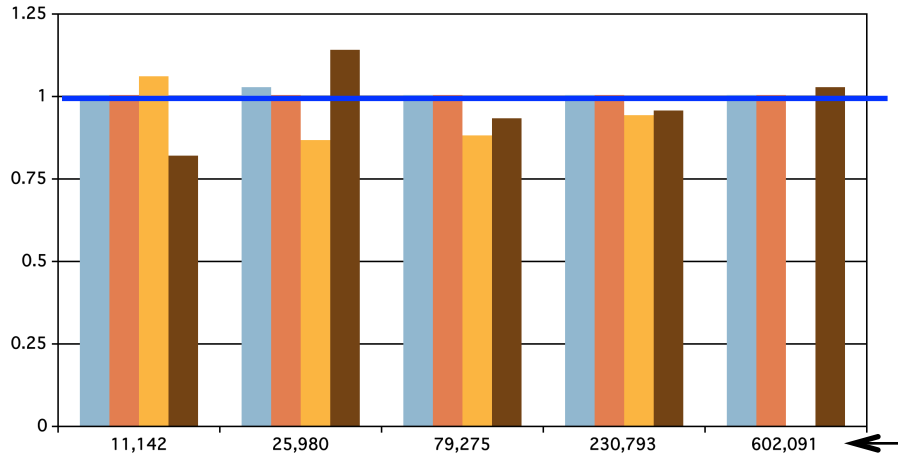
Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
for $i = 1, 2, \ldots$
  solve $Mz^{(i-1)} = r^{(i-1)}$
  $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
  if $i = 1$
    $p^{(1)} = z^{(0)}$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif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

Intriguing Potential

- Exploit lower precision as much as possible
  - Payoff in performance
    - Faster floating point
    - Less data to move
- Automatically switch between SP and DP to match the desired accuracy
  - Compute solution in SP and then a correction to the solution in DP
- Potential for GPU, FPGA, special purpose processors
  - Use as little you can get away with and improve the accuracy
- Applies to sparse direct and iterative linear systems and Eigenvalue, optimization problems, where Newton's method is used.

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

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

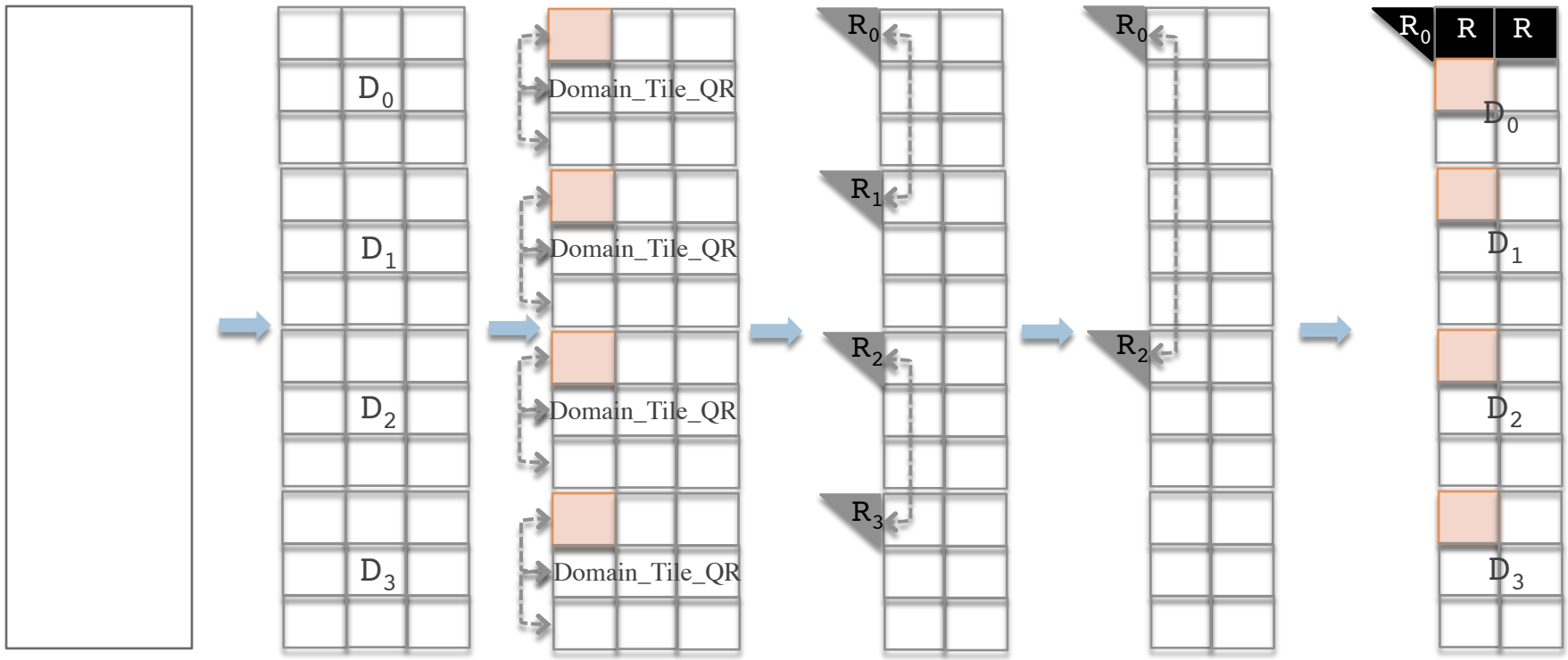
Correction = - A\(b – Ax)

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)
    - Mostly not in LAPACK or ScaLAPACK (yet)
- Iterative methods – Krylov subspace methods for Ax=b, Ax=λx
  - Communication lower bounds, and algorithms that attain them (depending on sparsity structure)
    - Not in any libraries (yet)
- For QR Factorization they can show:

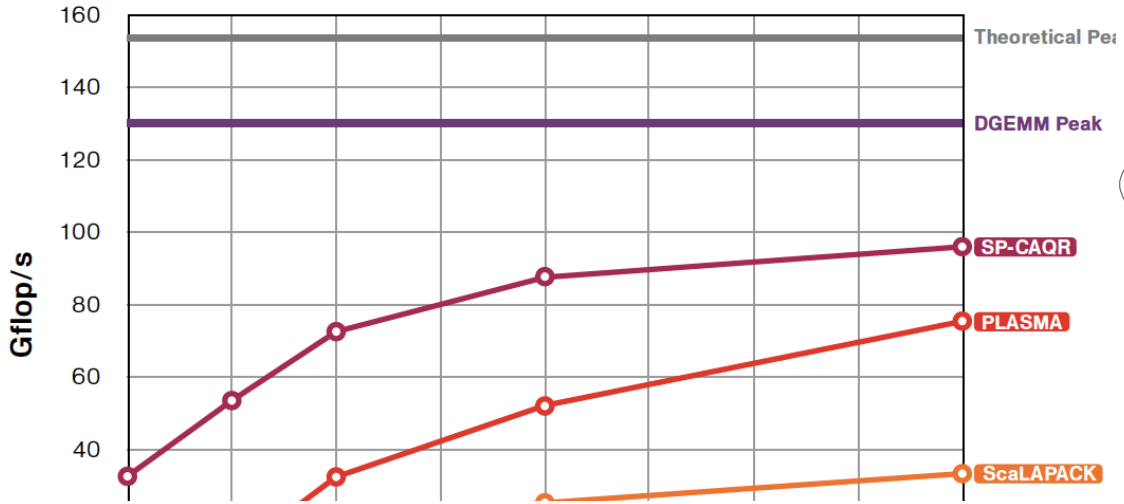
	Lower bound
# flops	$\Theta(mn^2)$
# words	$\Theta(\frac{mn^2}{\sqrt{W}})$
# messages	$\Theta(\frac{mn^2}{W^{3/2}})$

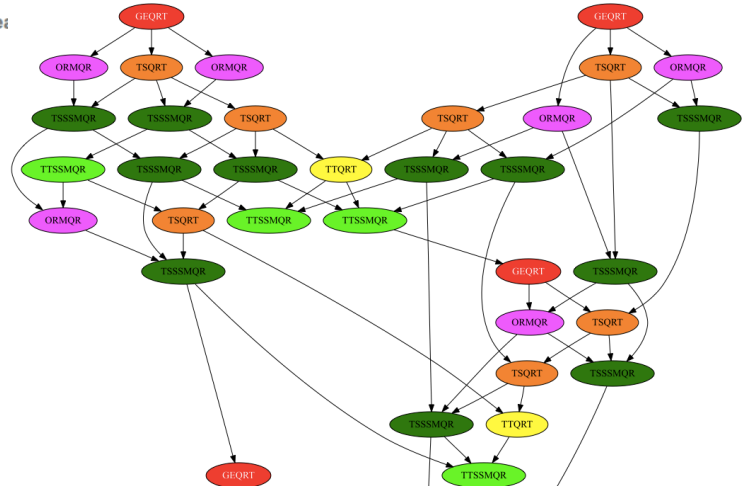
Communication Avoiding QR Example

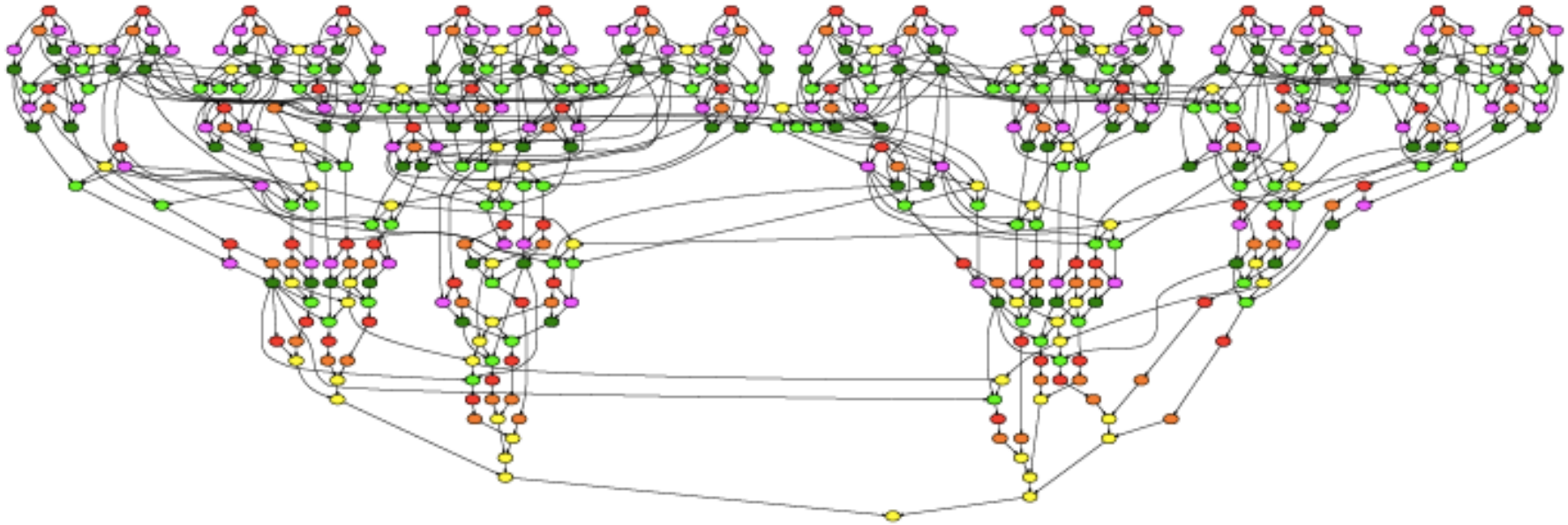


A. Pothen and P. Raghavan. Distributed orthogonal factorization. In *The 3rd Conference on Hypercube Concurrent Computers and Applications, volume II, Applications,* pages 1610–1620, Pasadena, CA, Jan. 1988. ACM. Penn. State.

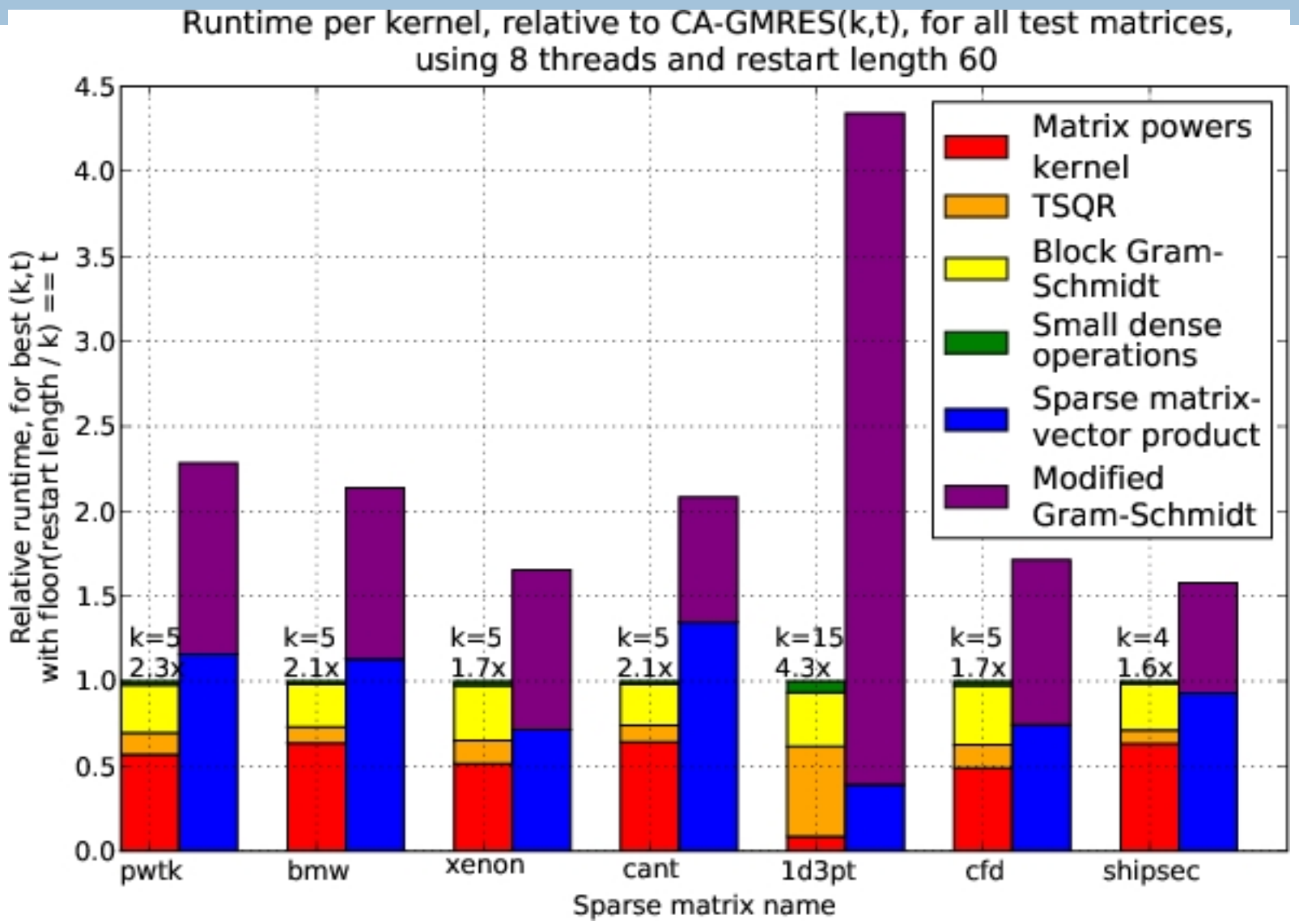
Communication Reducing QR Factorization





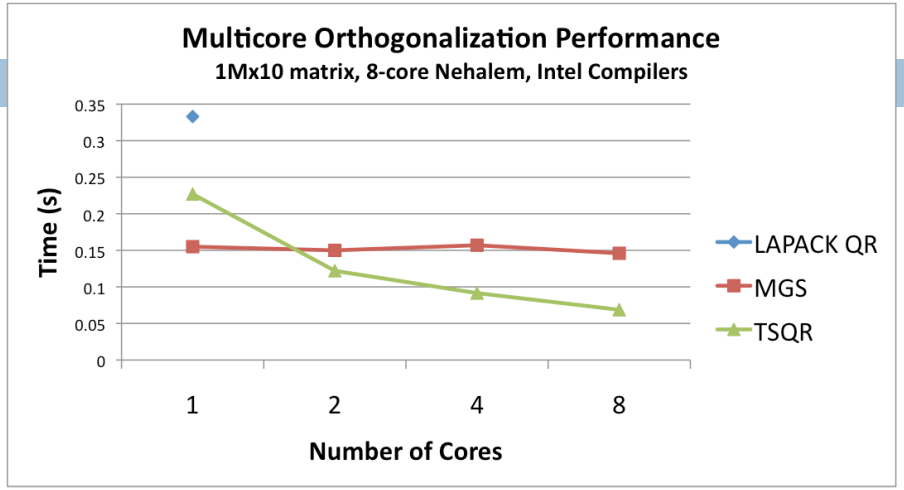


GMRES speedups on 8-core Clovertown



Communication-avoiding iterative methods

- Iterative Solvers:
  - Dominant cost of many apps (up to 80+% of runtime).
- Exascale challenges for iterative solvers:
  - Collectives, synchronization.
  - Memory latency/BW.
  - **Not viable on exascale systems in present forms.**
- Communication-avoiding (*s*-step) iterative solvers:
  - Idea: Perform *s* steps in bulk (*s*=5 or more):
    - *s* times fewer synchronizations.
    - *s* times fewer data transfers: Better latency/BW.
  - Problem: Numerical accuracy of orthogonalization.
- TSQR Implementation:
  - 2-level parallelism (Inter and intra node).
  - Memory hierarchy optimizations.
  - Flexible node-level scheduling via Intel Threading Building Blocks.
  - Generic scalar data type: supports mixed and extended precision.



LAPACK – Serial, MGS –Threaded modified Gram-Schmidt

TSQR capability:

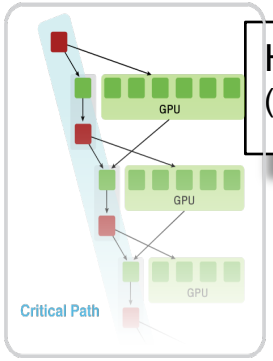
- Critical for exascale solvers.
- Part of the Trilinos scalable multicore capabilities.
- Helps all iterative solvers in Trilinos (available to external libraries, too).
- Staffing: Mark Hoemmen (lead, post-doc, UC-Berkeley), M. Heroux
- **Part of Trilinos 10.6 release, Sep 2010.**

Trilinos

Developing heterogeneous, multi-core-aware algorithms and software

- Dense solvers for multicore/GPUs – MAGMA Project
  - **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, to allow scientists to effortlessly port any LAPACK-relying software components to take advantage of 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)

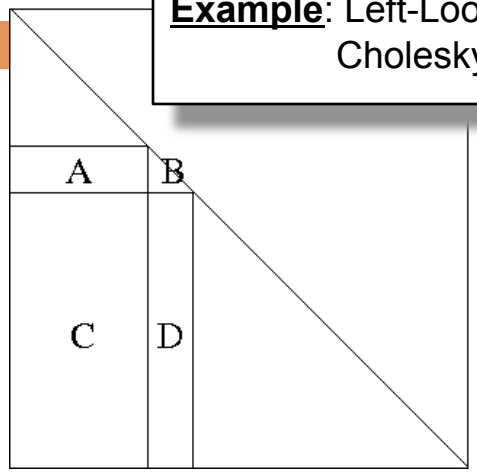
- MAGMA uses **HYBRIDIZATION** methodology based on
  - Representing linear algebra algorithms as collections of **TASKS** and **DATA DEPENDENCIES** among them
  - Properly **SCHEDULING** tasks' execution over multicore and GPU hardware components

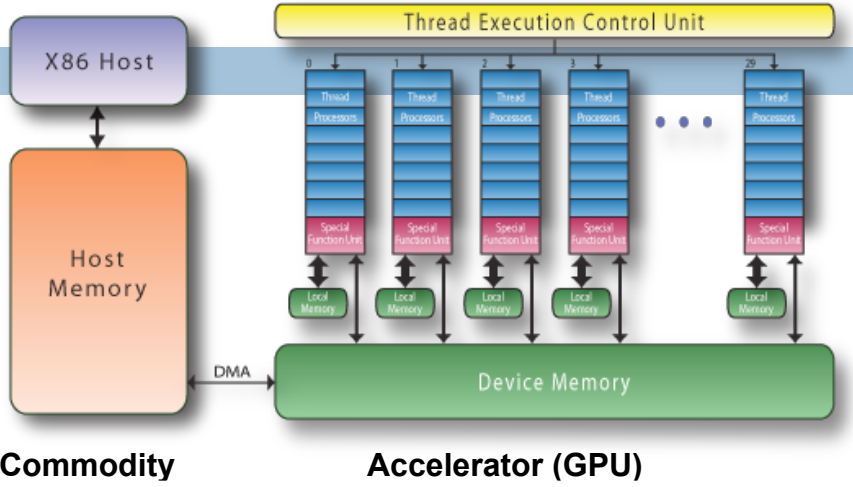


Hybrid CPU+GPU algorithms
(small tasks for multicores and large tasks for GPUs)

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

Example: Left-Looking Hybrid Cholesky factorization



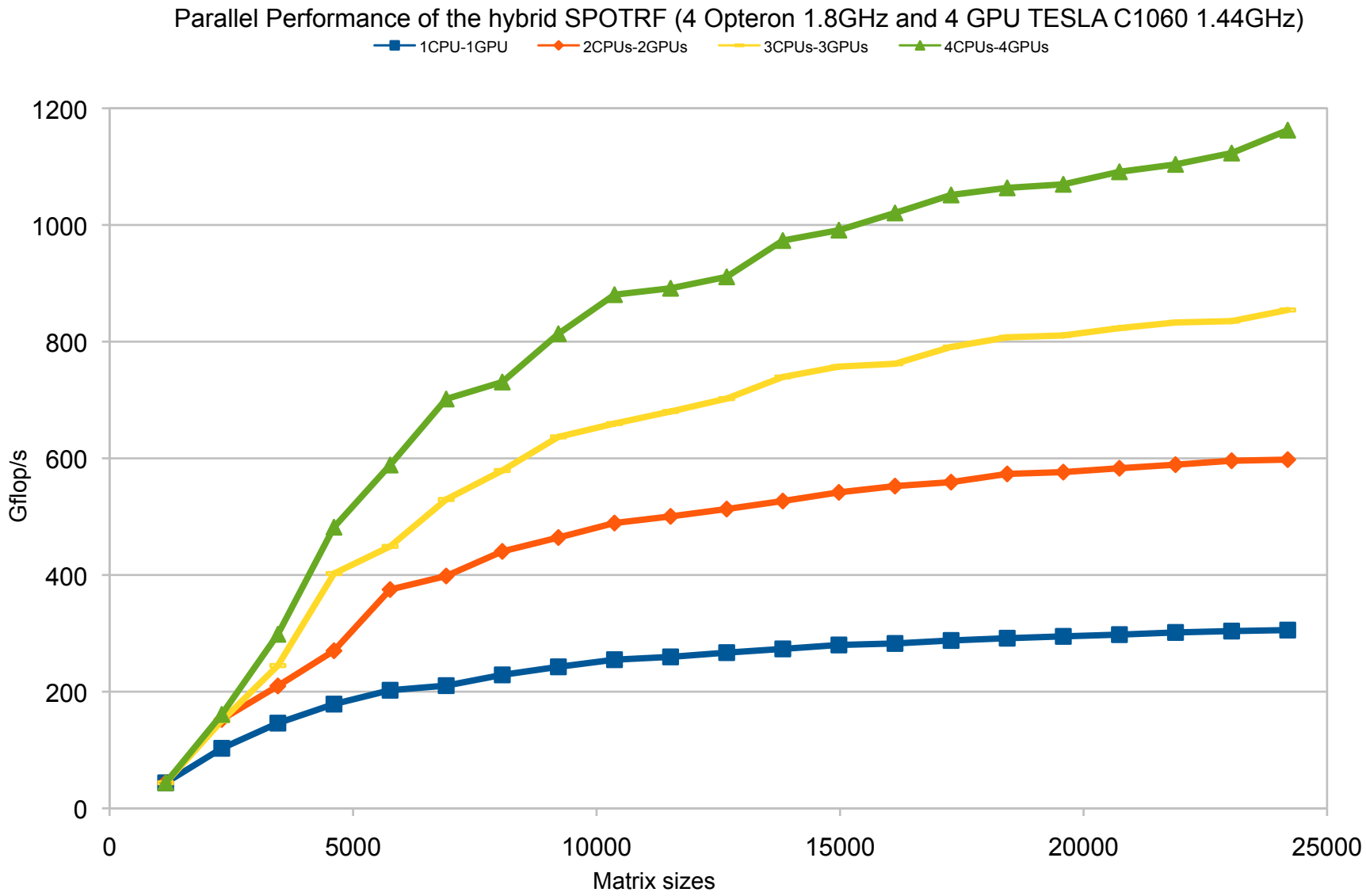


MATLAB code	LAPACK code	Hybrid code
(1) B = B – A*A'	ssyrk_(“L”, “N”, &nb, &j, &mone, hA(j,0), ...)	cublasSsyrk('L', 'N', nb, j. mone, dA(j,0), ...)
		cublasGetMatrix(nb, nb, 4, dA(j, j), *lda, hwork, nb)
(2) B = chol(B, 'lower')	spotrf_(“L”, &nb, hA(j, j), lda, info)	cublasSgemm('N', 'T', j, ...)
(3) 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

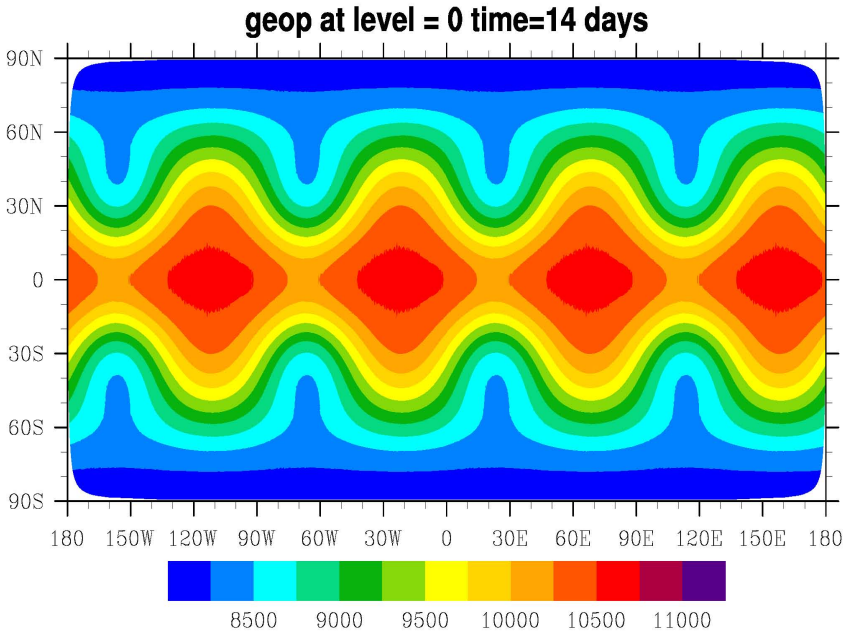
SP Cholesky on Multicore + Multi GPUs



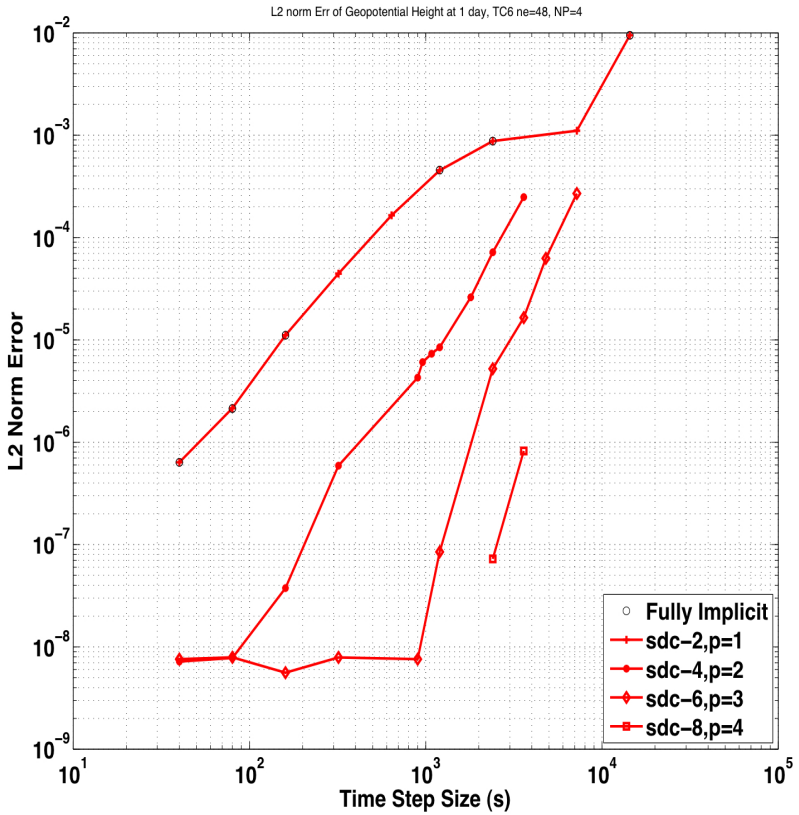
High Order Pseudo-Parallel Time Stepping With Application to Climate Dynamics

G. Fann, J. Jia, J. Hill, K. Evans ORNL
M. Taylor SNL

Toward Accurate Long Term Predictions



Simulation of the shallow-water equation using the HOMME for test case 6. The geopotential height is shown above for 14 simulated days.



Improved accuracy using high-order time stepping is illustrated as the simulation evolves over time. The error of the implicit Jacobian-Free-Newton-Krylov fully implicit method and the hybrid Krylov deferred correction implicit methods from orders 2 to order 8 are shown. These are more accurate than existing time-stepping methods in HOMME. This simulation was performed using more than 4000 cores on ORNL's Cray XT-5

EASI Runtime Research Overview

Develop the supporting Architecture Aware Runtime required by the above algorithms.

Look at issues such as process placement, memory affinity, thread scheduling and data movement typically out of the control of the application but which have dramatic effects on efficiency and performance on multi-core sockets and large-scale systems.

Extending MPI for Hierarchical Architectures

- Today the only "defined" communicator is MPI_COMM_WORLD which assumes a flat architecture (i.e. architecture unaware)

- **We have extended the MPI interface and runtime to enable existing MPI algorithms to discover and take advantage of the hardware hierarchy and multi-core shared memory.**

  - **Defined new architecture aware MPI communicators**
    MPI_COMM_NODE
    MPI_COMM_SOCKET
    MPI_COMM_NETWORK
    MPI_COMM_CACHE

Broader Impact and Standardization

Goal: Distribute the new algorithms and runtime support through widely used software packages

Open MPI and MPICH are the two most widely used MPI libraries.

In the past month we have gotten these MPI extensions officially accepted as a branch of the Open MPI source tree.

Standardization efforts: Ron Brightwell is an area lead in the **MPI-3** forum. He has begun formal discussions with the forum about getting these features into the standard by showing their advantages to future architecture aware algorithms

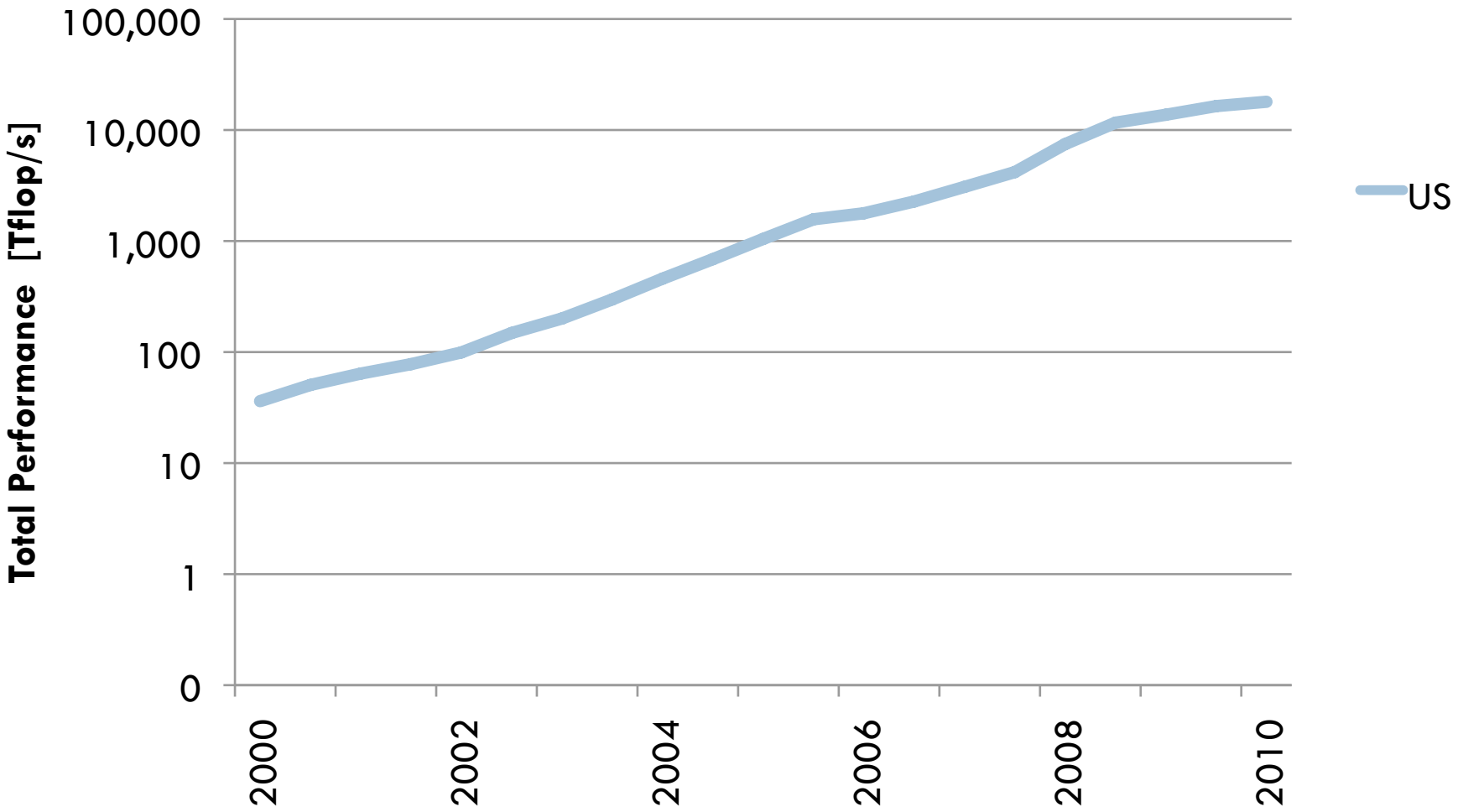
Current Status - UIUC

- PI: William Gropp
- Startup delayed due to delays in funding
  - Funding arrived in July, 2010
- Graduate students have joined the project
  - Elena Caraba
  - Vivek Kale
- Two major areas of focus:
  - Portable Hybrid MPI
  - Robust and Reliable Iterative Methods

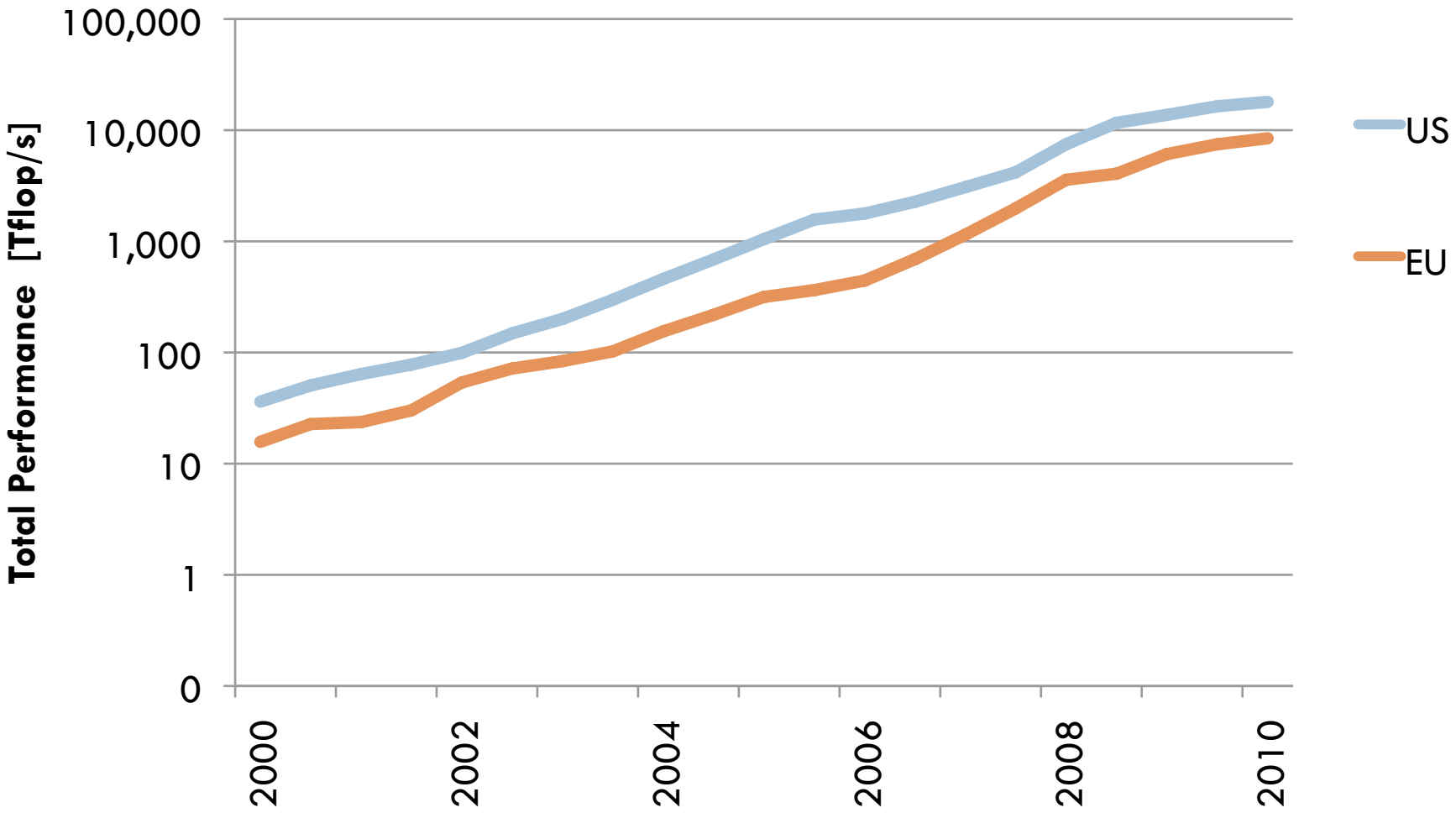
http://www.exascale.org

Jack Dongarra and Pete Beckman

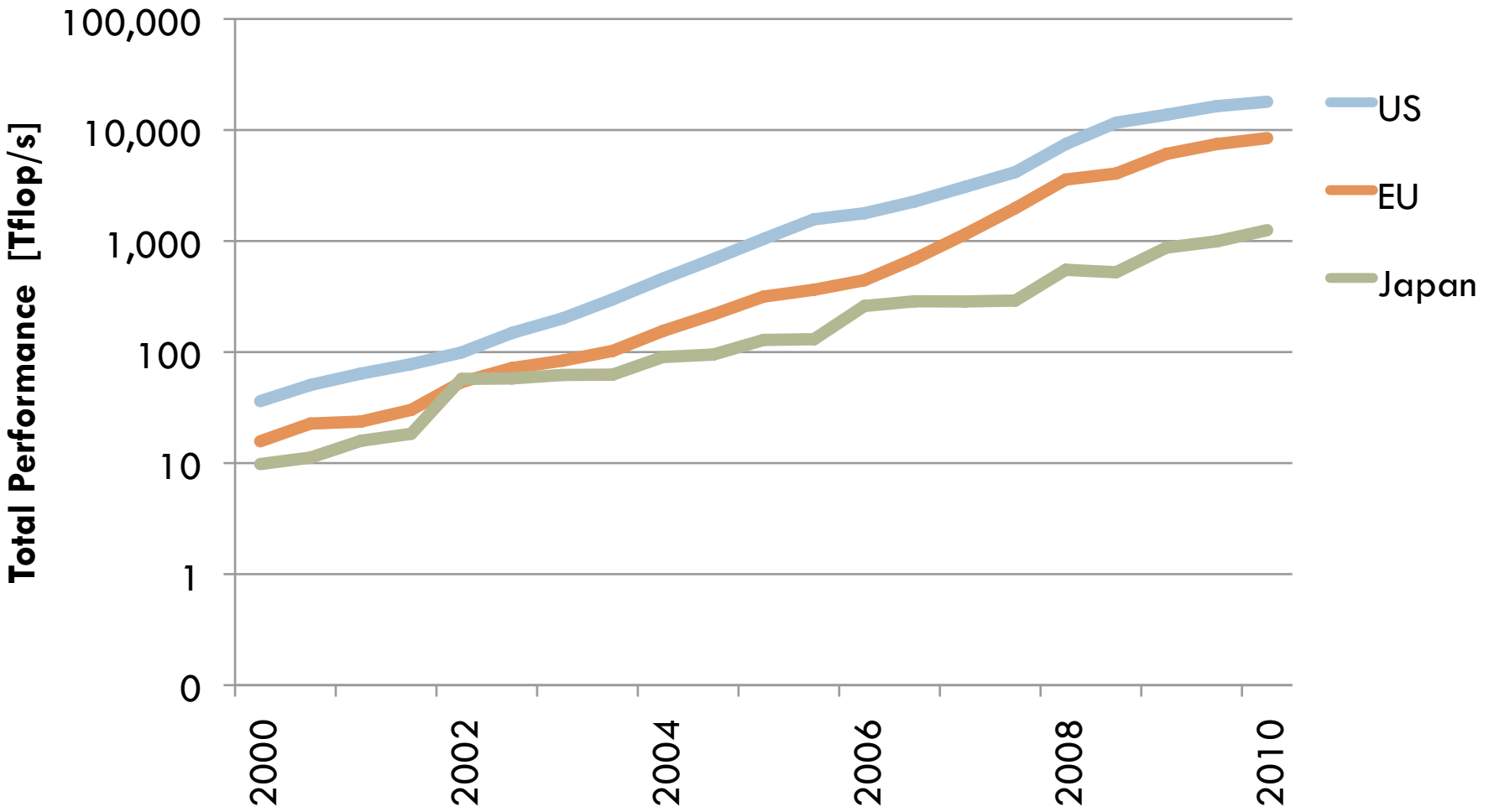
Performance of Countries



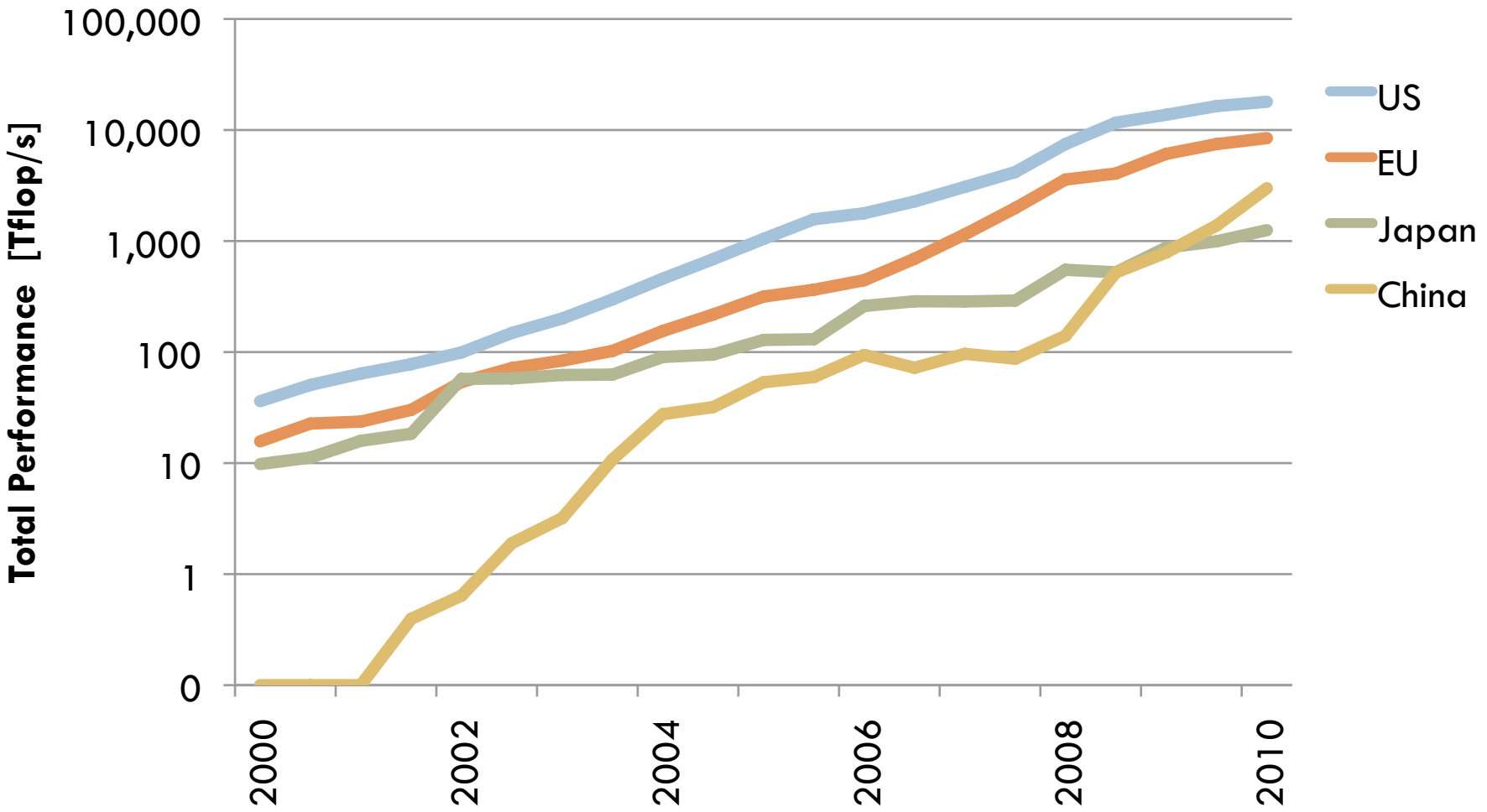
Performance of Countries



Performance of Countries



Performance of Countries

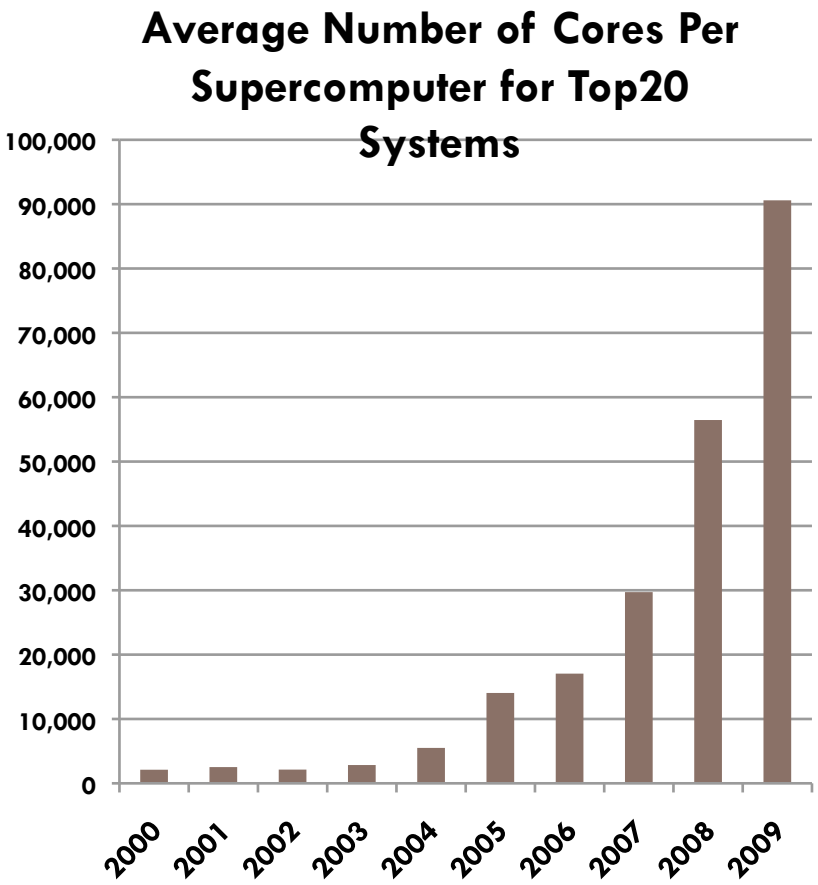


Potential System Architecture with a cap of $200M and 20MW

Systems	2010	2018	Difference Today & 2018
System peak	**2 Pflop/s**	**1 Eflop/s**	**O(1000)**
Power	**6 MW**	**~20 MW**	
System memory	0.3 PB	32 - 64 PB [**.03 Bytes/Flop**]	O(100)
Node performance	125 GF	1,2 or 15TF	O(10) – O(100)
Node memory BW	25 GB/s	2 - 4TB/s [**.002 Bytes/Flop**]	O(100)
Node concurrency	12	O(1k) or 10k	O(100) – O(1000)
Total Node Interconnect BW	3.5 GB/s	200-400GB/s (1:4 or 1:8 from memory BW)	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) to O(100) for latency hiding]	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)

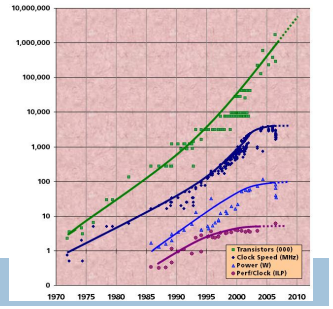
Factors that Necessitate Redesign

- **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



Software infrastructure does not exist today

A Call to Action



- Hardware has changed dramatically while software ecosystem has remained stagnant
- Previous approaches have not looked at co-design of multiple levels in the system software stack (OS, runtime, compiler, libraries, application frameworks)
- No global evaluation of key missing components
- Need to exploit new hardware trends (e.g., manycore, heterogeneity) that cannot be handled by existing software stack, memory per socket trends
- Emerging software technologies exist, but have not been fully integrated with system software, e.g., UPC, Cilk, CUDA, HPCS
- Community codes unprepared for sea change in architectures

IESP Goal

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 developing the next generation open source software for scientific high-performance computing

International Community Effort

- We believe this needs to be an international collaboration for various reasons including:
  - The scale of investment
  - The need for international input on requirements
  - US, Europeans, Asians, and others are working on their own software that should be part of a larger vision for HPC.
  - No global evaluation of key missing components
  - Hardware features are uncoordinated with software development

IESP Executive Committee

- Jack Dongarra, UTK & ORNL
- Pete Beckman, ANL
- Patrick Aerts, NWO Netherlands
- Franck Cappello, INRIA, France
- Thom Dunning, NCSA
- Thomas Lippert, Juelich, Germany
- Satoshi Matsuoka, TiTech, Japan
- Paul Messina, ANL
- Anne Trefethen, Oxford, UK
- Mateo Valero, BSC, Spain

Roadmap Purpose

- The IESP software roadmap is a planning instrument designed to enable the international HPC community to improve, coordinate and leverage their collective investments and development efforts.

- After we determine what needs to be accomplished, our task will be to construct the organizational structures suitable to accomplish the work

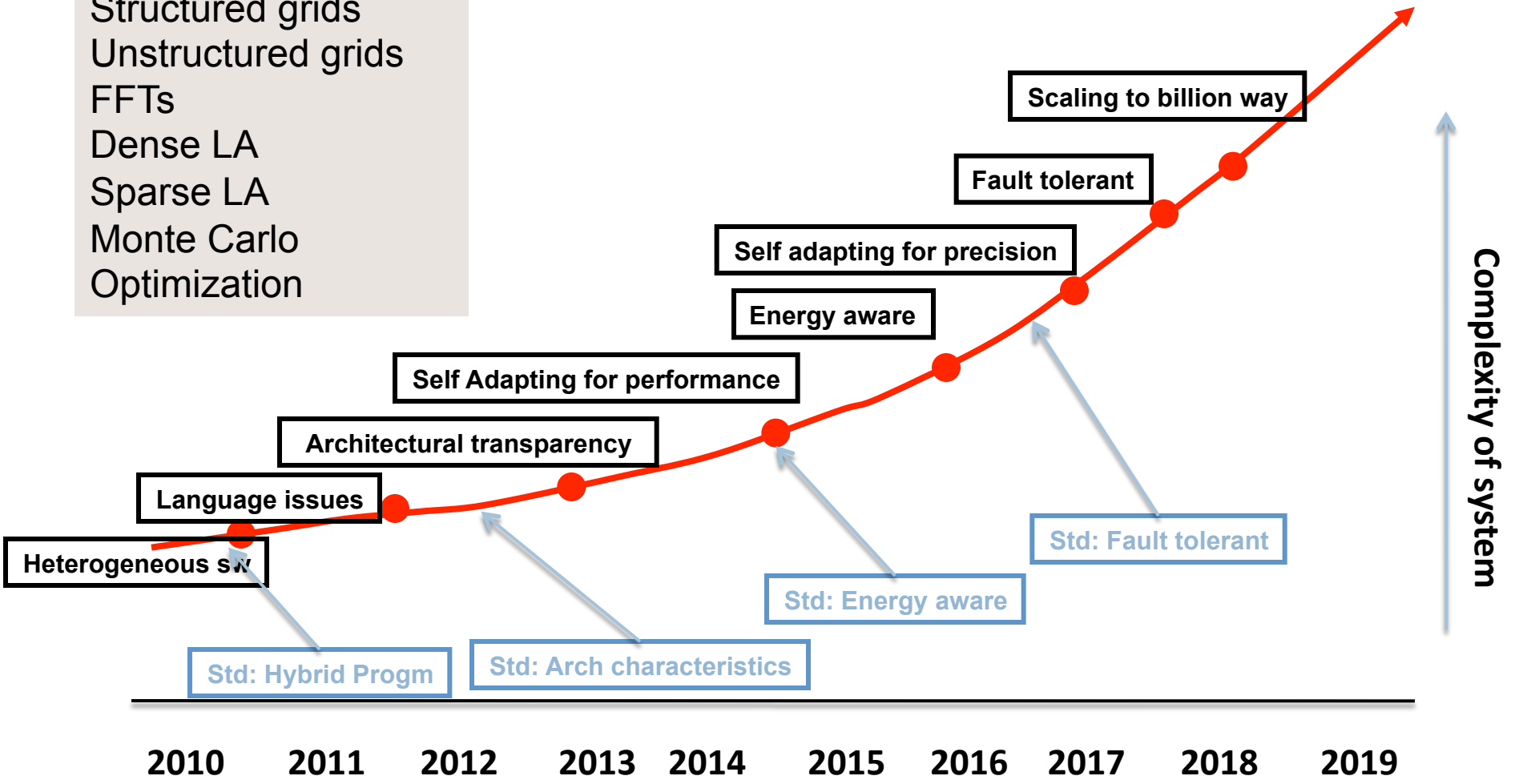
Roadmap Components

4.2.4 Numerical Libraries

- Technology drivers
  - Hybrid architectures
  - Programming models/languages
  - Precision
  - Fault detection
  - Energy budget
  - Memory hierarchy
  - Standards
- Alternative R&D strategies
  - Message passing
  - Global address space
  - Message-driven work-queue
- Recommended research agenda
  - Hybrid and hierarchical based software (eg linear algebra split across multi-core / accelerator)
  - Autotuning
  - Fault oblivious sw, Error tolerant sw
  - Mixed arithmetic
  - Architectural aware libraries
  - Energy efficient implementation
  - Algorithms that minimize communications
- Crosscutting considerations
  - Performance
  - Fault tolerance
  - Power management
  - Arch characteristics

4.2.4 Numerical Libraries

Numerical Libraries
Structured grids
Unstructured grids
FFTs
Dense LA
Sparse LA
Monte Carlo
Optimization



What Next? (1/3)

Moving from "What to Build" to "How to Build"

- Technology
  - Refining the roadmap for software and algorithms on extreme-scale systems
  - Setting a prioritized list of software components for Exascale computing as outlined in the Roadmap
  - Assessing the short-term, medium-term and long-term software and algorithm needs of applications for peta/exascale systems

What Next? (2/3)

Moving from "What to Build" to "How to Build"

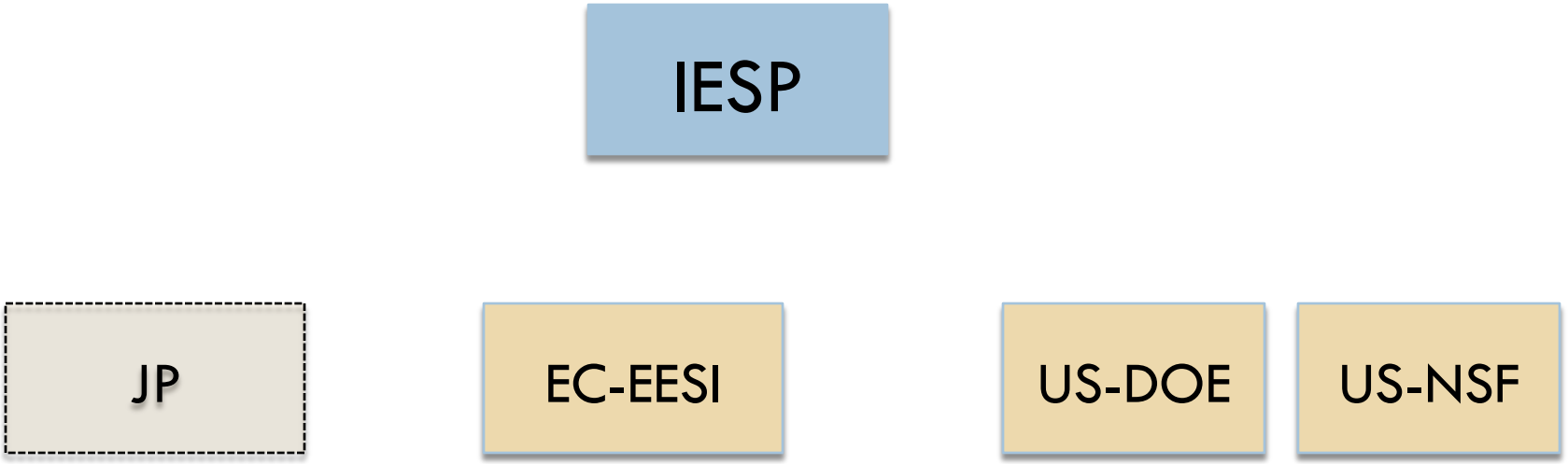
- Organization
  - Developing a governance, management, and organizational structure for the IESP
  - Exploring ways for funding agencies to coordinate their support of IESP-related R&D so that they complement each other
  - Exploring how laboratories, universities, and vendors can work together on coordinated HPC software
  - Creating a plan for working closely with HW vendors and application teams to co-design future architectures

What Next? (3/3)

Moving from "What to Build" to "How to Build"

- Execution
  - Developing a strategic plan for moving forward with the Roadmap
  - Creating a realistic timeline for constructing key organizational structures and achieving initial goals
  - Exploring community development techniques and risk plans to ensure key components are delivered on time
  - Exploring key components of any needed Intellectual Property agreements

Example Organizational Structure: Incubation Period (today):



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

EC and G8 Related

- G8 has a call out for "Interdisciplinary Program on Application Software towards Exascale Computing for Global Scale Issues"
  - 10 million € over three years
  - An initiative between Research Councils from Canada, France, Germany, Japan, Russia, the UK, and the USA
  - Proposals preselected due August 25th
- EC FP7 Exa-scale computing, software and simulation
  - Announcement due September 28, 2010
  - 25 million €

Where We Are Today:

- SC08 (Austin TX) meeting to generate interest
- Funding from DOE's Office of Science & NSF Office of Cyberinfratructure and sponsorship by Europeans and Asians
- US meeting (Santa Fe, NM) April 6-8, 2009
  - 65 people
- European meeting (Paris, France) June 28-29, 2009
  - Outline Report
- Asian meeting (Tsukuba Japan) October 18-20, 2009
  - Draft roadmap
  - Refine Report
- SC09 (Portland OR) BOF to inform others
  - Public Comment; Draft Report presented
- European meeting (Oxford, UK) April 13-14, 2010
  - Refine and prioritize roadmap
  - Explore governance structure and management models
- Maui Meeting October 18-19, 2010
- Kobe Meeting - Spring 2011

Apr 2009

Jun 2009

Oct 2009

Nov 2009

Apr 2010

Oct 2010

Next Steps

- Revise and extend initial draft
- Build management and collaboration plans
- Work with funding agencies to plan research activities

- Roadmap available at:
  - www.exascale.org

INTERNATIONAL EXASCALE SOFTWARE PROJECT ROADMAP 1.0

Jack Dongarra
Pete Beckman
Terry Moore
Patrick Aerts
Giovanni Aloiso
Jean-Claude Andre
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