

**THE INSTITUTE FOR ADVANCED ARCHITECTURES  
AND ALGORITHMS (IAA ALGORITHMS)**

**AND**

**THE EXTREME-SCALE ALGORITHMS & SOFTWARE  
INSTITUTE (EASI)**

Michael Heroux  
Sandia National Laboratories

**Fall Creek Falls Conference**  
**October 2010**

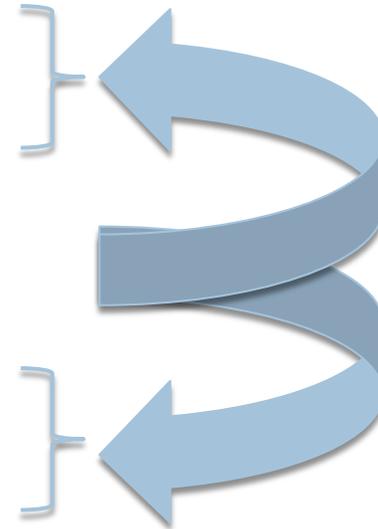
Research supported by DOE ASCR & NNSA

# About Co-Design (Simplified & Parochial)

- We have always done it, to some extent.
- Why the big interest now:
  - ▣ Power constraints imply ...
  - ▣ Node architecture changes imply...
  - ▣ OS/Runtime changes imply ...
  - ▣ Algorithm changes imply ...
  - ▣ New programming models imply ...
  - ▣ Application redesign ...
- The entire “stack” is in flux
- Opportunities for co-design.

Our  
Efforts:  
Algs &  
Libs

▣ OS/Runtime changes imply ...  
▣ Algorithm changes imply ...



Feedback  
& Advice  
to/from

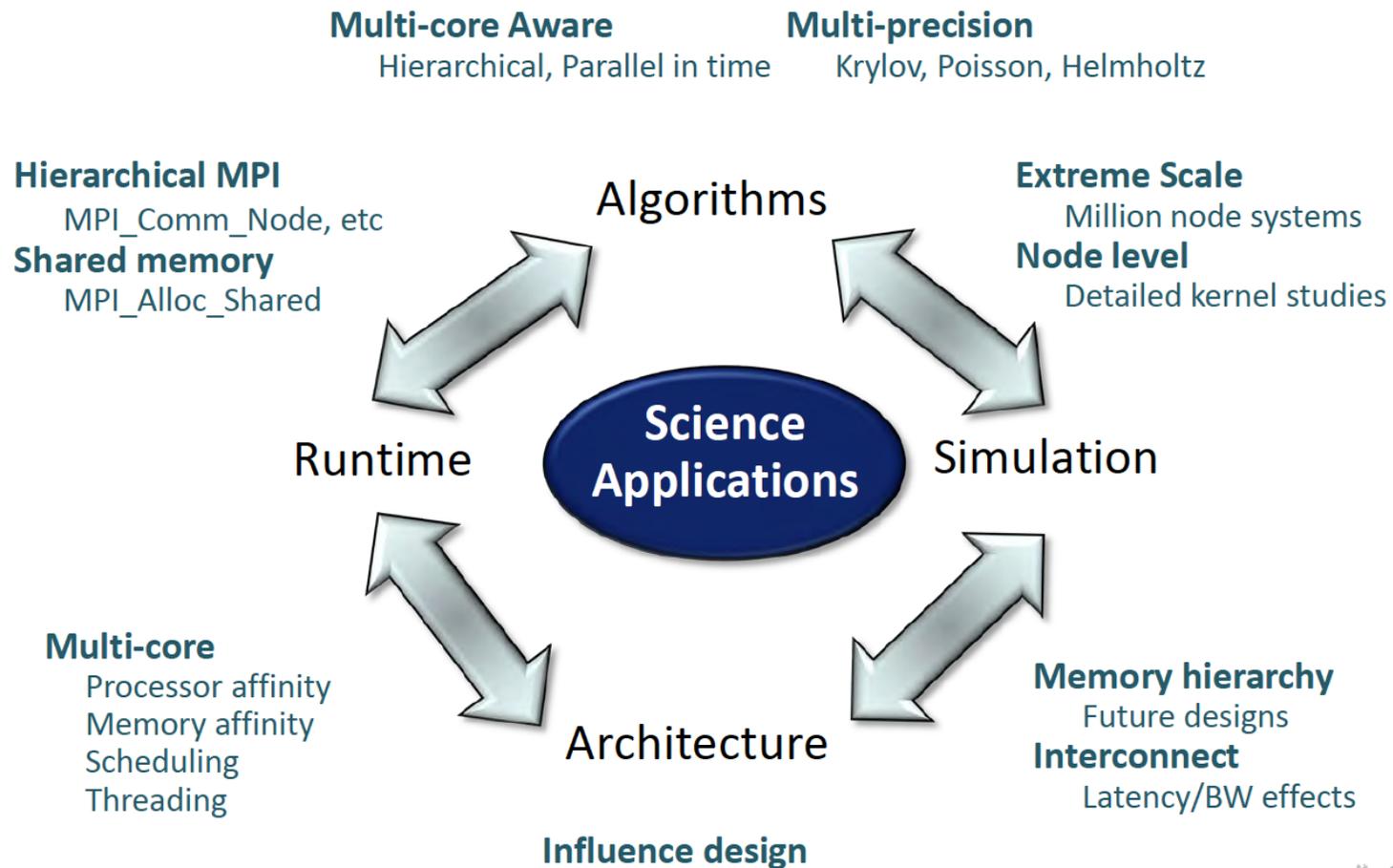
# IAA -Institute for Advanced Architectures and Algorithms



- Begun in FY2009 as Joint effort between Sandia National Labs and Oak Ridge National lab, it has a steering committee, advisory board, and underlying project(s)
- Focused R&D on key impediments to high performance in partnership with industry and academia
- Foster the integrated co-design of architectures and algorithms to enable more efficient and timely solutions to mission critical problems
- Impact vendor roadmaps through partnership and joint research and development
- IAA Algorithms Project is funded through this Institute

# IAA Algorithms Project Overview

## It all revolves around the science



# Technical Details

## Architecture Aware Algorithms



- Develop robust multi-precision algorithms:
  - ▣ Multi-precision Krylov and block Krylov solvers.
  - ▣ Multi-precision preconditioners: multi-level, smoothers.
  - ▣ Multi-resolution, multi-precision solver fast Poisson and Helmholtz solvers coupling direct and iterative methods
- Develop multicore-aware algorithms:
  - ▣ Hybrid distributed/shared preconditioners.
  - ▣ Develop hybrid programming support: Solver APIs that support MPI-only in the application and MPI+multicore in the solver.
  - ▣ Parallel in time algorithms such as Implicit Krylov Deferred Correction
- Develop the supporting architecture aware runtime:
  - ▣ Multi-level MPI communicators (Comm\_Node, Comm\_Net).
  - ▣ Multi-core aware MPI memory allocation (MPI\_Alloc\_Shared).
  - ▣ Strong affinity -process-to-core, memory-to-core placement.
  - ▣ Efficient, dynamic hybrid programming support for hierarchical MPI plus shared memory in the same application.

# IAA Algorithms Project Team

## Mix of math, CS, apps experts



- Climate (HOMME)
  - Mike Heroux, Mark Taylor, Chris Baker (SNL)
  - George Fann, Jun Jia, Kate Evans (ORNL)
- Materials and Chemistry (MADNESS)
  - George Fann, Judith Hill, Robert Harrison (ORNL)
  - Mike Heroux, Curt Janssen (SNL)
- Semiconductor device physics (Charon)
  - George Fann, John Turner (ORNL)
  - Mike Heroux, John Shadid, Paul Lin (SNL)
- Runtime and Affinity
  - Ron Brightwell, Kevin Pedretti, Brian Barrett (SNL)
  - Al Geist, Geoffroy Vallee, Gregg Koenig (ORNL)
- Simulation
  - Arun Rodrigues, Scott Hemmert (SNL),
  - Christian Engelmann, Kalyan Perumalla (ORNL)
  - Bob Numrich (UM), Bruce Jacobs (U Maryland), Sudhakar (GaTech)
- Project team includes key application developers
- Excellent cross site teaming

# Four DOE Math/CS Institutes

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

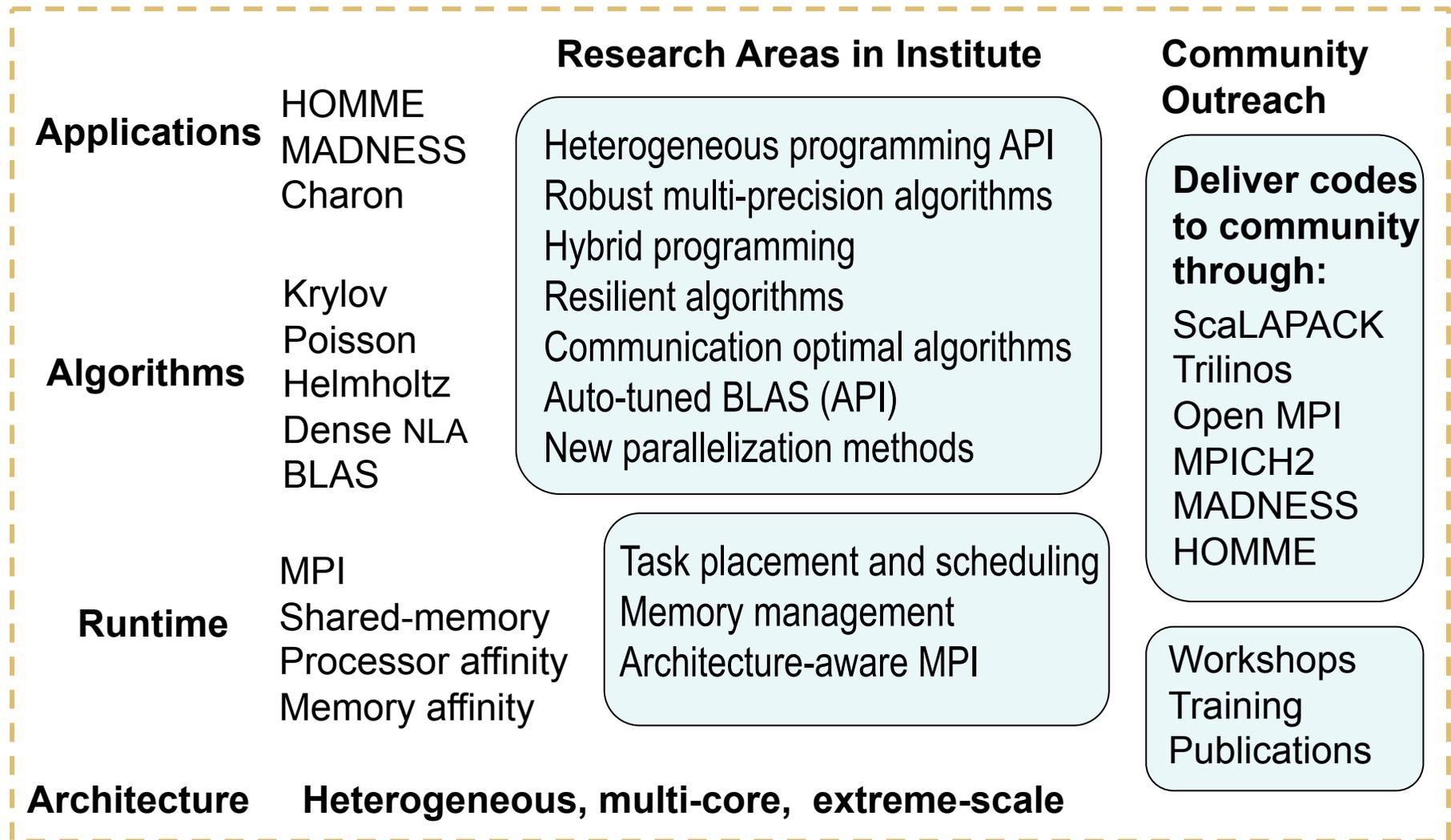
**Architecture-aware algorithms and associated runtime** to enable many science applications to better exploit the architectural features of DOE's petascale systems.

**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



# EASI Budget



- Duration: 3 years.
  - Started in Fall of 2009 for Labs
  - Spring/Summer 2010 for Universities
- Total funding over 3 years \$7.425M
  - ORNL \$1M/year
  - SNL \$1M/year
  - UTK ~\$150K/year
  - UCB ~\$150K/year
  - UIUC ~\$150K/year

# Basic Exascale Concerns: Trends, Manycore

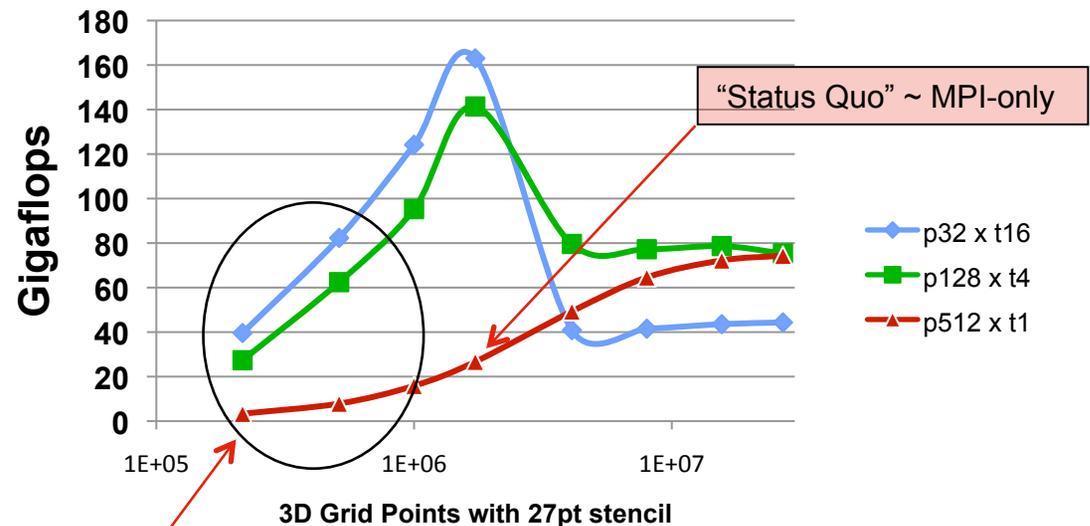
- Stein's Law: *If a trend cannot continue, it will stop.*

Herbert Stein, chairman of the Council of Economic Advisers under Nixon and Ford.

- Trends at risk:

- Power.
- Single core performance.
- Node count.
- Memory size & BW.
- Concurrency expression in existing Programming Models.

**Parallel CG Performance 512 Threads**  
32 Nodes = 2.2GHz AMD 4sockets X 4cores



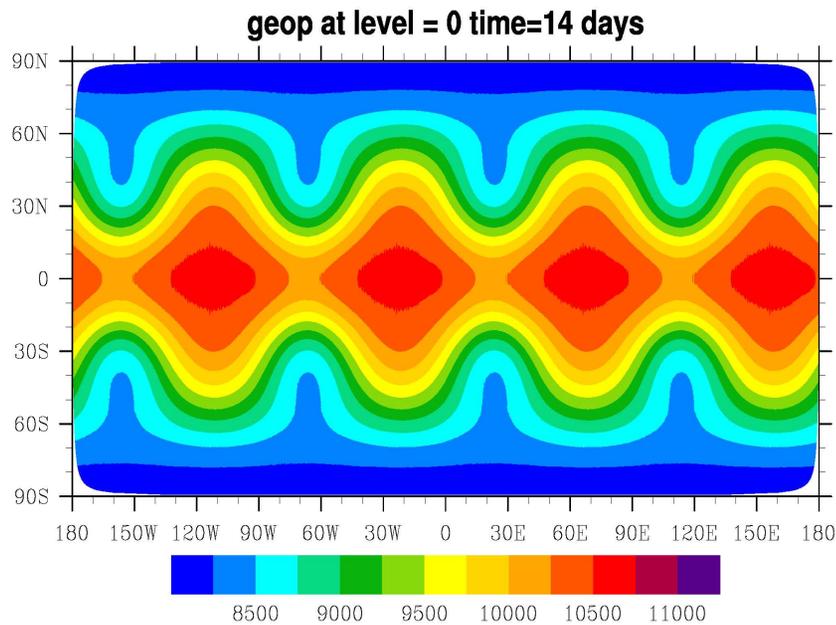
Strong Scaling Potential

Edwards: SAND2009-8196  
Trilinos ThreadPool Library v1.1.

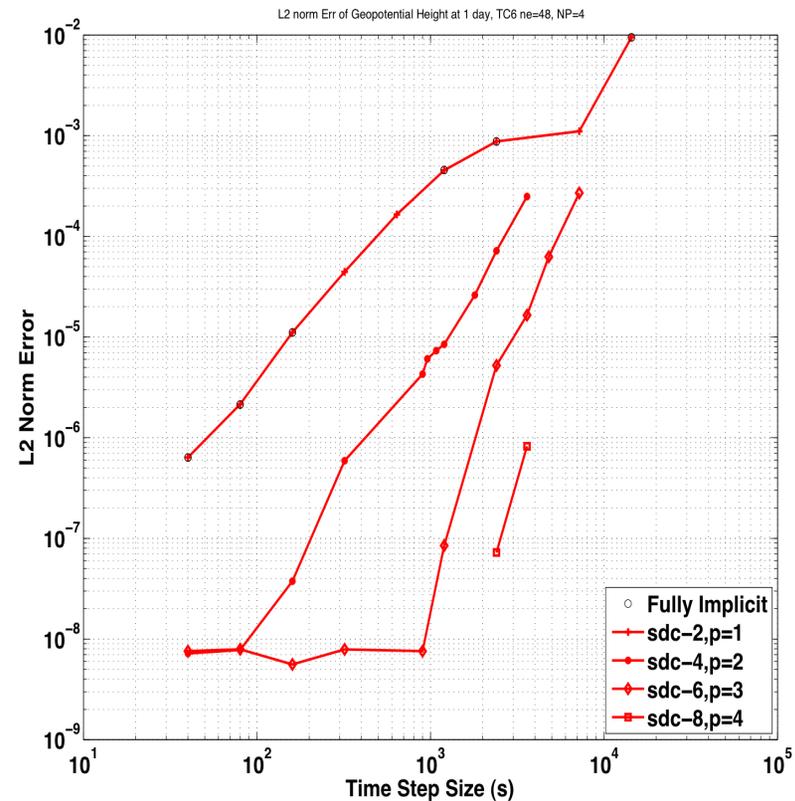
# ***Breaking Timestep Sequentiality***

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

## 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

## ***Minimizing Data Movement***

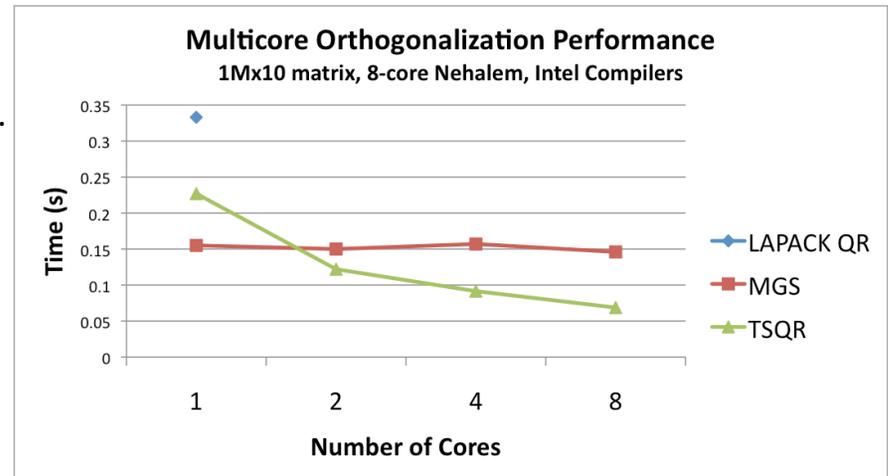
# Communication Avoiding Algorithms

17

- Goal: Algorithms that communicate as little as possible
- - 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  $Ax=b$ ,  $Ax=\lambda x$ 
    - ▣ Communication lower bounds, and algorithms that attain them (depending on sparsity structure)
      - Not in any libraries (yet). Coming to Trilinos.

# 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.
- **New orthogonalization algorithm:**
  - Tall Skinny QR factorization (**TSQR**).
  - Communicates less *and* more accurate than previous approaches.
  - Enables reliable, efficient  $s$ -step methods.
- 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

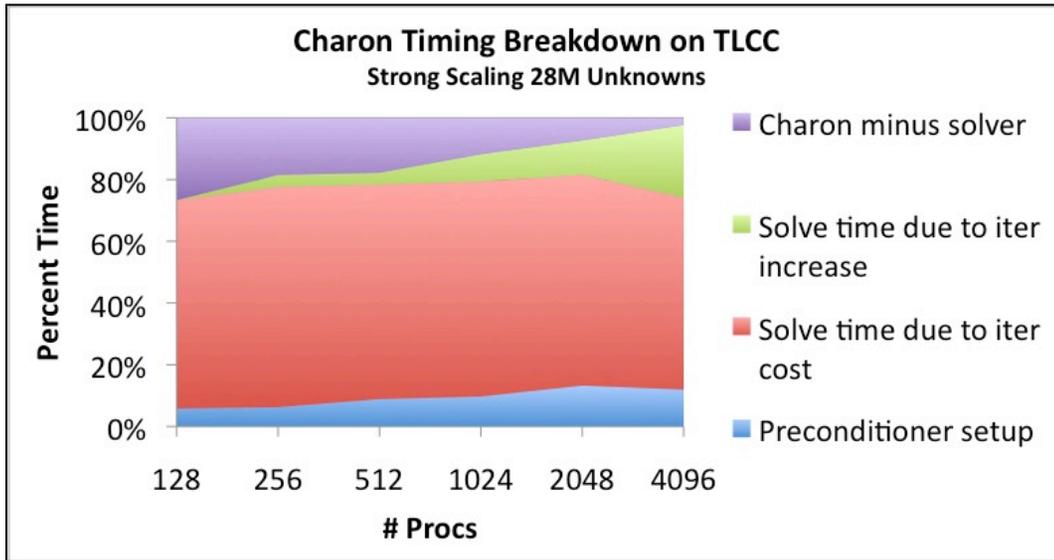
## TSQR capability:

- 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.**

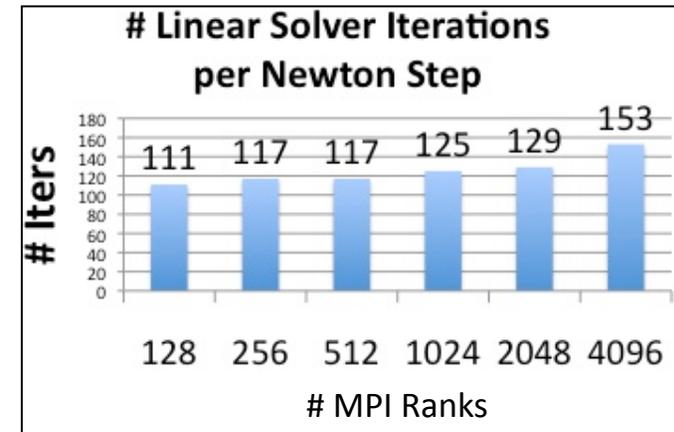


## ***Bi-Modal MPI and MPI+X***

# Preconditioners for Scalable Multicore Systems



Strong scaling of Charon on TLCC (P. Lin, J. Shadid 2009)



- Observe: Iteration count increases with number of subdomains.
- With scalable threaded smoothers (LU, ILU, Gauss-Seidel):
  - Solve with fewer, larger subdomains.
  - Better kernel scaling (threads vs. MPI processes).
  - Better convergence, More robust.
- Exascale Potential: Tiled, pipelined implementation.
- Three efforts:
  - Level-scheduled triangular sweeps (ILU solve, Gauss-Seidel).
  - Decomposition by partitioning
  - Multithreaded direct factorization

MPI Tasks	Threads	Iterations
4096	1	153
2048	2	129
1024	4	125
512	8	117
256	16	117
128	32	111

*Factors Impacting Performance of Multithreaded Sparse Triangular Solve*, Michael M. Wolf and Michael A. Heroux and Erik G. Boman, VECPAR 2010.

# MPI Shared Memory Allocation

## Idea:

- Shared memory alloc/free functions:
  - MPI\_Comm\_alloc\_mem
  - MPI\_Comm\_free\_mem
- Predefined communicators:
  - MPI\_COMM\_NODE – ranks on node
  - MPI\_COMM\_SOCKET – UMA ranks
  - MPI\_COMM\_NETWORK – inter node
- Status:
  - Available in current development branch of OpenMPI.
  - First “Hello World” Program works.
  - Incorporation into standard still not certain. Need to build case.
  - Next Step: Demonstrate usage with threaded triangular solve.
- Exascale potential:
  - Incremental path to MPI+X.
  - Dial-able SMP scope.

```
int n = ...;
double* values;
MPI_Comm_alloc_mem(
    MPI_COMM_NODE, // comm (SOCKET works too)
    n*sizeof(double), // size in bytes
    MPI_INFO_NULL, // placeholder for now
    &values); // Pointer to shared array (out)

// At this point:
// - All ranks on a node/socket have pointer to a shared buffer (values).
// - Can continue in MPI mode (using shared memory algorithms) or
// - Can quiet all but one:
int rank;
MPI_Comm_rank(MPI_COMM_NODE, &rank);
if (rank==0) { // Start threaded code segment, only on rank 0 of the node
    ...
}

MPI_Comm_free_mem(MPI_COMM_NODE, values);
```

*Bi-Modal MPI and MPI+Threads Computing on Scalable Multicore Systems,*  
Michael A. Heroux and Michael M. Wolf IPDPS 2011, submitted.



# ***Improving Data Placement***

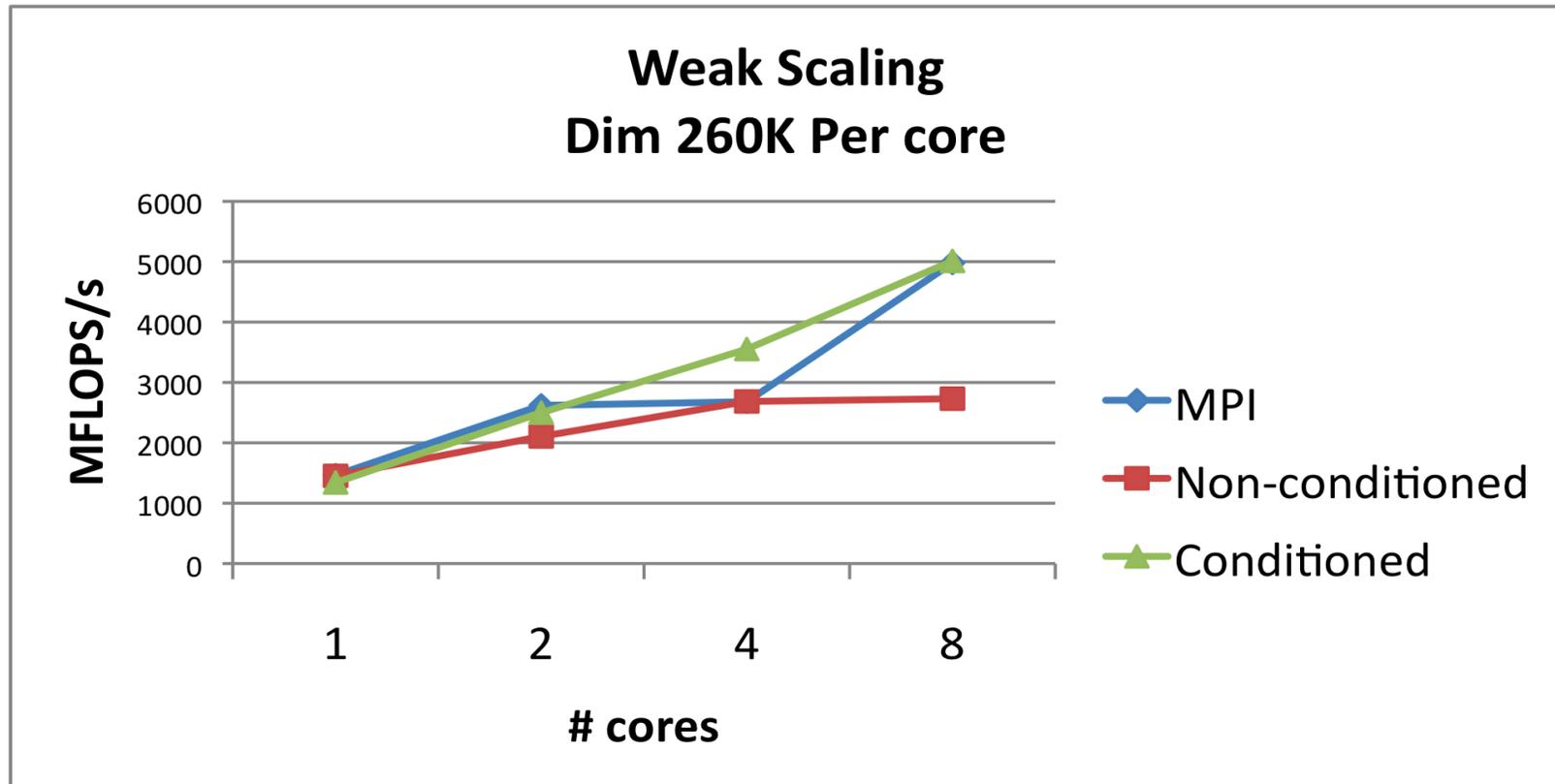
# Data Placement on NUMA

- Memory Intensive computations: Page placement has huge impact.
- Most systems: First touch.
- Application data objects:
  - Phase 1: Construction phase, e.g., finite element assembly.
  - Phase 2: Use phase, e.g., linear solve.
- Problem: First touch difficult to control in phase 1.
- Idea: Page migration.
  - Not new: SGI Origin. Many old papers on topic.

## Data placement experiments

- MiniApp: HPCCG (Mantevo Project)
- Construct sparse linear system, solve with CG.
- Two modes:
  - Data placed by assembly, not migrated for NUMA
  - Data migrated using parallel access pattern of CG.
- Results on dual socket quad-core Nehalem system.

# Weak Scaling Problem



- MPI and conditioned data approach comparable.
- Non-conditioned very poor scaling.

## Page Placement summary

- MPI+OpenMP (or any threading approach) is best overall.
- But:
  - Data placement is big issue.
  - Hard to control.
  - Insufficient runtime support.
- Current work:
  - Migrate on next-touch (MONT).
  - Considered in OpenMP (next version).
  - Also being studied in Kitten (Kevin Pedretti).

## ***Reducing Data Storage & Transfer Costs***

# Develop robust multi-precision algorithms

29

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

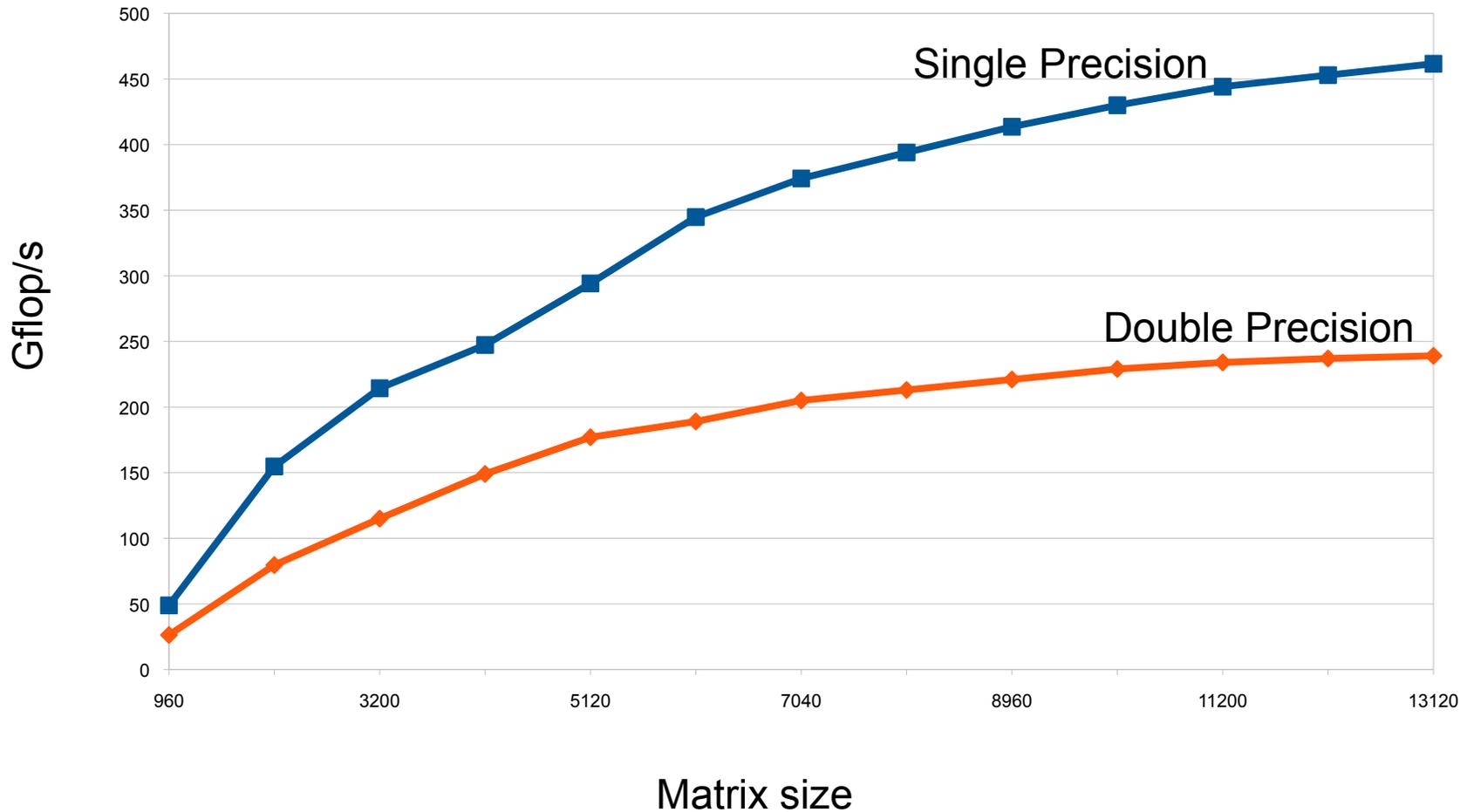
- Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

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

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

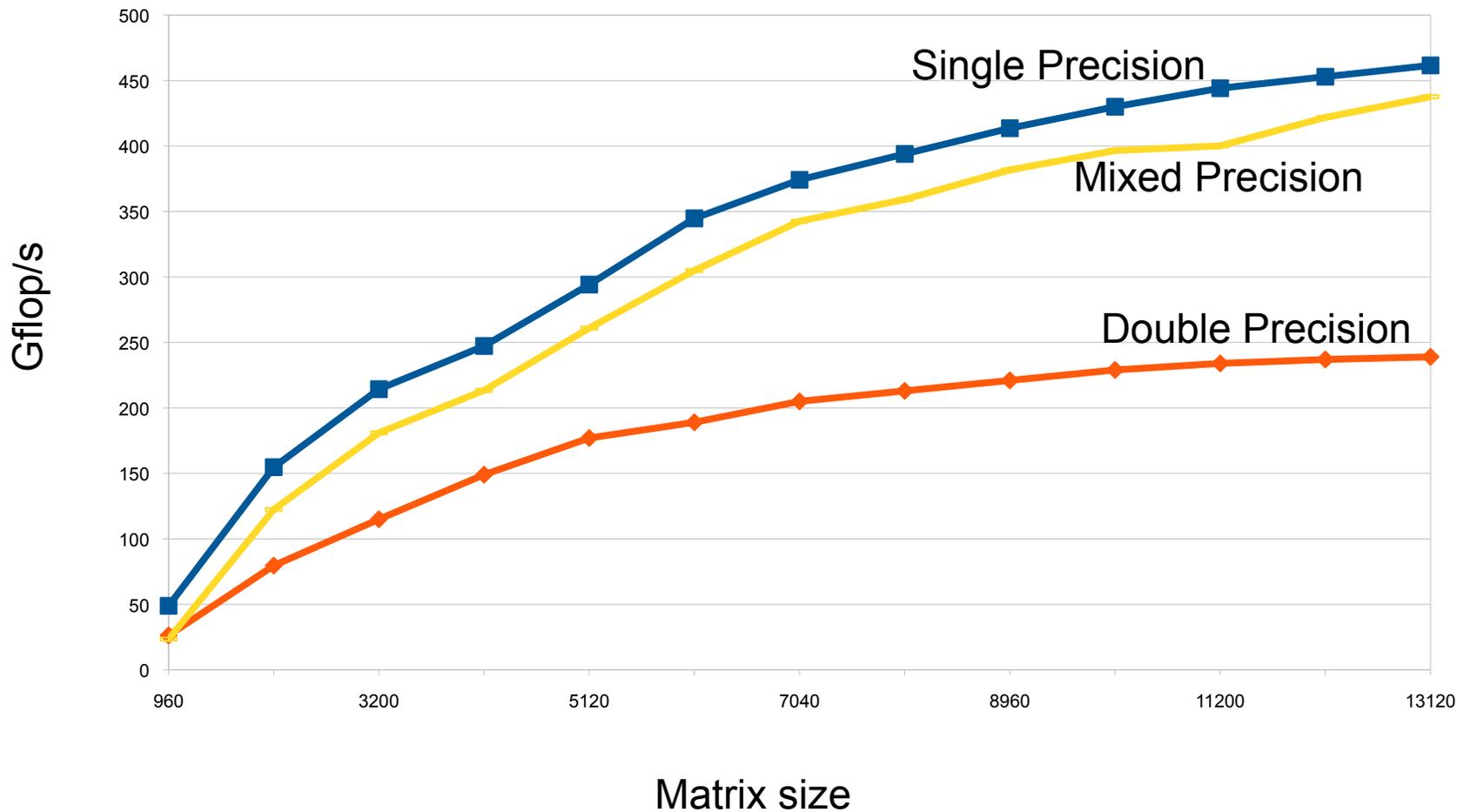
- Single precision is faster than DP because:
  - **Higher parallelism within floating point units**
    - 4 ops/cycle (usually) instead of 2 ops/cycle
  - **Reduced data motion**
    - 32 bit data instead of 64 bit data
  - **Higher locality in cache**
    - More data items in cache

$$Ax = b$$



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.

$$Ax = b$$

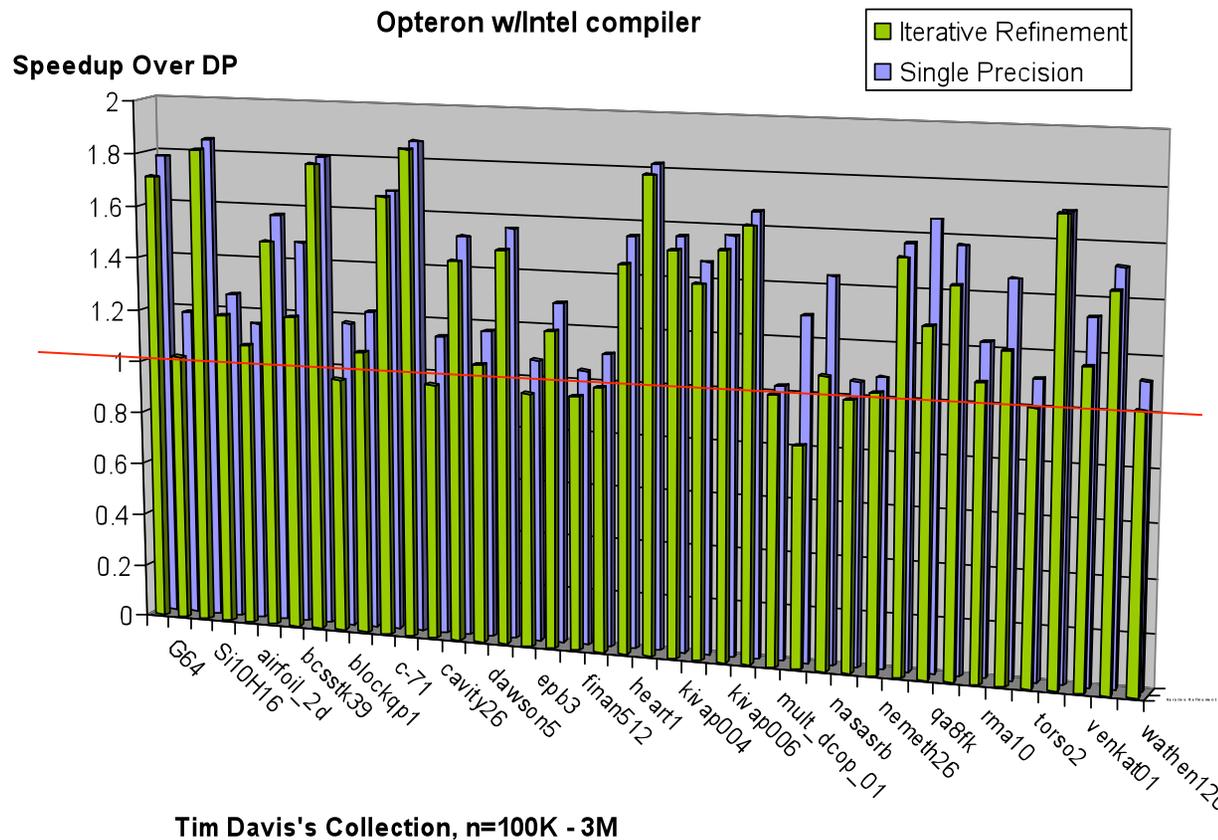


Tesla C2050, 448 CUDA cores (14 multiprocessors x 32) @ 1.15 GHz.,  
3 GB memory, connected through PCIe to a quad-core Intel @2.5 GHz.

# Sparse Direct Solver and Iterative Refinement

34

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  $i = 1, 2, \dots$

  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  $i = 1, 2, \dots$

  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

$q^{(i)} = Ap^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)T} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

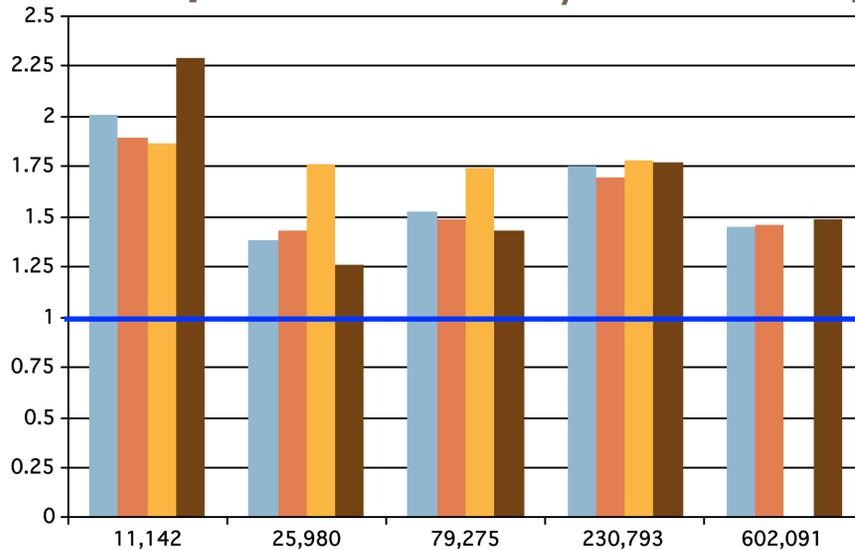
  check convergence; continue if necessary

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

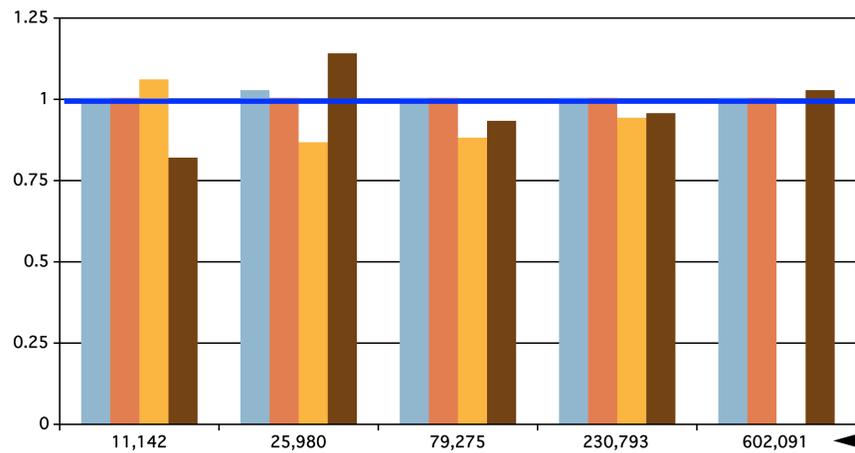
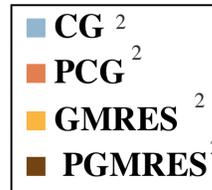
30



**Speedups** for mixed precision

Inner SP/Outer DP (SP/DP) iter. methods vs DP/DP  
(CG<sup>2</sup>, GMRES<sup>2</sup>, PCG<sup>2</sup>, and PGMRES<sup>2</sup> 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

# ***Hybrid CPU/GPU Computing***

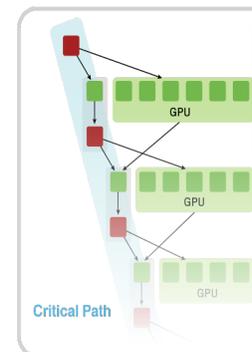
# Developing heterogeneous, multi-core-aware algorithms and software

## □ 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

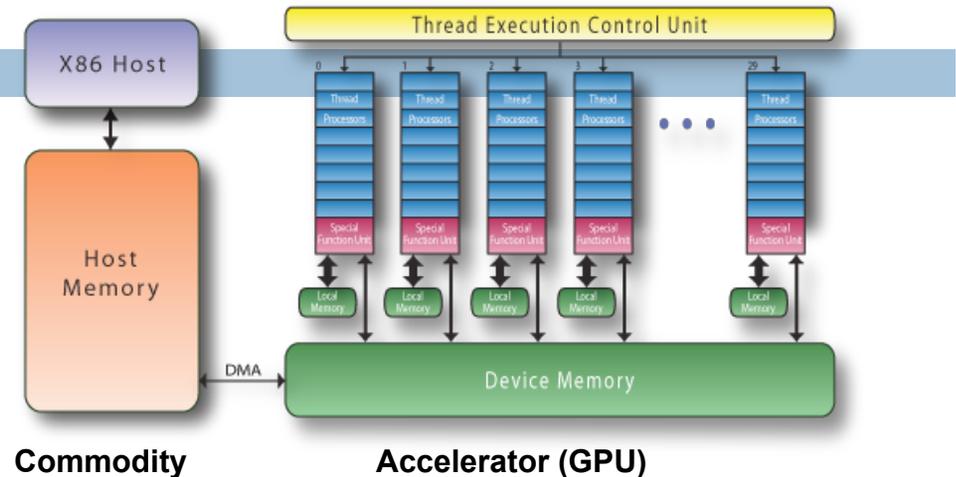
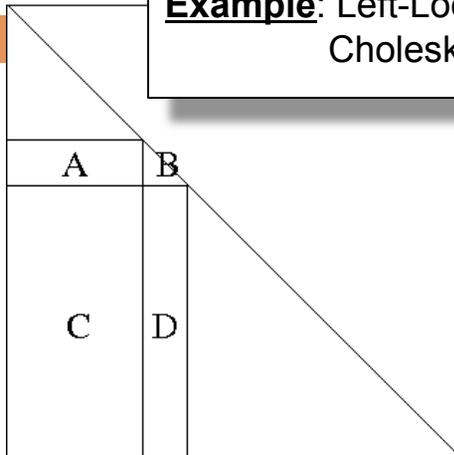
- Representing linear algebra algorithms as collections of **TASKS** and **DATA DEPENDENCIES** among them
- Properly **SCHEDULING** tasks' execution over multicore and GPU hardware components



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

(1)  $B = B - A \cdot A'$

(2)  $B = \text{chol}(B, \text{'lower'})$

(3)  $D = D - C \cdot A'$

(4)  $D = B \setminus D$

## LAPACK code

`ssyrk_("L", "N", &nb, &j, &mone, hA(j,0), ... )`

`spotrf_("L", &nb, hA(j, j), lda, info)`

`sgemm_("N", "T", &j, ... )`

`strsm_("R", "L", "T", "N", &j, ... )`

## Hybrid code

`cublasSsyrk('L', 'N', nb, j, mone, dA(j,0), ... )`

`cublasGetMatrix(nb, nb, 4, dA(j, j), *lda, hwork, nb)`

`cublasSgemm('N', 'T', j, ... )`

`spotrf_("L", &nb, hwork, &nb, info)`

`cublasSetMatrix(nb, nb, 4, hwork, nb, dA(j, j), *lda)`

`cublasStrsm('R', 'L', 'T', 'N', j, ... )`

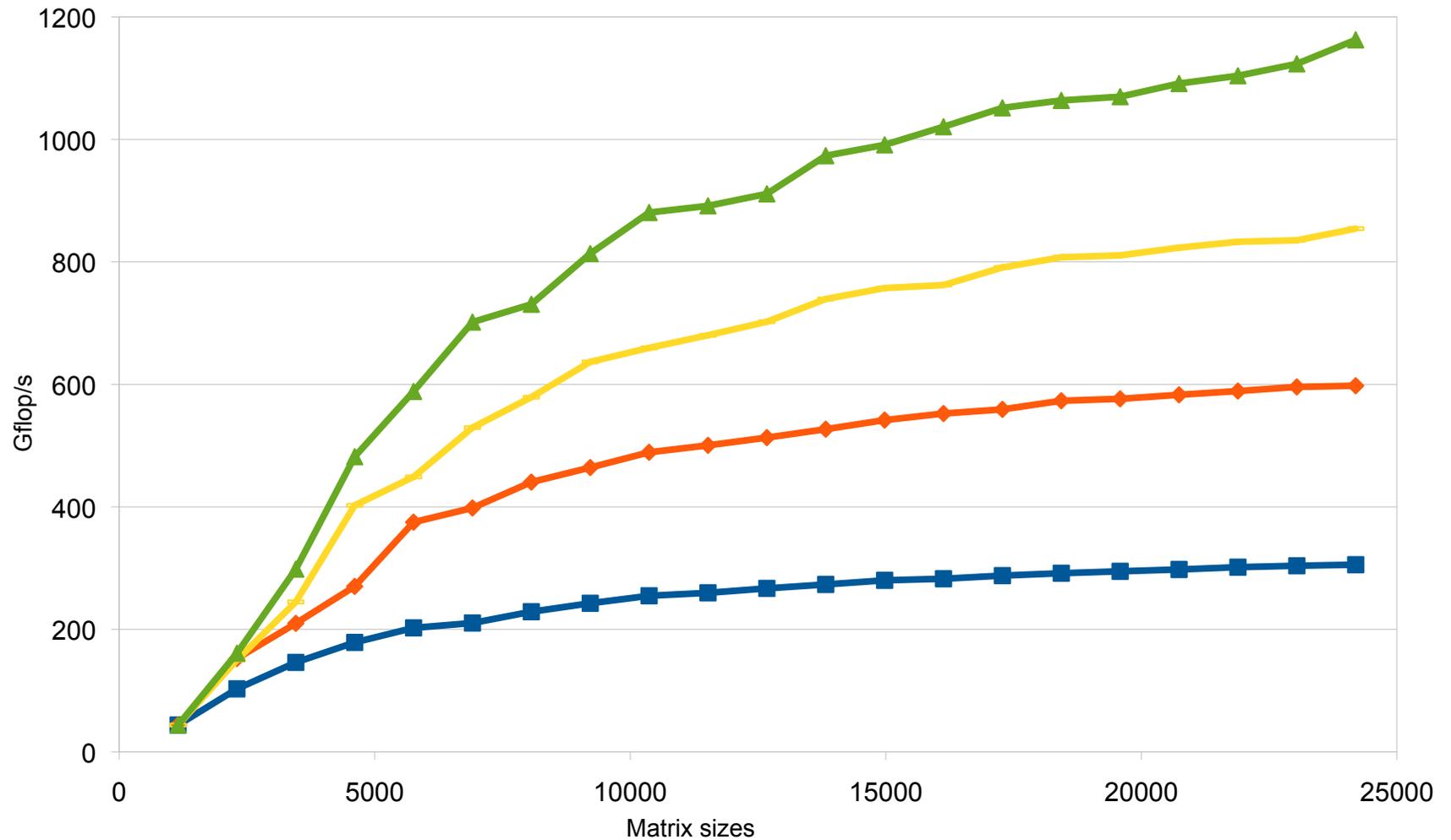
## CUDA implementation:

- `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

Parallel Performance of the hybrid SPOTRF (4 Opteron 1.8GHz and 4 GPU TESLA C1060 1.44GHz)

1CPU-1GPU 2CPUs-2GPUs 3CPUs-3GPUs 4CPUs-4GPUs



## ***Increasing Resilience to Soft Errors***

# Every calculation matters

Description	Iters	FLOPS	Recursive Residual Error	Solution Error
All Correct Calcs	35	343M	4.6e-15	1.0e-6
Iter=2, y[1] += 1.0 SpMV incorrect Ortho subspace	35	343M	6.7e-15	3.7e+3
Q[1][1] += 1.0 Non-ortho subspace	N/C	N/A	7.7e-02	5.9e+5

- Small PDE Problem: ILUT/GMRES
- Correct result: 35 Iters, 343M FLOPS
- 2 examples of a **single** bad op.
- Solvers:
  - 50-90% of total app operations.
  - Soft errors most likely in solver.
- Need new algorithms for soft errors:
  - Well-conditioned wrt errors.
  - Decay proportional to number of errors.
  - Minimal impact when no errors.

# Soft Error Resilience

- New Programming Model Elements: SW-enabled, highly reliable:
  - Data storage, paths.
  - Compute regions.
- Idea: New algorithms with minimal usage of high reliability.
- First new algorithm: Flexible-operator (FO)-GMRES.
  - Resilient to soft errors.
  - Only orthogonalization vectors and computations highly reliable.
  - Vast majority of data, ops done with base reliability:
    - Operator, preconditioner data
    - SpMV, Preconditioner application

Staffing: M. Heroux, M. Hoemmen

## ***Delivering The Software***

# Major Libraries



- PLASMA, MAGMA:
  - Next Generation of BLAS/LAPACK.
- OpenMPI, MPICH
  - Support for shared memory allocation.
- Trilinos
  - Vertical software stack.
  - Integrates other capabilities (e.g., PLASMA).
  - Framework for application development.

*“Are C++ templates safe? No, but they are good.”*

# Compile-time Polymorphism

Templates and Sanity upon a shifting foundation

## Software delivery:

- Essential Element of EASI

## How can we:

- Implement mixed precision algorithms?
- Implement generic fine-grain parallelism?
- Support hybrid CPU/GPU computations?
- Support extended precision?
- Explore redundant computations?
- Prepare for both exascale “swim lanes”?

## C++ templates only sane way:

- Moving to completely templated Trilinos libraries.
- Other important benefits.
- A usable stack exists now in Trilinos.

## Template Benefits:

- Compile time polymorphism.
- True generic programming.
- No runtime performance hit.
- Strong typing for mixed precision.
- Support for extended precision.
- Many more...

## Template Drawbacks:

- Huge compile-time performance hit:
  - But good use of multicore :)
  - Eliminated for common data types.
- Complex notation:
  - Esp. for Fortran & C programmers).
  - Can insulate to some extent.

# C++ Templates and Multi-precision

```
// Standard method prototype for apply matrix-vector multiply:  
template<typename ST, typename OT>  
CrsMatrix::apply(Vector<ST, OT> const& x, Vector<ST, OT>& y)
```

```
// Mixed precision method prototype (DP vectors, SP matrix):  
template<typename ST, typename OT>  
CrsMatrix::apply(Vector<ScalarTraits<ST>::dp(), OT> const& x,  
                 Vector<ScalarTraits<ST>::dp(), OT> & y)
```

```
// Sample usage:  
Tpetra::Vector<double, int> x, y;  
Tpetra::CrsMatrix<float, int> A;  
A.apply(x, y); // Single precision matrix applied to double precision vectors
```

# Tpetra Linear Algebra Library

- **Tpetra** is a templated version of the Petra distributed linear algebra model in Trilinos.

- ◆ Objects are templated on the underlying data types:

```
MultiVector<scalar=double, local_ordinal=int,
            global_ordinal=local_ordinal> ...
CrsMatrix<scalar=double, local_ordinal=int,
          global_ordinal=local_ordinal> ...
```

- ◆ Examples:

```
MultiVector<double, int, long int> V;
CrsMatrix<float> A;
```

Speedup of float over double  
in Belos linear solver.

float	double	speedup
18 s	26 s	1.42x

Scalar	float	double	double- double	quad- double
Solve time (s)	2.6	5.3	29.9	76.5
Accuracy	10 <sup>-6</sup>	10 <sup>-12</sup>	10 <sup>-24</sup>	10 <sup>-48</sup>

Arbitrary precision solves  
using Tpetra and Belos  
linear solver package

# Kokkos Node API

- **Kokkos** provides two main components:
  - **Kokkos memory model** addresses Difficulty #1
    - Allocation, deallocation and efficient access of memory
    - **compute buffer**: special memory used for parallel computation
    - New: Local Store Pointer and Buffer with size.
  - **Kokkos compute model** addresses Difficulty #2
    - Description of kernels for parallel execution on a node
    - Provides stubs for common parallel work constructs
    - Currently, **parallel for loop** and **parallel reduce**
- Code is developed around a polymorphic Node object.
- Supporting a new platform requires only the implementation of a new **node type**.

# Kokkos Memory Model

- A generic node model must at least:
  - support the scenario involving **distinct device memory**
  - allow **efficient** memory access under traditional scenarios
- Nodes provide the following memory routines:

```
ArrayRCP<T> Node::allocBuffer<T>(size_t sz);  
void Node::copyToBuffer<T>( T * src,  
                             ArrayRCP<T> dest);  
void Node::copyFromBuffer<T>(ArrayRCP<T> src,  
                              T * dest);  
ArrayRCP<T> Node::viewBuffer<T>(ArrayRCP<T> buff);  
void Node::readyBuffer<T>(ArrayRCP<T> buff);
```

# Kokkos Compute Model

- How to make shared-memory programming generic:
  - **Parallel reduction** is the intersection of `dot()` and `norm1()`
  - **Parallel for loop** is the intersection of `axpy()` and mat-vec
  - We need a way of **fusing** kernels with these basic **constructs**.
- Template meta-programming is **the answer**.
  - This is the same approach that Intel TBB and Thrust take.
  - Has the effect of requiring that Tpetra objects be templated on Node type.
- Node provides generic parallel constructs, user fills in the rest:

```
template <class WDP>
void Node::parallel_for(
    int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- loop body via `WDP::execute(i)`

```
template <class WDP>
WDP::ReductionType Node::parallel_reduce(
    int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- reduction type `WDP::ReductionType`
- element generation via `WDP::generate(i)`
- reduction via `WDP::reduce(x, y)`

## Example Kernels: `axpy()` and `dot()`

```
template <class WDP>
void
Node::parallel_for(int beg, int end,
                  WDP workdata   );
```

```
template <class WDP>
WDP::ReductionType
Node::parallel_reduce(int beg, int end,
                    WDP workdata   );
```

```
template <class T>
struct AxyOp {
    const T * x;
    T * y;
    T alpha, beta;
    void execute(int i)
    { y[i] = alpha*x[i] + beta*y[i]; }
};
```

```
template <class T>
struct DotOp {
    typedef T ReductionType;
    const T * x, * y;
    T identity()      { return (T)0;      }
    T generate(int i) { return x[i]*y[i]; }
    T reduce(T x, T y) { return x + y;    }
};
```

```
AxyOp<double> op;
op.x = ...;  op.alpha = ...;
op.y = ...;  op.beta  = ...;
node.parallel_for< AxyOp<double> >
    (0, length, op);
```

```
DotOp<float> op;
op.x = ...;  op.y = ...;
float dot;
dot = node.parallel_reduce< DotOp<float> >
    (0, length, op);
```

# Hybrid Timings (Tpetra)

- Tests of a simple iterations:
  - **power method**: one sparse mat-vec, two vector operations
  - **conjugate gradient**: one sparse mat-vec, five vector operations

- DNVS/x104 from UF Sparse Matrix Collection (100K rows, 9M entries)

- NCCS/ORNL **Lens** node includes:

- one NVIDIA Tesla C1060
- one NVIDIA 8800 GTX
- Four AMD quad-core CPUs

- Results are **very tentative!**

- suboptimal GPU traffic
- bad format/kernel for GPU
- bad data placement for threads

Node	PM (mflop/s)	CG (mflop/s)
Single thread	140	614
8800 GPU	1,172	1,222
Tesla GPU	1,475	1,531
Tesla + 8800	981	1,025
16 threads	816	1,376
<b>1 node</b>		
15 threads + Tesla	867	1,731
<b>2 nodes</b>		
15 threads + Tesla	1,677	2,102

# Summary



- IAA Algs & EASI: Focused on explicit co-design
  - Architecture-runtime-aware algorithms
  - Algorithm-architecture-aware runtime
- Lots of exploratory projects:
  - Multi-precision, CA, FT, Parallel Time
  - Multicore, GPGPU, manycore
- With concrete delivery in libraries:
  - PLASMA, MAGMA, OpenMPI, MPICH, Trilinos