SAND2010-3120P

Extreme Algorithms and Software Co-Design: It's EASI!

Michael A. Heroux
Scalable Algorithms

CIS External Panel Review
May 26-28, 2010

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

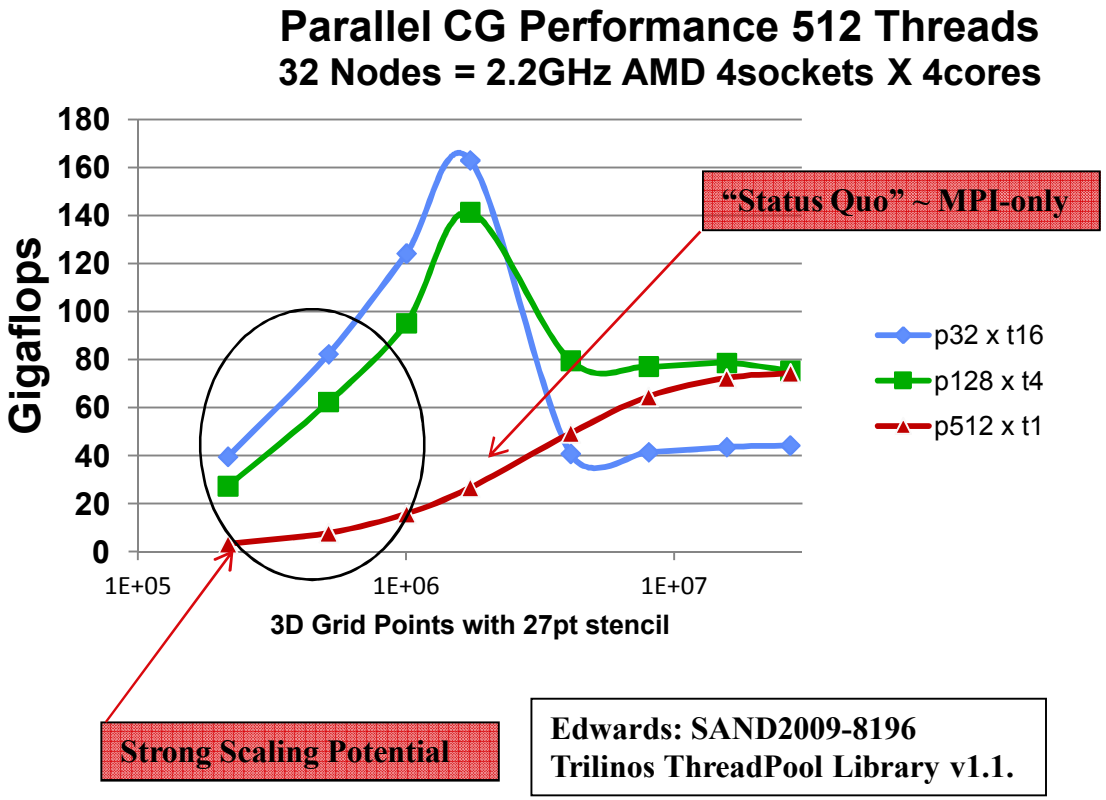
Sandia National Laboratories

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.

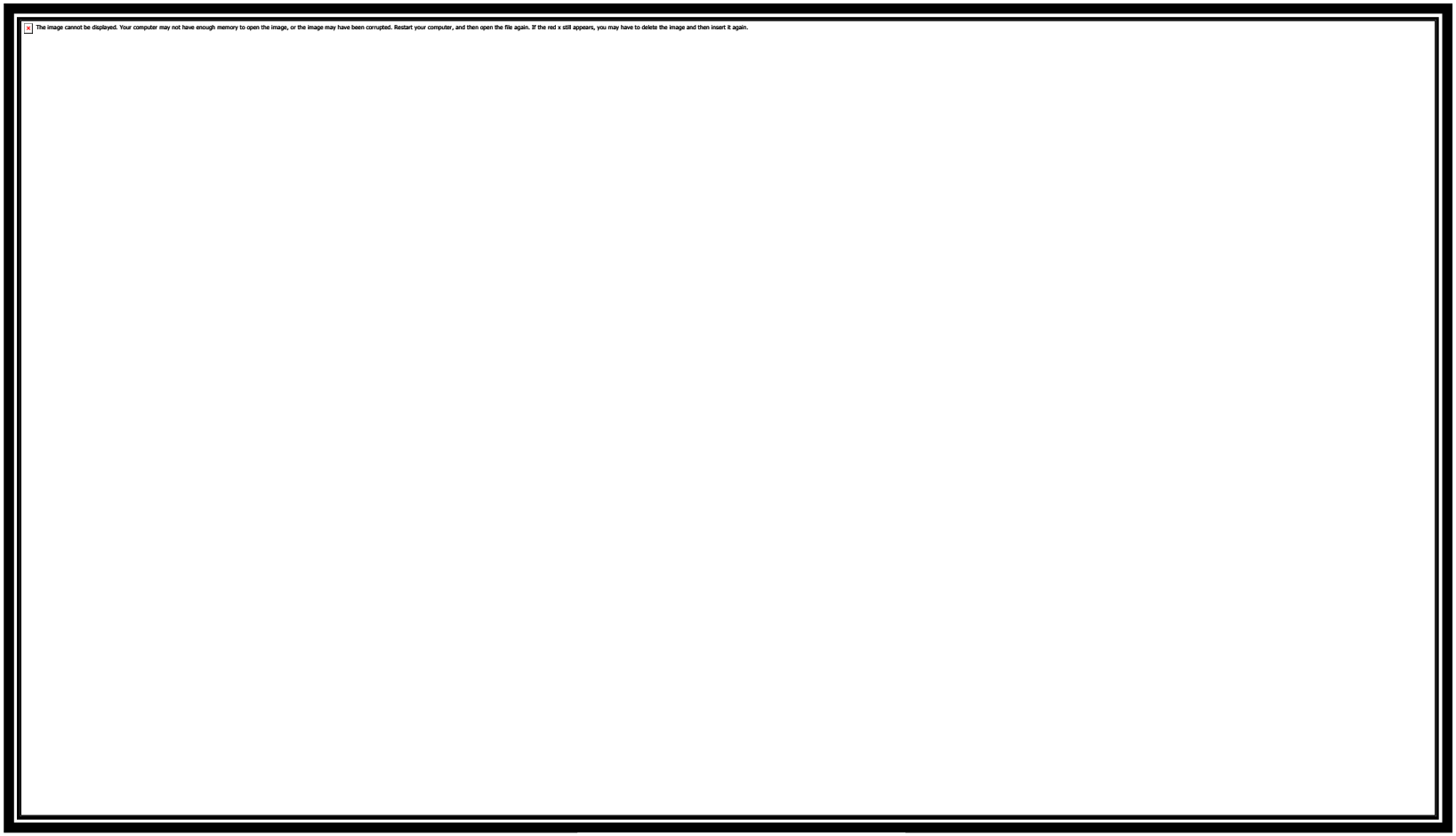


Sandia National Laboratories

Extreme-scale Algorithms & SW Institute (EASI)

- DOE Office of Science Math/CS Institute
- $7.425M ($3M SNL), 3-year project.
- Algorithms and Software.

IAA/Algorithms ➔ EASI ➔ X-Stack (We Hope)



EASI Project Team

- B. Barrett, Boman, Brightwell, Heroux (SNL)
- Baker, Fann, Geist*, Vallée (ORNL)
- Demmel (UC Berkeley)
- Dongarra (UTK)
- Gropp (UIUC)
- Mark Hoemmen (Demmel, UC)
- Siva Rajamanickam (Davis, FL)
- Michael Wolf (Heath, UIUC)

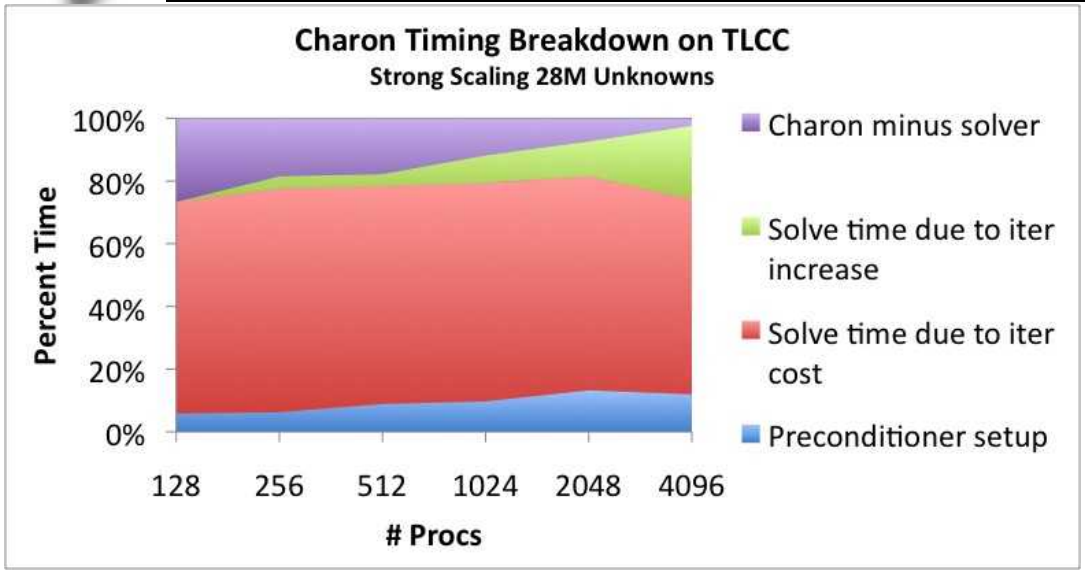
MPI Shared Memory Allocation

Idea:

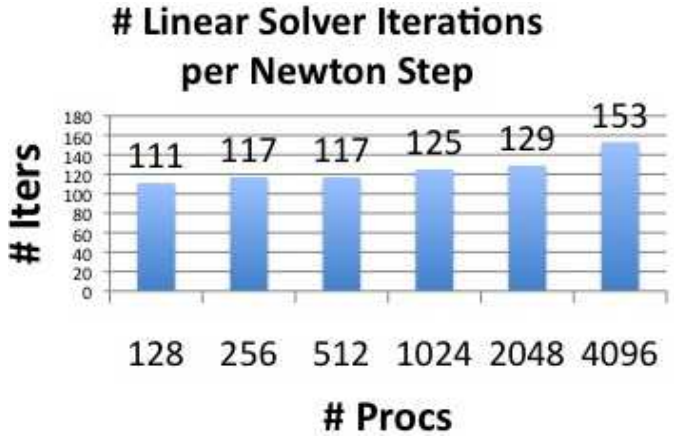
- Shared memory alloc/free functions:
  - MPI_Comm_alloc_mem
  - MPI_Comm_free_mem
- Predefined communicators:
  - MPI_COMM_NODE – ranks on node
  - MPI_COMM_SOCKET – UMA ranks
  - MPI_COMM_NETWORK – inter node
- Status:
  - Available in current development branch of OpenMPI.
  - First "Hello World" 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);
```

Preconditioners for Scalable Multicore Systems



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

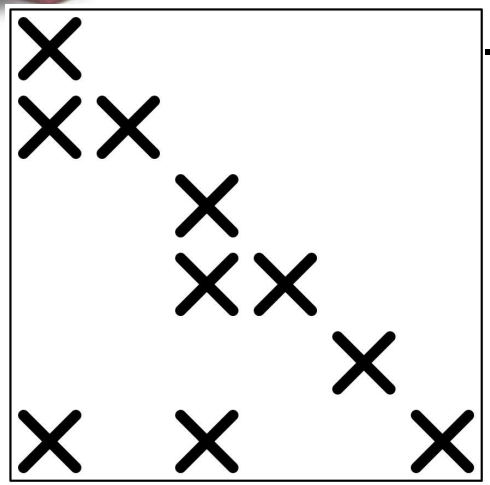


- Observe: Iteration count increases with number of subdomains.
- With scalable threaded triangular solves
  - Solve triangular system on larger subdomains.
  - Reduce number of subdomains.
- Goal:
  - Better kernel scaling (threads vs. MPI processes).
  - Better convergence, More robust.
- Note: App (-solver) scales very well in MPI-only mode.
- Exascale Potential: Tiled, pipelined implementation.

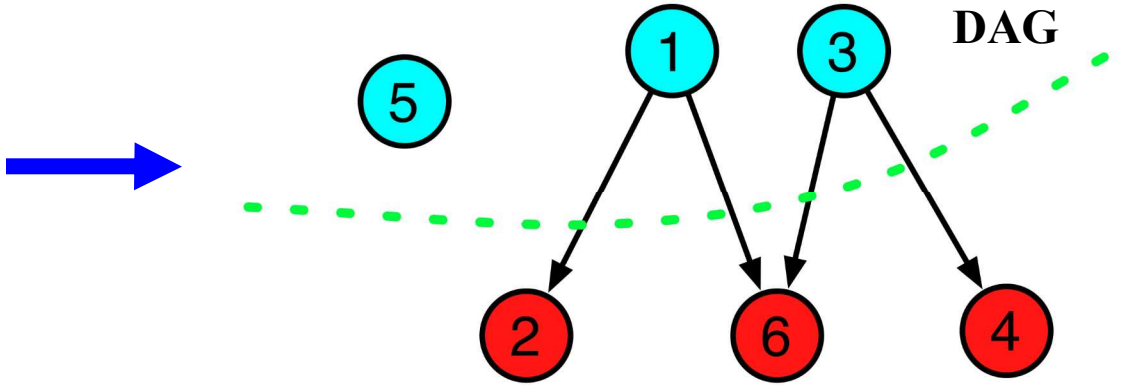
MPI Tasks	Threads	Iterations
4096	1	153
2048	2	129
1024	4	125
512	8	117
256	16	117
128	32	111

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

Level Set Triangular Solver



L



Triangular Solve:
- Critical Kernel
  - MG Smoothers
  - Incomplete IC/ILU
- Naturally Sequential
- Building on classic algorithms:
  - Level Sched:
    - circa 1990.
    - Vectorization.
  - New: Generalized.

$$\tilde{L} = PLP^T = \begin{bmatrix} D_1 & & & & \\ A_{2,1} & D_2 & & & \\ A_{3,1} & A_{3,2} & D_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ A_{l,1} & A_{l,2} & A_{l,3} & \dots & D_l \end{bmatrix}$$

Permuted System

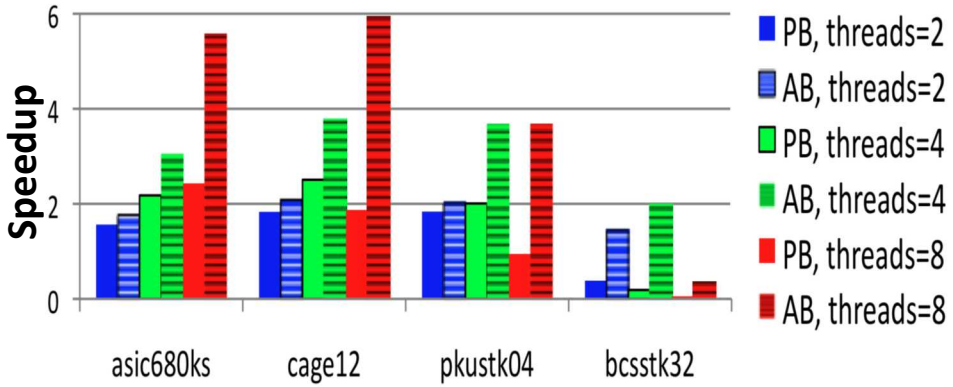
$$\begin{aligned} \tilde{x}_1 &= D_1^{-1}\tilde{y}_1 \\ \tilde{x}_2 &= D_2^{-1}\left(\tilde{y}_2 - A_{2,1}\tilde{x}_1\right) \\ &\vdots \\ \tilde{x}_l &= D_l^{-1}\left(\tilde{y}_l - A_{l,1}\tilde{x}_1 - \ldots - A_{l,l-1}\tilde{x}_{l-1}\right) \end{aligned}$$

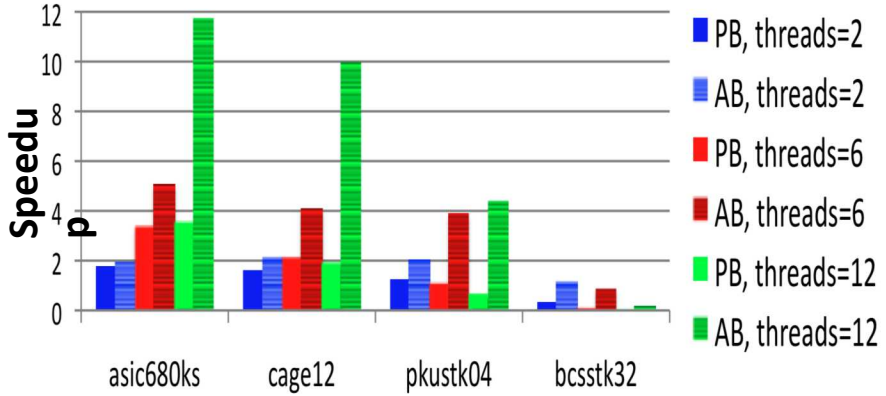
Multi-step Algorithm

Triangular Solve Results

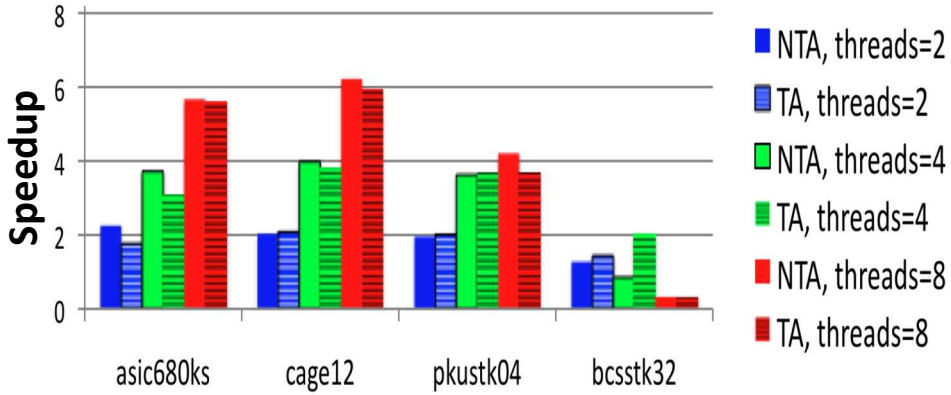
Name	N	nnz	N/nlevels	application area
asic680ks	682,712	2,329,176	13932.9	circuit simulation
cage12	130,228	2,032,536	1973.2	DNA electrophoresis
pkustk04	55,590	4,218,660	149.4	structural engineering
bcsstk32	44,609	2,014,701	15.1	structural engineering

Passive (PB) vs. Active (AB) Barriers: Critical for Performance

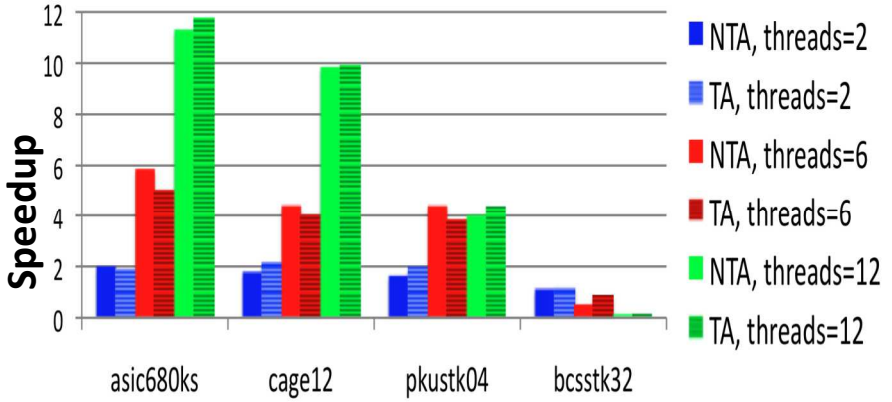




Nehalem



Istanbul



AB + No Thread Affinity (NTA) vs. AB + Thread Affinity (TA) : Also Helpful

Level sets: Trilinos/Isorropia Core Kernel Timings: Trilinos/Kokkos.

"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.

Summary, Additional Topics

EASI is not Easy: Great Team, Good Start, Challenging Decade Ahead

Summary

- MPI+X is essential, even now.
- Hybrid MPI-only/MPI+threading important:
  - Working prototype in OpenMPI.
  - Enables dual mode.
  - Allows gradual app migration to full MPI+threading.
  - Co-design with Runtime/OS.
- Threaded programming:
  - Challenging, but ultimately useful.
  - Should dramatically reduce memory/core requirement.
  - Structuring framework for "the average programmer" still an issue.
  - Co-design (barriers): PMs, HW
- C++ templates and meta-programming:
  - Write-once, run-anywhere portability.
  - Mixed-precision, extended precision, redundant computation.
  - Threaded parallelism.
  - Hybrid CPU/GPU computations.

Additional Topics

- Software Development and Delivery.
- Evolving Parallel Programming Model.
- Mixed-precision computations.
- Manycore Node API.
- Hybrid CPU/GPU computing.
- Data Placement for NUMA.
- Resilient Algorithms.
- New Core Linear Algebra Needs for Advanced Modeling & Simulation.
- Communication-avoiding Algorithms.
- Threaded incomplete factorizations.
- Scenarios for extra-precise computations.

Extra Slides

EASI-related External Collaborations

- IESP: Int'l Exascale SW Project
  - Quarterly meetings: US, Europe, Asia.
  - Draft Software Plan.
  - Dosanjh, Heroux from SNL (Secretary for software write-up).
- X-Stack: DOE Office of Science Call (Proposal completed).
  - 4 labs (SNL is only NNSA lab).
  - 6 universities.
- TOPS-2: Toward Optimal Petascale Simulation.
  - Forum for DOE library efforts to collaborate.
  - ANL, LBL, LLNL, SNL, universities.
- SciDAC-e: U of Texas, Wheeler.

Software Development and Delivery

Solver Software Stack

Trilinos

Phase I packages: SPMD, int/double | **Phase II packages: Templated**

Optimization **Unconstrained:** **Constrained:**	Find $u \in \Re^n$ that minimizes $g(u)$ Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x,u)$ s.t. $f(x,u) = 0$	MOOCHO
Bifurcation Analysis	Given nonlinear operator $F(x,u) \in \Re^{n+m}$ For $F(x,u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$	LOCA
Transient Problems **DAEs/ODEs:**	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0,T], x(0) = x_0, \dot{x}(0) = x_0'$ for $x(t) \in \Re^n, t \in [0,T]$	Rythmos
Nonlinear Problems	Given nonlinear operator $F(x) \in \Re^m \rightarrow \Re$ Solve $F(x) = 0$ $x \in \Re^n$	NOX
Linear Problems **Linear Equations:** **Eigen Problems:**	Given Linear Ops (Matrices) $A, B \in \Re^{m \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \Re^n$, $\lambda \in$	Anasazi Ifpack, ML, etc... AztecOO
Distributed Linear Algebra **Matrix/Graph Equations:** **Vector Problems:**	Compute $y = Ax; A = A(G); A \in \Re^{m \times n}, G \in \Im^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \Re^n$	Epetra Teuchos

Sensitivities (Automatic Differentiation: Sacado)

Solver Software Stack

Trilinos

Phase I packages | **Phase II packages** | **Phase III packages: Manycore*, templated**

Optimization

Unconstrained:

Constrained:

Find $u \in \Re^n$ that minimizes $g(u)$

Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x,u)$ s.t. $f(x,u) = 0$

MOOCHO

Bifurcation Analysis

Given nonlinear operator $F(x,u) \in \Re^{n+m}$

For $F(x,u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$

LOCA | T-LOCA

Transient Problems

DAEs/ODEs:

Solve $f(\dot{x}(t), x(t), t) = 0$

$t \in [0,T], x(0) = x_0, \dot{x}(0) = x_0'$

for $x(t) \in \Re^n, t \in [0,T]$

Rythmos

Nonlinear Problems

Given nonlinear operator $F(x) \in \Re^m \rightarrow \Re$

Solve $F(x) = 0 \quad x \in \Re^n$

NOX | T-NOX

Linear Problems

Linear Equations:

Eigen Problems:

Given Linear Ops (Matrices) $A, B \in \Re^{m \times n}$

Solve $Ax = b$ for $x \in \Re^n$

Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \Re^n, \quad \lambda \in$

Anasazi

AztecOO Ifpack, ML, etc... | Belos* T-Ifpack*, T-ML*, etc.

Sensitivities (Automatic Differentiation: Sacado)

Distributed Linear Algebra

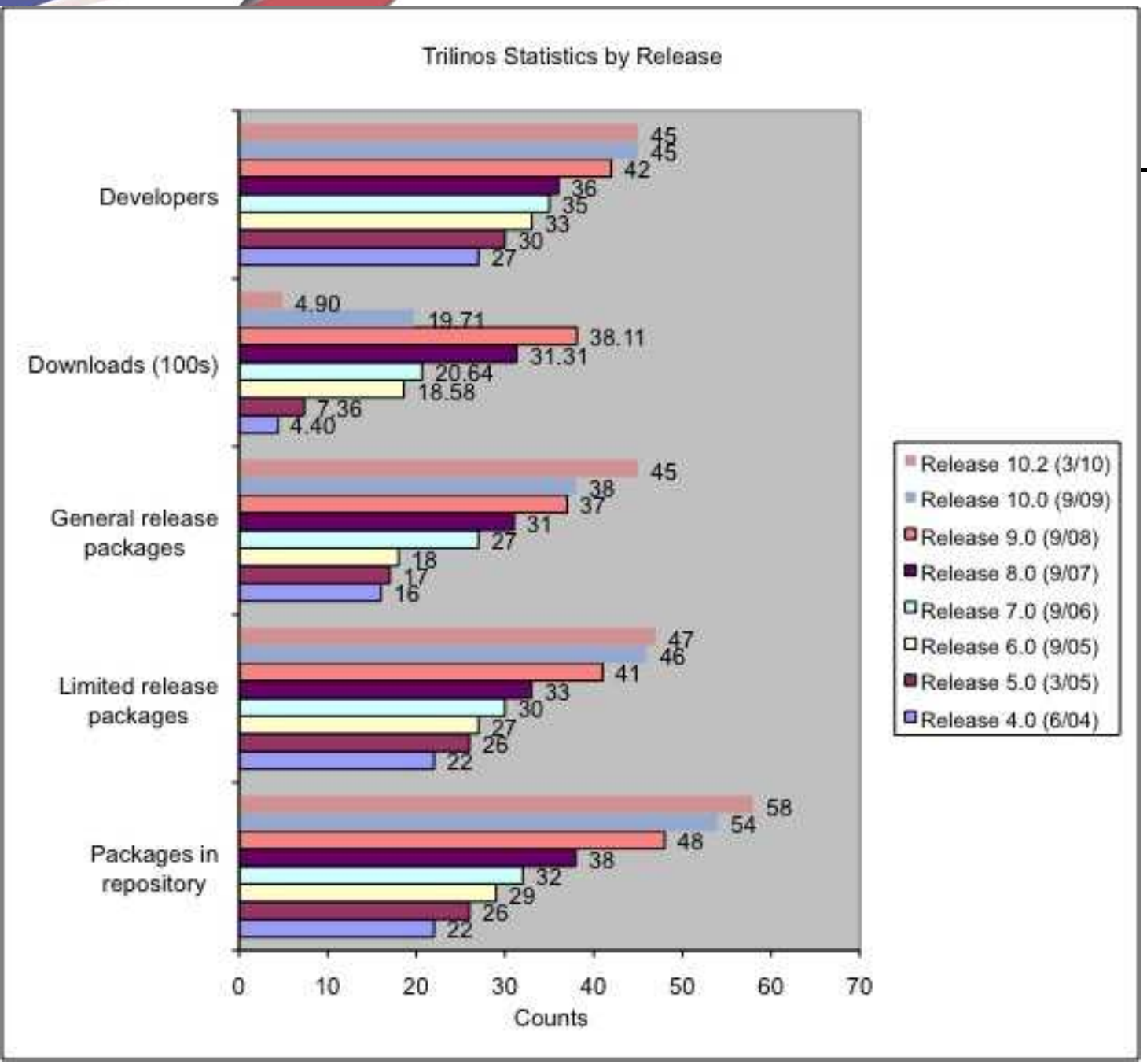
Matrix/Graph Equations: Compute $y = Ax; A = A(G); A \in \Re^{m \times n}, G \in \Im^{m \times n}$

Vector Problems: Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \Re^n$

Epetra | Tpetra* Kokkos

Teuchos

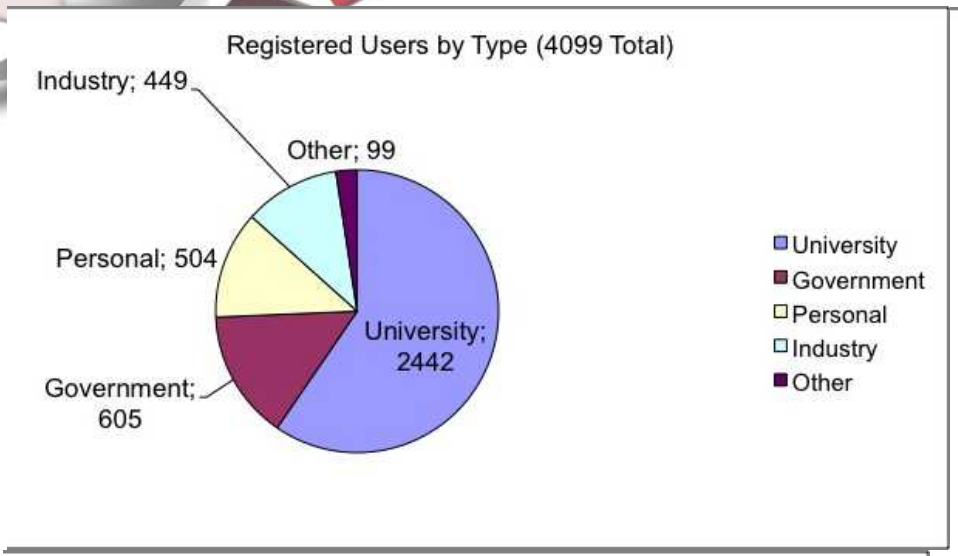
Trilinos Statistics

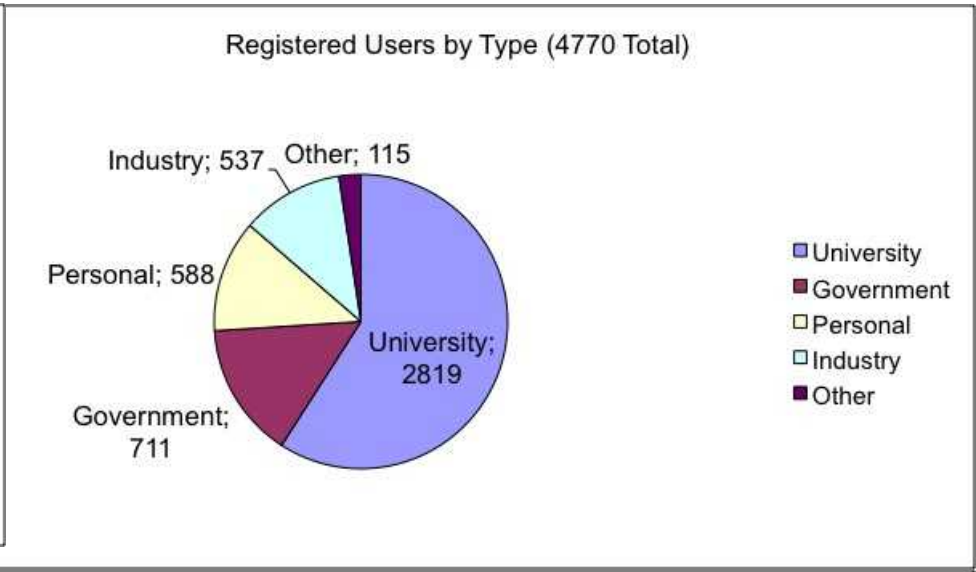


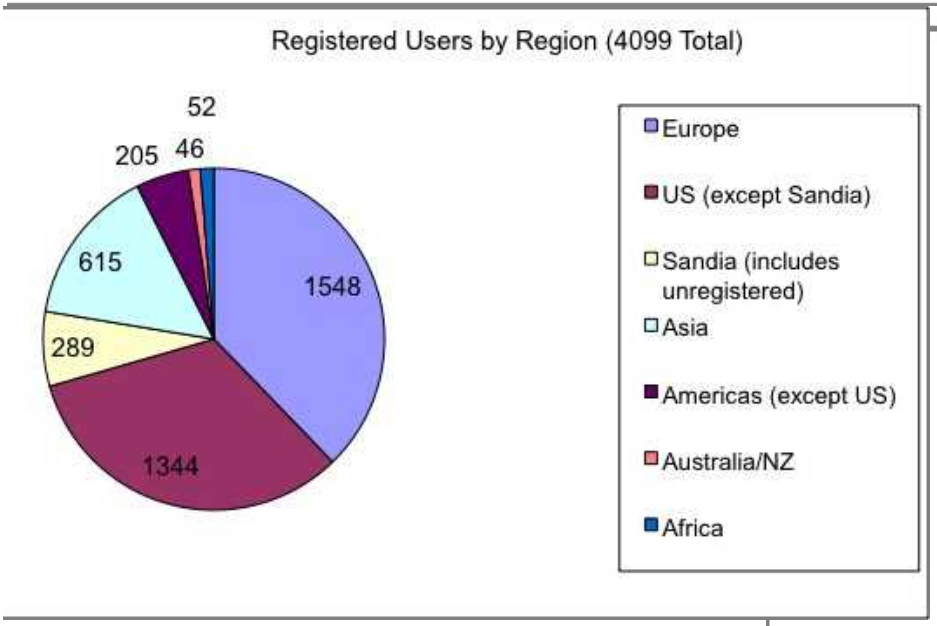
Stats: Trilinos Download Page 05/10/2010.

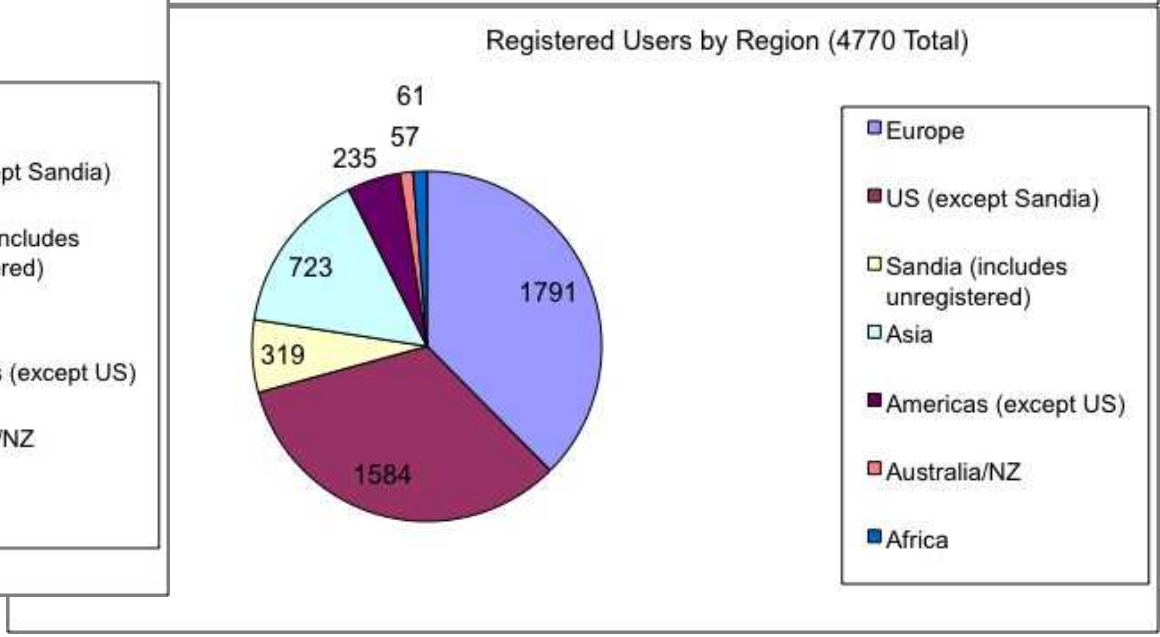
Sandia National Laboratories

Registered User Statistics









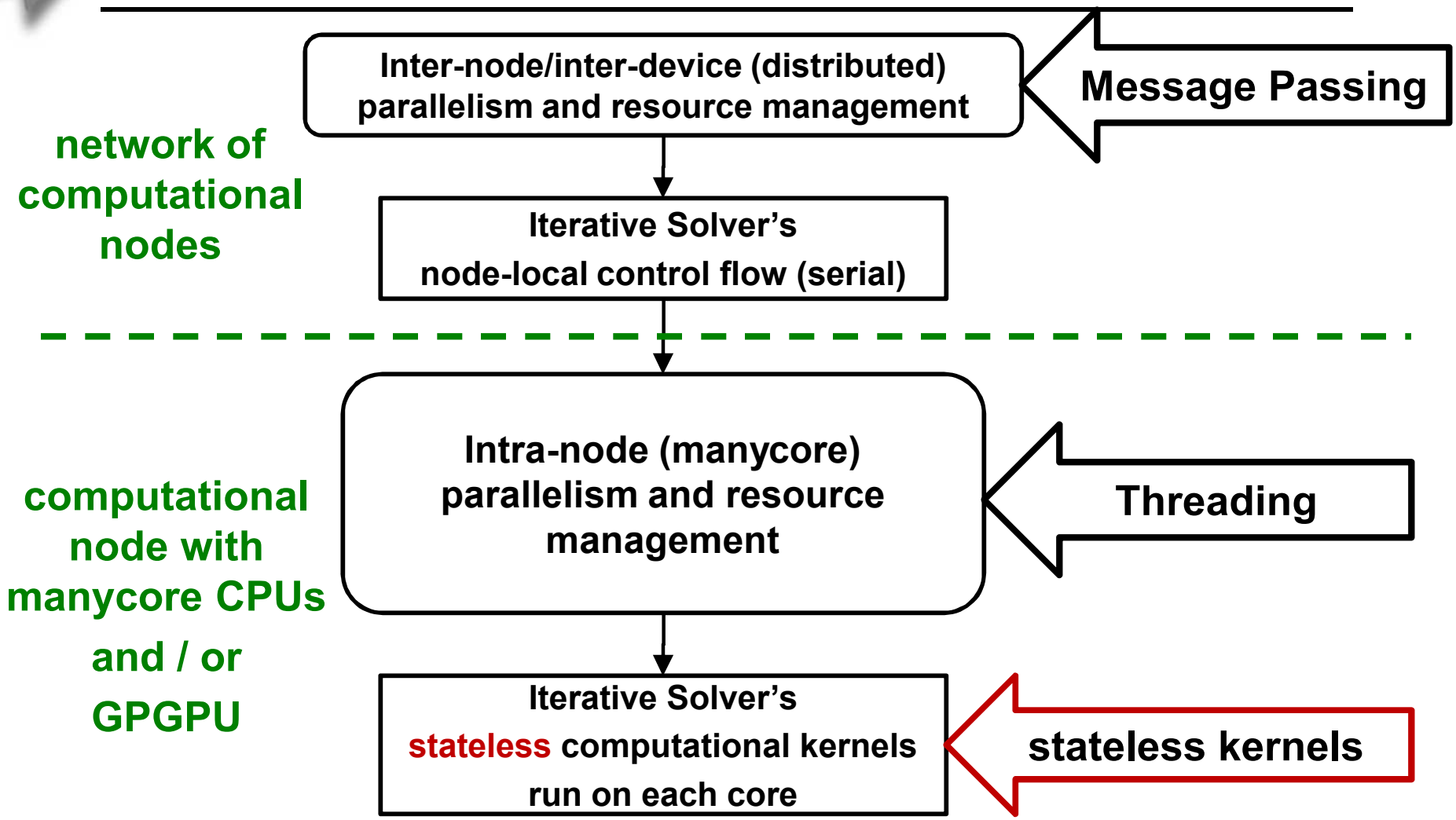
Stats: Trilinos Download Page 11/02/2009.

Stats: Trilinos Download Page 05/10/2010.

Sandia National Laboratories

Evolving Parallel Programming Model

Parallel Programming Model: Multi-level/Multi-device



Slide courtesy H. Carter Edwards

Mixed Precision Computations

C++ Templates and Mixed-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^{-6}	10^{-12}	10^{-24}	10^{-48}

Arbitrary precision solves using Tpetra and Belos linear solver package

FP Accuracy Analysis:
FloatShadowDouble Datatype

```
class FloatShadowDouble {

public:
  FloatShadowDouble( ) {
    f = 0.0f;
    d = 0.0;  }
  FloatShadowDouble( const FloatShadowDouble & fd) {
    f = fd.f;
    d = fd.d;  }
…
inline FloatShadowDouble operator+= (const FloatShadowDouble & fd ) {
    f += fd.f;
    d += fd.d;
    return *this;  }
…
inline std::ostream& operator<<(std::ostream& os, const FloatShadowDouble& fd) {
  os << fd.f << "f " << fd.d << "d";  return os;}
```

- Templates enable new analysis capabilities
- Example: Float with "shadow" double.

FloatShadowDouble

Sample usage:

```
#include “FloatShadowDouble.hpp”
Tpetra::Vector<FloatShadowDouble> x, y;
Tpetra::CrsMatrix<FloatShadowDouble> A;
A.apply(x, y);  // Single precision, but double results also computed, available
```

```
Initial Residual =          455.194f       455.194d
Iteration = 15   Residual = 5.07328f       5.07618d
Iteration = 30   Residual = 0.00147022f    0.00138466d
Iteration = 45   Residual = 5.14891e-06f   2.09624e-06d
Iteration = 60   Residual = 4.03386e-09f   7.91927e-10d
```

Trilinos/Kokkos Node API

Abstract Inter-node Communicator

- Not a novel idea.
  - Many distributed solvers/applications provide some wrapper for the communication, via diverse approaches.
- This technique is typically used for a few interfaces:
  - Trivial serial scenario, MPI and PVM
- Increasing combination of shared and distributed memory in legacy applications suggests more exotic possibilities may be ripe.
- Tpetra utilizes abstract base class `Comm` from Teuchos packages.

Generic Shared Memory Node

- Abstract inter-node comm provides DMP support.
- Need some way to portably handle SMP support.
- Goal: allow code, once written, to be run on any parallel node, regardless of architecture.
- Difficulty #1: Many different memory architectures
  - Node may have multiple, disjoint memory spaces.
  - Optimal performance may require special memory placement.
- Difficulty #2: Kernels must be tailored to architecture
  - Implementation of optimal kernel will vary between archs
  - No universal binary ➔ need for separate compilation paths

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 AxpyOp {
  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;     }
};
```

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

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

Hybrid CPU/GPU Computing

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
1 node 15 threads + Tesla	867	1,731
2 nodes 15 threads + Tesla	1,677	2,102

Data Placement

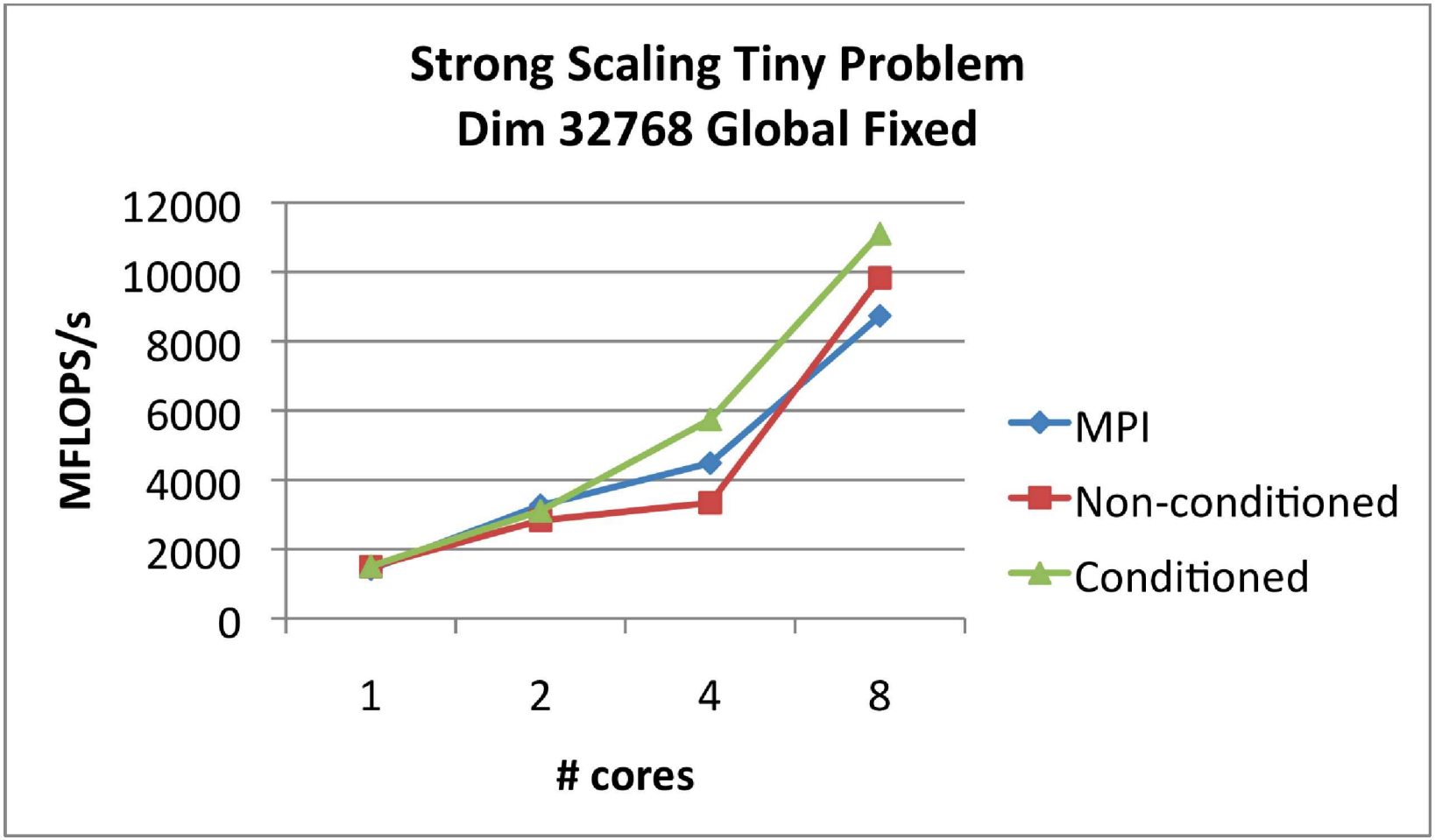
Data Placement on NUMA

- 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

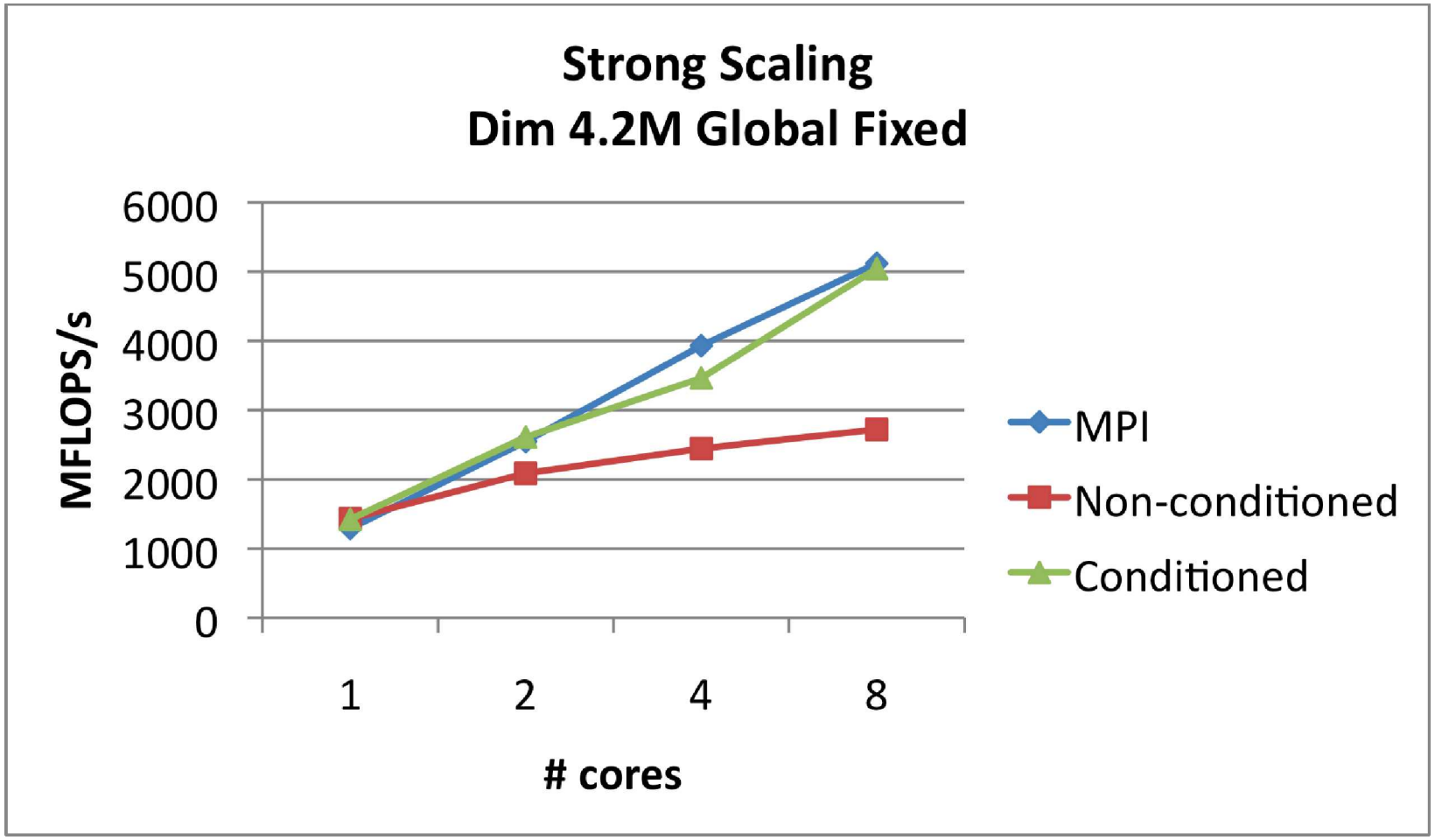
- MiniApp: HPCCG (Mantevo Project)
- Construct sparse linear system, solve with CG.
- Two modes:
  - Data placed by assembly, not migrated for NUMA
  - Data migrated using parallel access pattern of CG.
- Results on dual socket quad-core Nehalem system.

"Tiny Problem"



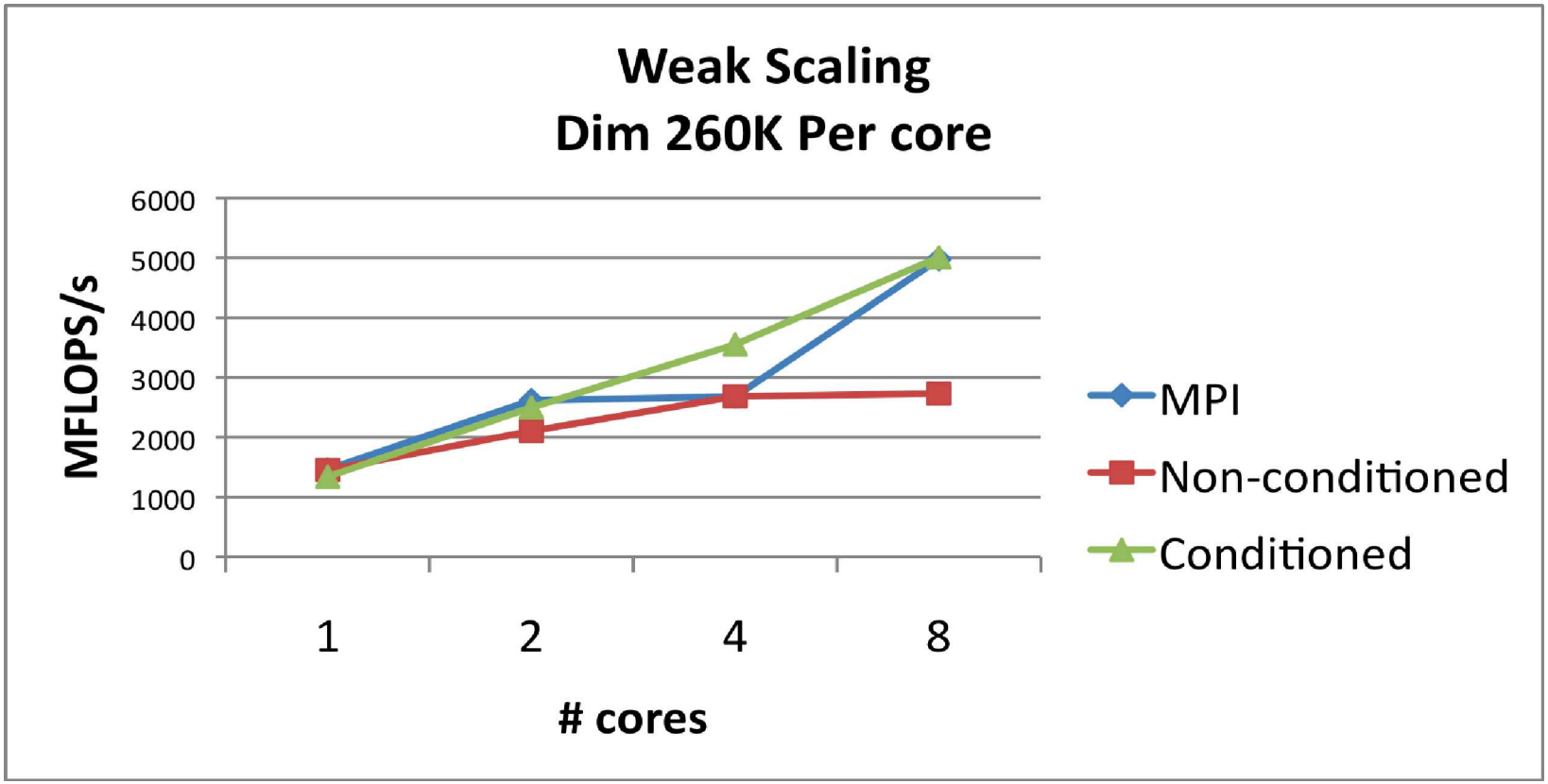
- Qualitatively similar behavior between three approaches.
- **Threaded (both versions) will outpace MPI.**

Strong Scaling Problem



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

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).

Resilient Algorithms

My Luxury in Life (wrt FT/Resilience)

The privilege to think of a computer as a *reliable, digital* machine.

Emerging Reality

"At 8 nm process technology, it will be harder to tell a 1 from a 0."

(W. Camp 2008, 2010)

Users' View of the System Now

- "All nodes up and running."
- Certainly nodes fail, but invisible to user.
- No need for me to be concerned.
- Someone else's problem.

Users' View of the System
Future

- Nodes in one of four states.
  1. Dead.
  2. Dying (perhaps producing faulty results).
  3. Reviving.
  4. Running properly
     a) Fully reliable or…
     b) Maybe still producing an occasional bad result.
- States 1-3:
  - Do I need to worry? I hope not.
- State 4b: Must I address this state? Hopefully not but…
- Here are some ideas.

Algorithm-Based Fault Tolerance

- Numerous approaches.
- Most common strategies:
  - Meta data:
    - Embed meta data into user-defined data structures.
    - Manage fault detection, recovery manually.
  - Algorithm results validation:
    - Use known algorithm properties.
    - Validate computed to known (e.g., residual check).

Two New? Ideas

1. Atomic Computation Regions.
2. Template-based redundancy.

Consider GMRES as an example of how soft errors affect correctness

- Basic Steps
  1) Compute Krylov subspace (preconditioned sparse matrix-vector multiplies)
  2) Compute orthonormal basis for Krylov subspace (matrix factorization)
  3) Compute vector yielding minimum residual in subspace (linear least squares)
  4) Map to next iterate in the full space
  5) Repeat until residual is sufficiently small

- More examples in Bronevetsky & Supinski, 2008

Why GMRES?

- Most Sandia Apps are implicit.
- Most popular linear solver is preconditioned GMRES.
- Only small subset of calculations need to be reliable.
  - GMRES is iterative, but also direct.

Every calculation matters

- Small PDE Problem: Dim 21K, Nz 923K.
- ILUT/GMRES
- Correct computation 35 Iters: 343M FLOPS
- Two examples of a single bad floating point op

Description	Iterations	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

One possible approach is transactional computation

- Database transactions: atomic
- Transactional memory: atomic memory operation
- Transactional computation:
  - Designated sensitive computation region (orthogonalization step in GMRES)
  - Guarantee accurate computation or notify user.

Needs to be coupled with SW-enabled guaranteed data regions

- User-designated reliable data region
- Extra protection to improve reliable data storage and transfer
- Examples
  - Original input data (needed for verification)
  - Linear solver: *A, x, b*
  - Orthogonal vectors for GMRES

Goal

- Algorithms well-conditioned wrt soft failure.
- Now:
  - Single soft error produces erroneous results.
- Goal:
  - Correct results always.
  - Cost increase proportional to number of soft errors.
- Note: These are just two approaches to ABFT.

RedundantCalcDouble Datatype

```
class RedundantCalcDouble {

public:
  RedundantCalcDouble( ) {
    d = 0.0;  }
  RedundantCalcDouble ( const RedundantCalcDouble & d_in) {
    d = d_in.d;  }
…
inline RedundantCalcDouble operator+= (const RedundantCalcDouble & d_in ) {
  d += d_in.d;
  double d_tmp = d + d_in.d;
  if (d_tmp!=d) throw std::exception;
    return *this;  }
…
inline std::ostream& operator<<(std::ostream& os, const RedundantCalcDouble & d_in) {
  os <<d_in.d";  return os;}
```

- Templates enable new resiliency capabilities
- Example: Redundant calculation

RedundantCalcDouble

Sample usage:

```
#include “RedundantCalcDouble.hpp”
Tpetra::Vector<RedundantCalcDouble> x, y;
Tpetra::CrsMatrix<RedundantCalcDouble> A;
try {
   A.apply(x, y);  // Redundant calculation
 }
catch (…) {
          …
}
```

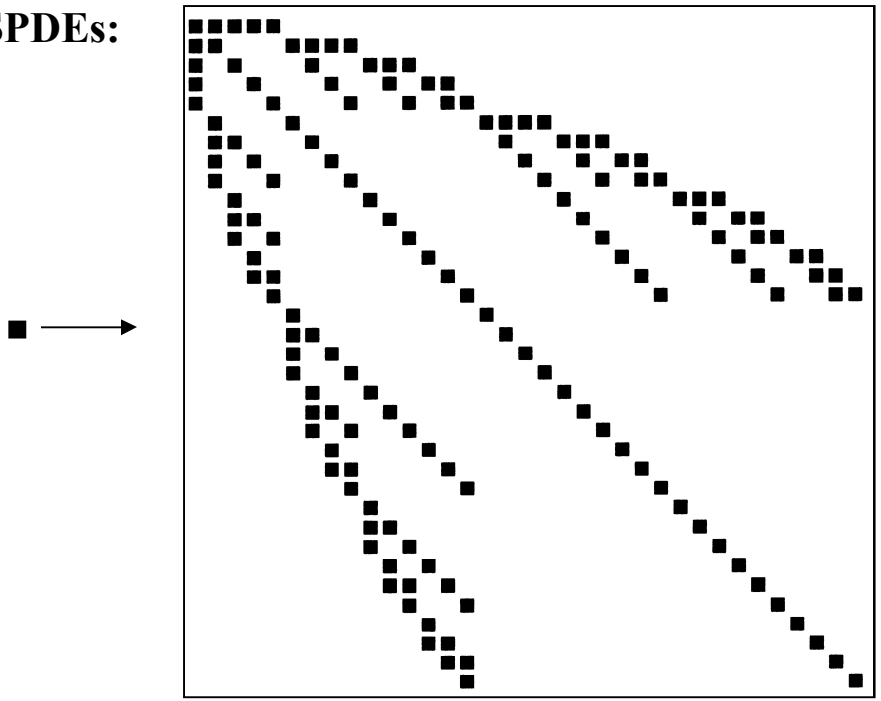
New Core Linear Algebra Needs

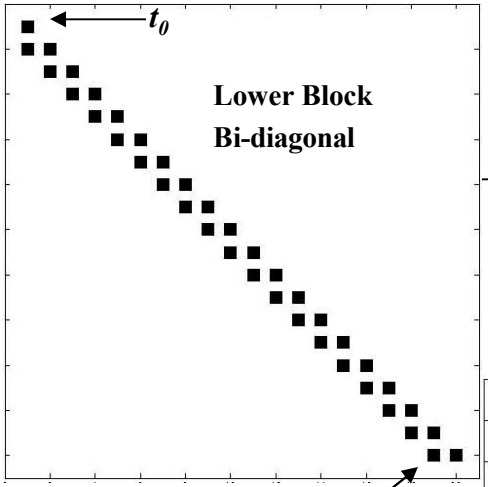
Sandia National Laboratories

Advanced Modeling and Simulation Capabilities: Stability, Uncertainty and Optimization

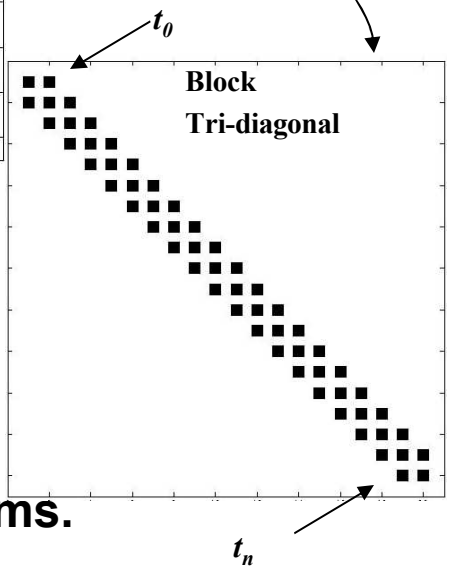
- **Promise: 10-1000 times increase in parallelism (or more).**

SPDEs:





Transient Optimization:



- **Pre-requisite: High-fidelity "forward" solve:**
  - **Computing families of solutions to similar problems.**
  - **Differences in results must be meaningful.**

■ - Size of a single forward problem

Sandia National Laboratories

Advanced Capabilities: Readiness and Importance

Modeling Area	Sufficient Fidelity?	Other concerns	Advanced capabilities priority
Seismic *S. Collis, C. Ober*	Yes.	None as big.	Top.
Shock & Multiphysics (Alegra) *A. Robinson, C. Ober*	Yes, but some concerns.	Constitutive models, material responses maturity.	Secondary now. Non-intrusive most attractive.
Multiphysics (Charon) *J. Shadid*	Reacting flow w/ simple transport, device w/ drift diffusion, …	Higher fidelity, more accurate multiphysics.	Emerging, not top.
Solid mechanics *K. Pierson*	Yes, but…	Better contact. Better timestepping. Failure modeling.	Not high for now.

Advanced Capabilities: Other issues

- Non-intrusive algorithms (e.g., Dakota):
  - Task level parallel:
    - A true peta/exa scale problem?
    - Needs a cluster of 1000 tera/peta scale nodes.
- Embedded/intrusive algorithms (e.g., Trilinos):
  - Cost of code refactoring:
    - Non-linear application becomes “subroutine”.
    - Disruptive, pervasive design changes.
- Forward problem fidelity:
  - Not uniformly available.
  - Smoothness issues.
  - Material responses.

Advanced Capabilities: Derived Requirements

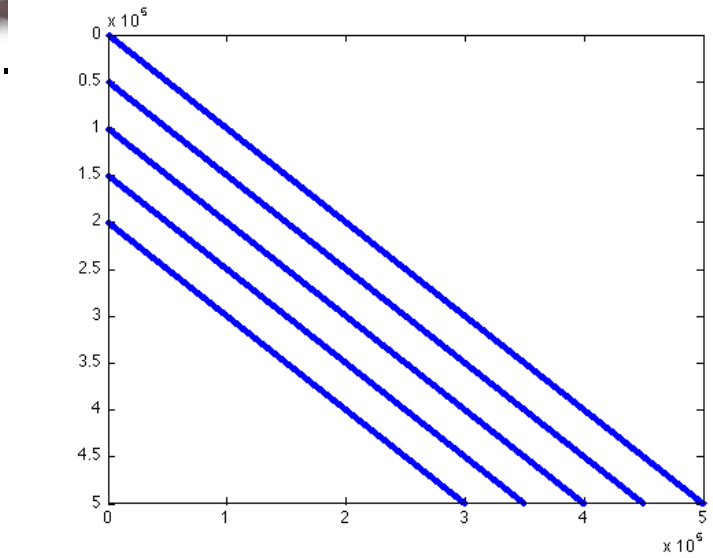
- Large-scale problem presents collections of related subproblems with forward problem sizes.
- Linear Solvers: $Ax = b \rightarrow AX = B,\ \ Ax^i = b^i,\ \ A^i x^i = b^i$
  - Krylov methods for multiple RHS, related systems.
- Preconditioners: $A^i = A_0 + \Delta A^i$
  - Preconditioners for related systems.
- Data structures/communication: $pattern(A^i) = pattern(A^j)$
  - Substantial graph data reuse.

Threaded Triangular Solve Details

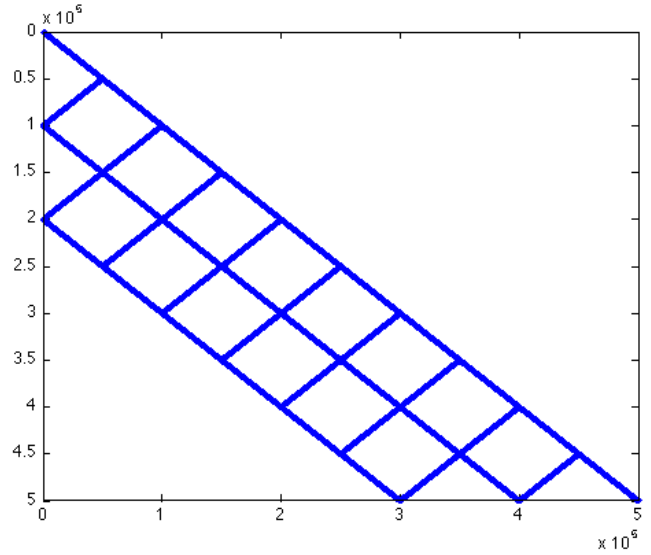
Motivation for Triangular Solve Work

- Triangular solver is important numerical kernel
- Essential role in preconditioning linear systems
  - Forward/back solve of Incomplete factorizations.
  - Smoother kernel for Gauss-Seidel.
- Difficult algorithm to parallelize.
- Focus of work: threaded triangular solve on each node/socket

Simple Prototype



"Good" data locality



"Bad" data locality

- Initial attempts at threaded triangular solve were awful.
- Simple prototype of level set threaded triangular solve
  - Assumes fixed number of rows per level
  - Assumes matrices preordered by level
  - Pthreads
- Allowed us to explore factors affecting performance

62

Factor 1: Type of Barrier

Algorithm 1 Passive Barrier.

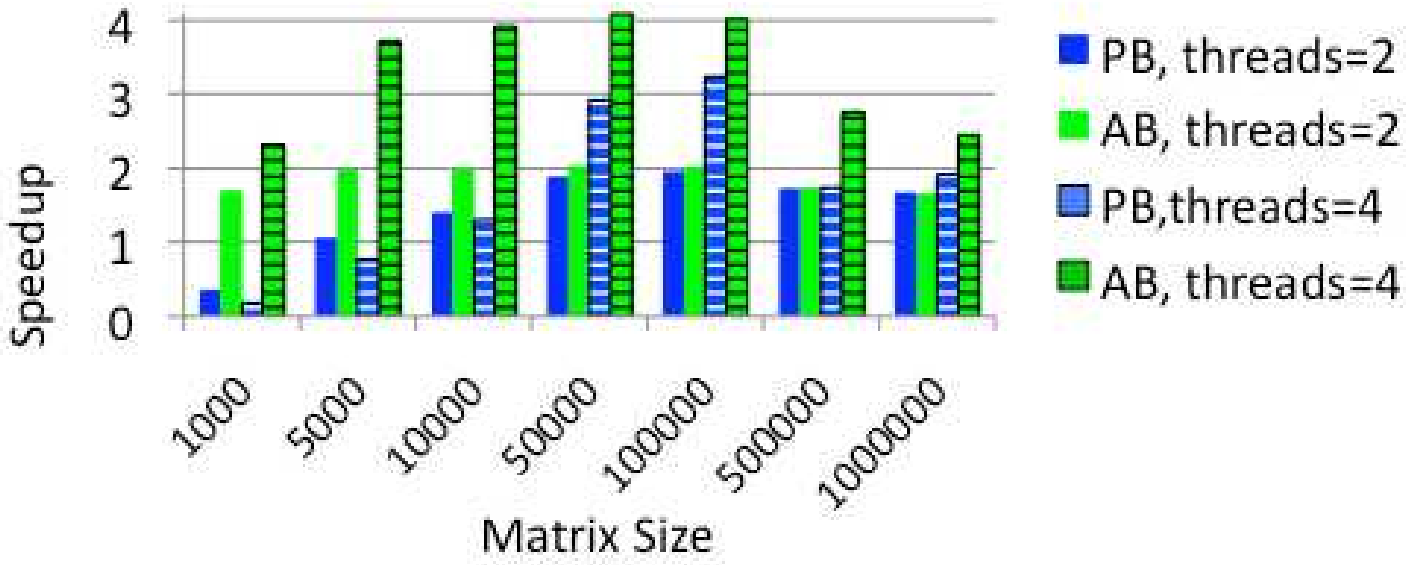
```
void passiveBarrier()
{
   pthread_mutex_lock(&mutex);
   numArrived++;
   if(numArrived < NUM_THREADS) {
     pthread_cond_wait(&barrCond,&mutex);
   }
   else {
     pthread_cond_broadcast(&barrCond);
     numArrived = 0;
   }
   pthread_mutex_unlock(&mutex);
}
```

Algorithm 2 Active Barrier.

```
void activeBarrier()
{
   pthread_spin_lock(&lock);
   actNumArrived++;
   if(actNumArrived==NUM_THREADS) {
     actLoopFlag = false;
   }
   pthread_spin_unlock(&lock);

   while(actLoopFlag) {}
}
```

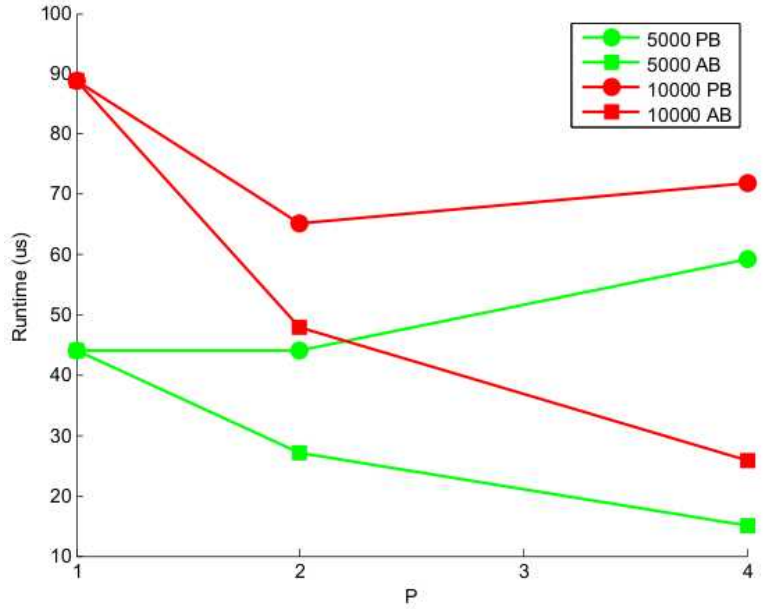
- Implemented two different barriers
  - "Passive" barrier
    - Mutexes and conditional wait statements
  - "Active" barrier
    - Spin locks and active polling

Factor 1: Type of Barrier



- Results for good data locality matrices
- Active/aggressive barriers essential for scalability

Factor 1: Type of Barrier

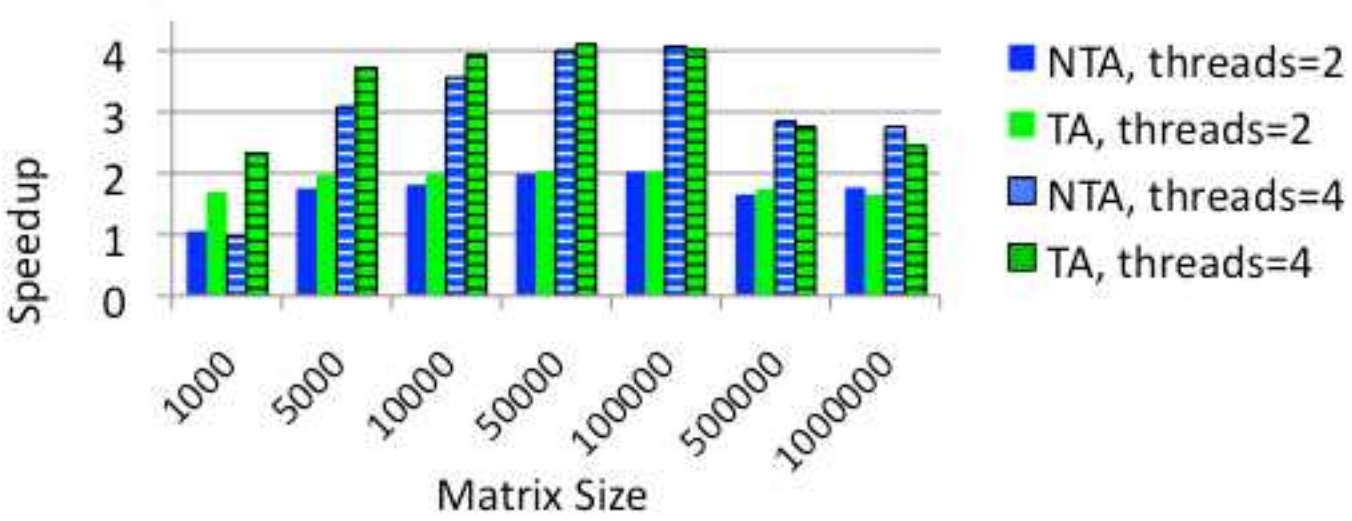


- Results for good data locality matrices
- Active/aggressive barriers essential for scalability

Factor 2: Thread Affinity

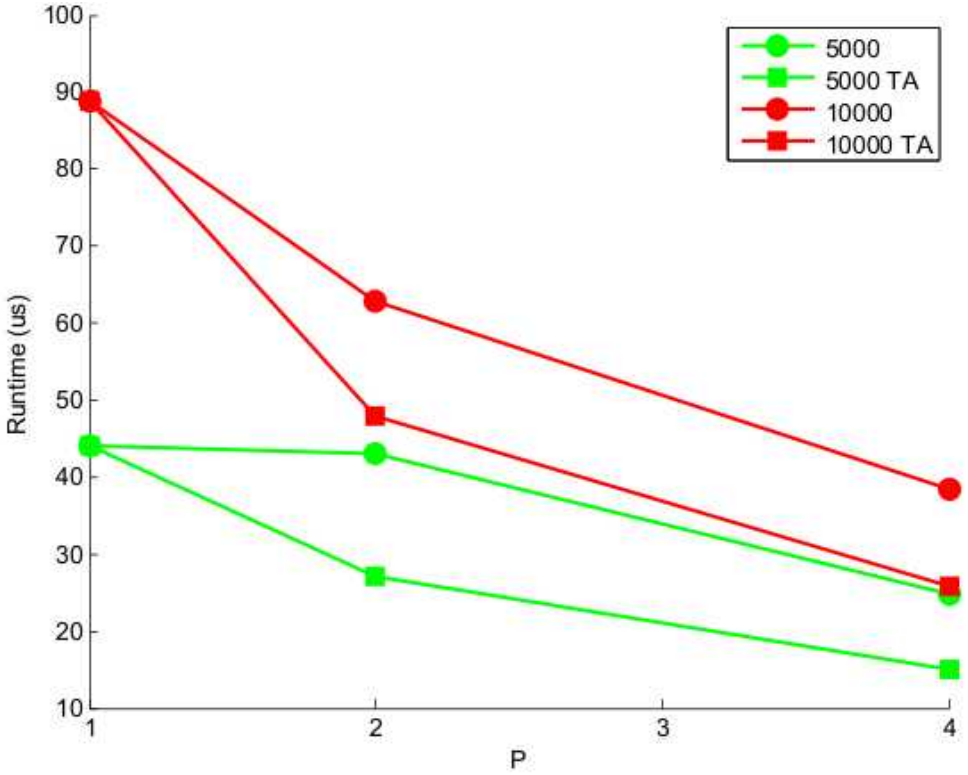
- Studied the importance of thread affinity
- Thread affinity allows threads to be pinned to cores
  - Less likely for threads to be switched (beneficial for cache utilization)
  - Ensures that threads are running on same socket

Factor 2: Thread Affinity



- Results for good data locality matrices, active barrier
- Thread affinity not as important as active barrier
  - But can be beneficial for some problem sizes

Factor 2: Thread Affinity



- Thread affinity not as important as active barrier
  - But can be beneficial for some problem sizes

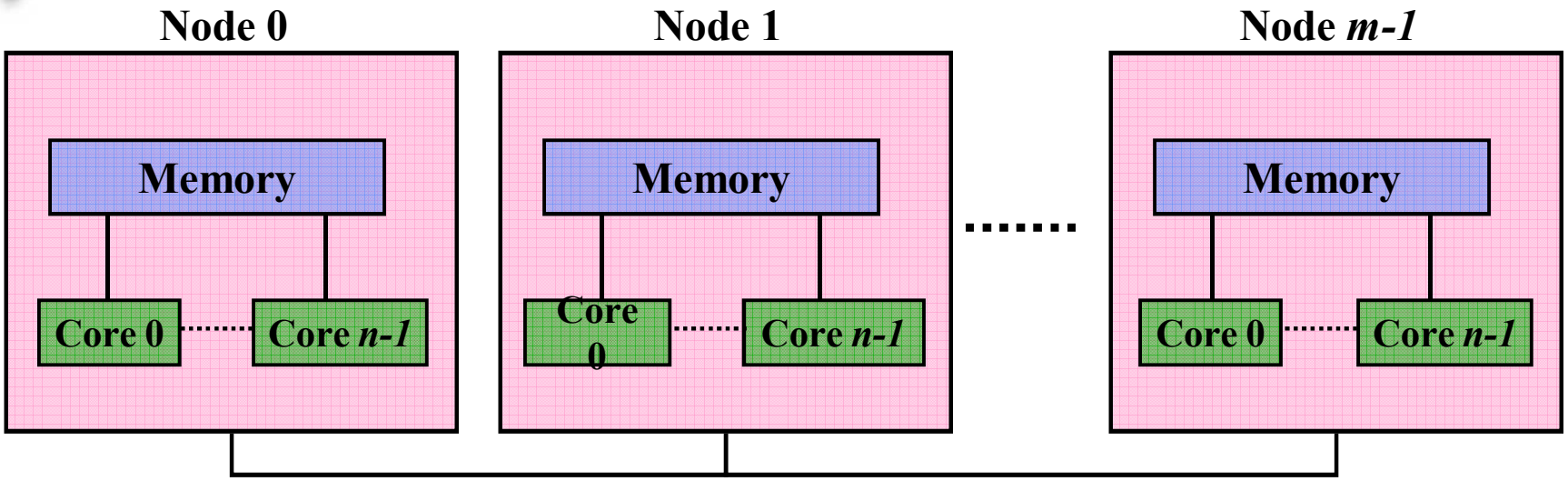
Threaded Solve Summary

- Threading can improve performance:
  - Good speedup.
  - Reduction in global iteration counts.
  - Modest size level sets needed. Multicoloring can increases sizes.
- Existing programming models lacking sufficient barriers:
  - Need two types of barriers:
    - Passive: exiting library to user control.
    - Active: syncing between phases of libraries execution.
  - Not supported in current PMs (OpenMP, TBB, CUDA)
    - Pthreads (not a model) can support it.
  - In Discussion with PM developers (e.g., IWOMP 2010).
- Thread affinity:
  - Still an open issue.
  - Other work shows affinity is essential.
- Final Note: Per core memory requirement should go down!

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

Hybrid MPI-only, MPI+threading details

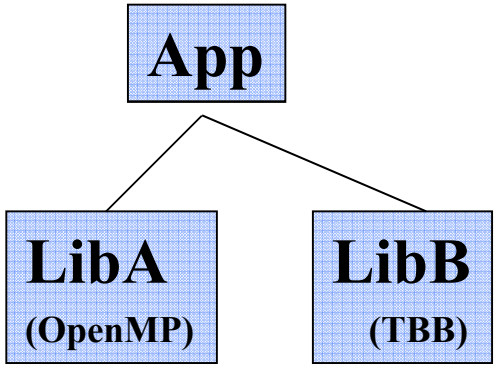
Parallel Machine Block Diagram



- Parallel machine with $p = m * n$ processors:
  - m = number of nodes.
  - n = number of shared memory processors per node.
- Two ways to program:
  - Way 1: p MPI processes.
  - Way 2: m MPI processes with n threads per MPI process.
- New third way:
  - "Way 1" in some parts of the execution (the app).
  - "Way 2" in others (the solver).

Threading under MPI

- Default approach: Successful in many applications.
- Concerns:
  - Opaqueness of work/data pair assignment.
    - Lack of granularity control.
  - Collisions: Multiple thread models.
    - Performance issue, not correctness.



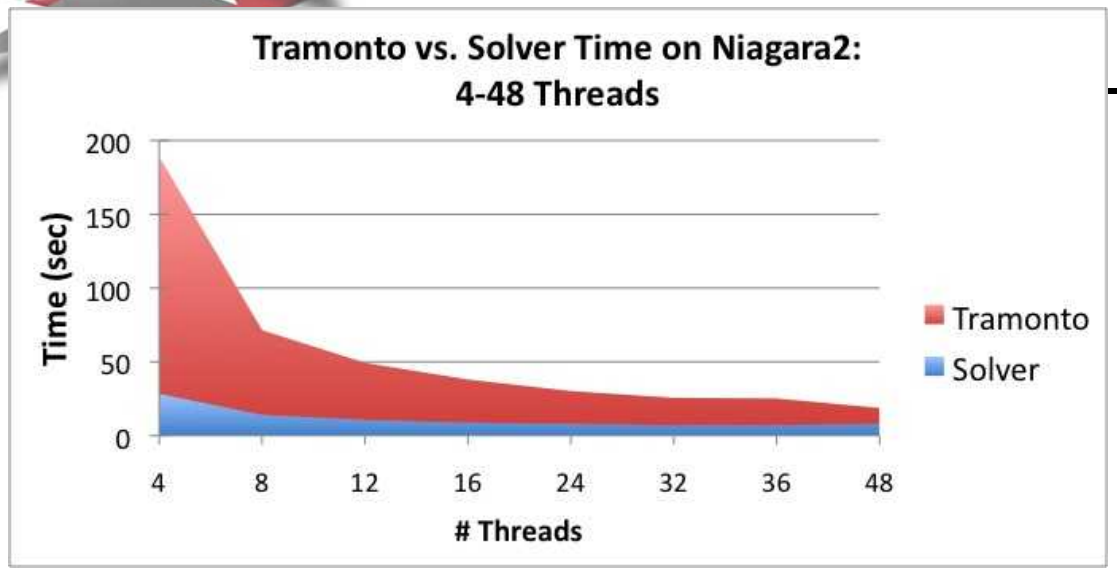
- Bright spot: Intel Thread Building Blocks (TBB).
  - Iterator (C++ language feature) model.
  - Opaque or transparent: User choice.

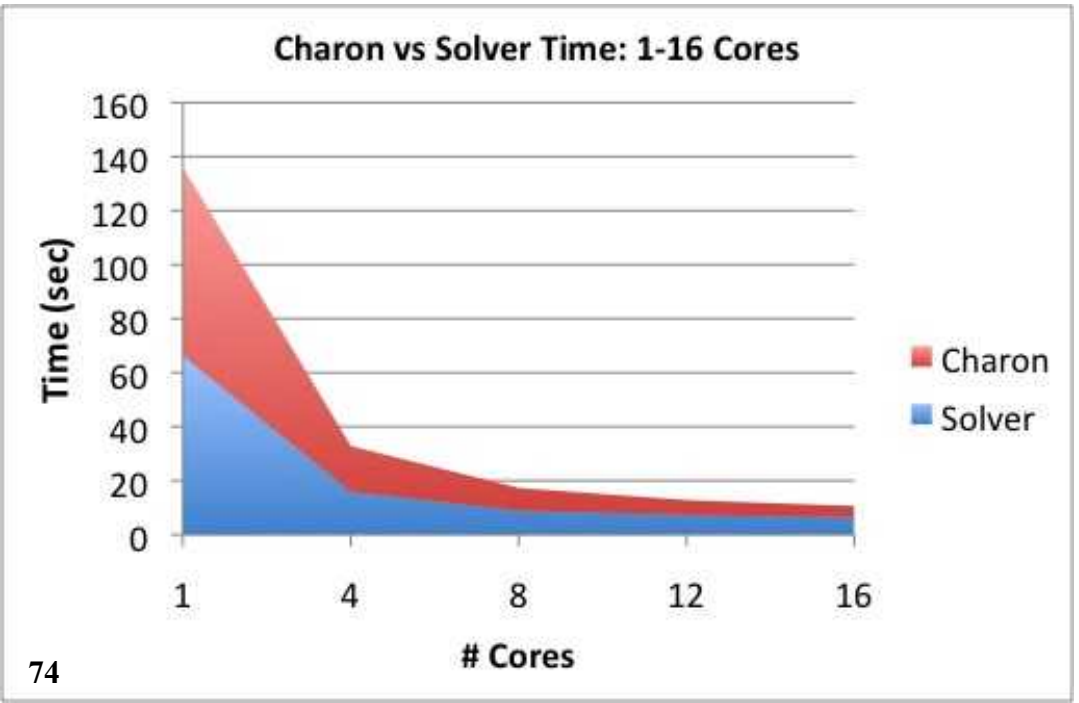
MPI Under MPI

- Scalable multicore:
  - Two different MPI architectures.
  - Machines within a machine.
- Exploited in single-level MPI:
  - Short-circuited messages.
  - Reduce network B/W.
  - Missing some potential.
- Nested algorithms.
- Already possible.
- Real attraction: No new node programming model.
- Can even implement shared memory algorithms (with some enhancements to MPI).

"Ping-pong" test	Latency (microsec)	Bandwidth (MB/sec)
Intra-node machine	0.71	1082
Inter-node machine	47.5	114

Multicore Scaling: App vs. Solver





Application:

- Scales well (sometimes superlinear)
- MPI-only sufficient.

Solver:

- Scales more poorly.
- Memory system-limited.
- MPI+threads can help.

* **Charon Results:**
Lin & Shadid TLCC Report

MPI-Only + MPI/Threading: *Ax=b*