SAND2011-4710C

Preparing for Tomorrow's Systems: Manycore, Resilience, Patterns and Transition

Michael A. Heroux
Scalable Algorithms Department
Sandia National Laboratories

Collaborators:
SNL Staff: [B.|R.] Barrett, E. Boman, R. Brightwell, H.C. Edwards, A. Williams
SNL Postdocs: M. Hoemmen, S. Rajamanickam
MIT Lincoln Lab: M. Wolf
ORNL staff: Chris Baker

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.

Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
4. All future programmers will need to write parallel code.
5. DRY is not possible across CPUs and GPUs.
6. CUDA and OpenCL will be footnotes in computing history.
7. Extended precision is too expensive to be useful.
8. Resilience will be built into algorithms.
9. A solution with error bars complements architecture trends.
10. Global SIMT is sufficient parallelism for scientific computing.

Trilinos Background & Motivation

Trilinos Contributors

Current Contributors

Chris Baker
Ross Bartlett
Pavel Bochev
Erik Boman
Lee Buermann
Todd Coffey
Eric Cyr
David Day
Karen Devine
Clark Dohrmann
David Gay
Glen Hansen
David Hensinger
Mike Heroux
Mark Hoemmen
Russell Hooper
Jonathan Hu
Sarah Knepper
Patrick Knupp
Joe Kotulski
Jason Kraftcheck
Rich Lehoucq
Nicole Lemaster
Kevin Long
Karla Morris
Chris Newman
Kurtis Nusbaum
Ron Oldfield
Mike Parks
Roger Pawlowski
Brent Perschbacher
Kara Peterson
Eric Phipps
Siva Rajamanickam
Denis Ridzal
Lee Ann Riesen
Damian Rouson
Andrew Salinger
Nico Schlömer
Chris Siefert
Greg Sjaardema
Bill Spotz
Heidi Thornquist
Ray Tuminaro
Jim Willenbring
Alan Williams
Michael Wolf

Past Contributors

Paul Boggs
Jason Cross
Michael Gee
Esteban Guillen
Bob Heaphy
Ulrich Hetmaniuk
Robert Hoekstra
Vicki Howle
Kris Kampshoff
Tammy Kolda
Joe Outzen
Mike Phenow
Paul Sexton
Ken Stanley
Marzio Sala
Cedric Chevalier

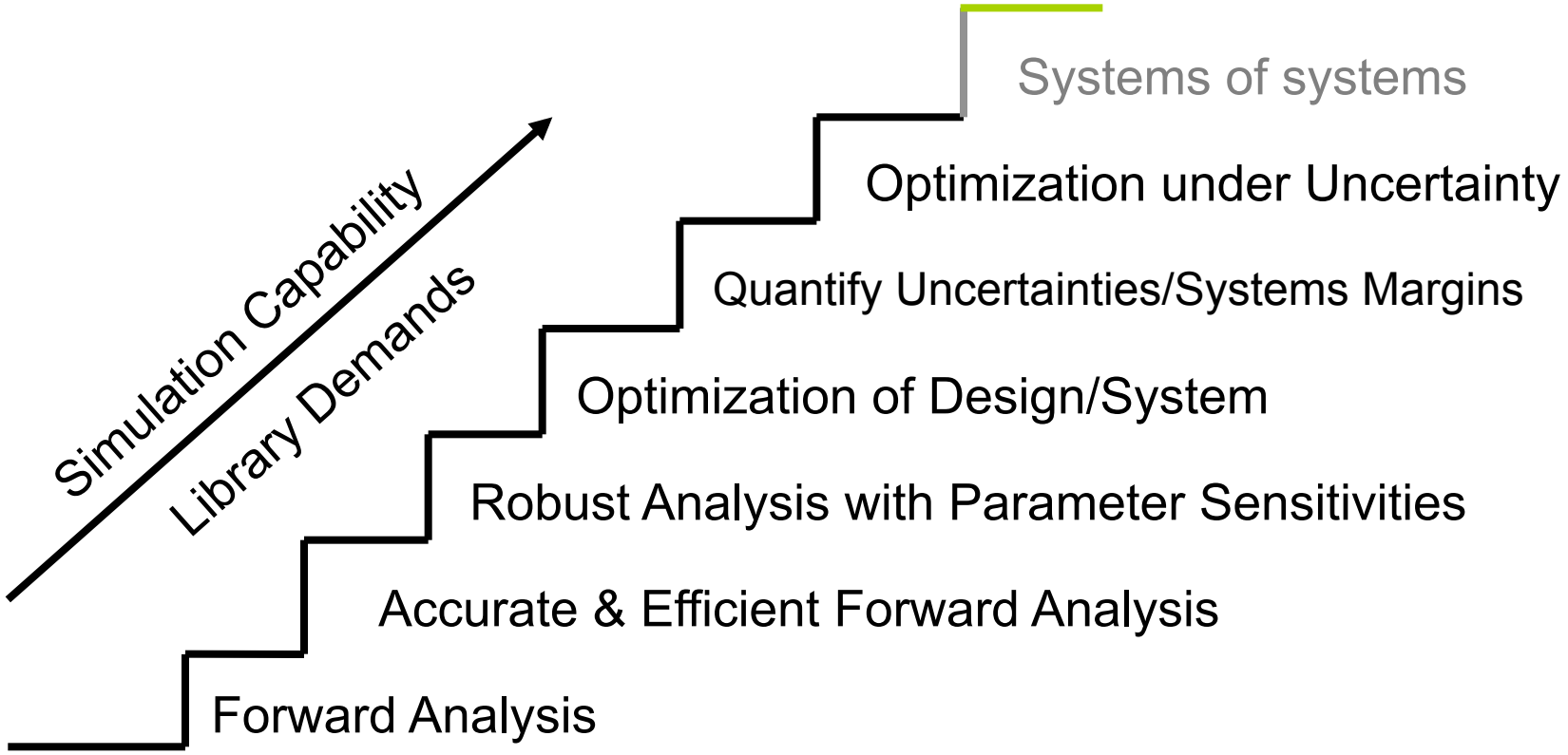
Evolving Trilinos Solution

- Trilinos[1] is an evolving framework to address these challenges:
  - Fundamental atomic unit is a *package*.
  - Includes core set of vector, graph and matrix classes (Epetra/Tpetra packages).
  - Provides a common abstract solver API (Thyra package).
  - Provides a ready-made package infrastructure:
    - Source code management (git).
    - Build tools (Cmake).
    - Automated regression testing.
    - Communication tools (mail lists, trac).
  - Specifies requirements and suggested practices for package SQA.
- In general allows us to categorize efforts:
  - Efforts best done at the Trilinos level (useful to most or all packages).
  - Efforts best done at a package level (peculiar or important to a package).
  - **Allows package developers to focus only on things that are unique to their package.**

1. Trilinos loose translation: “A string of pearls”

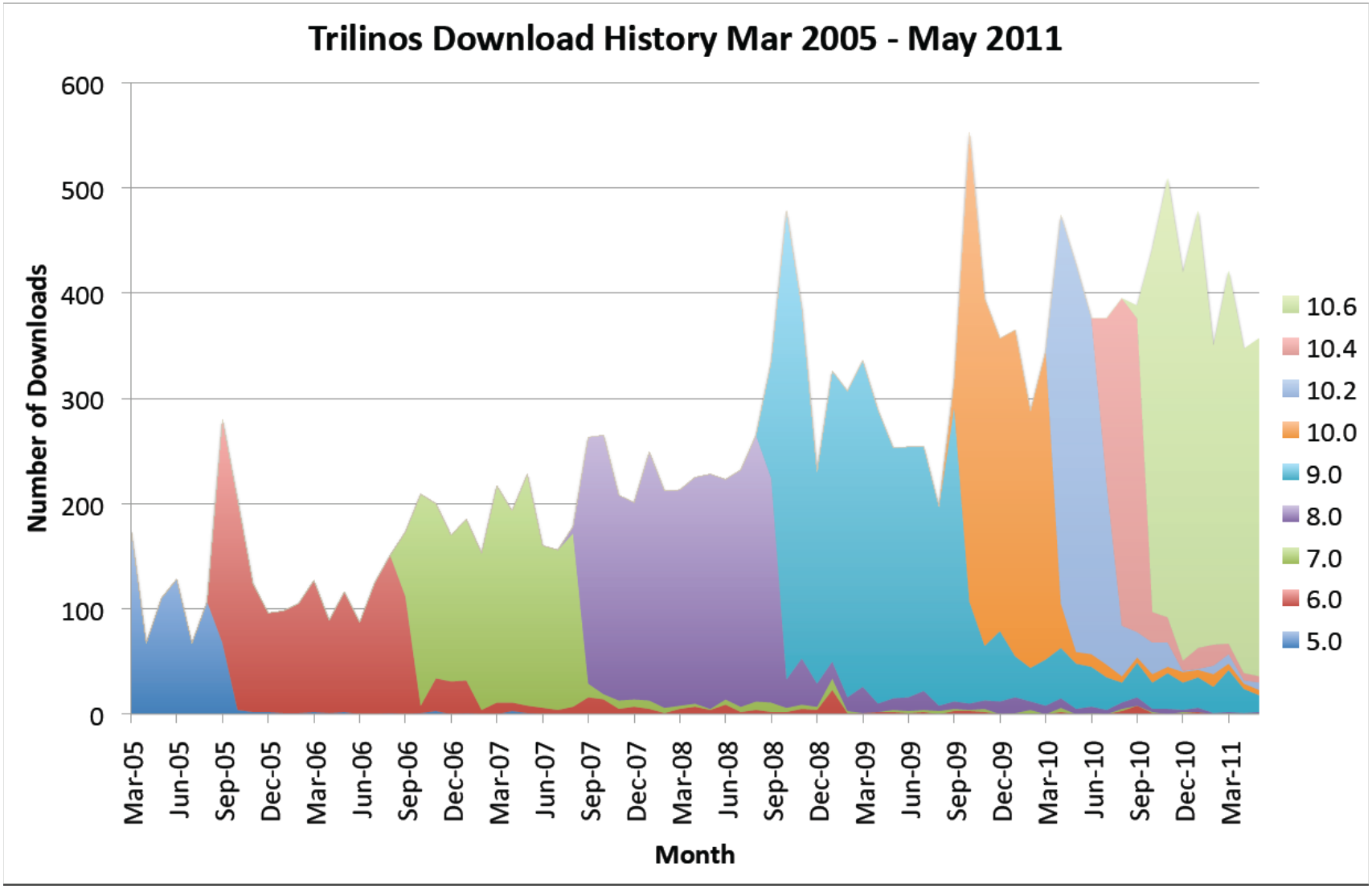
Sandia National Laboratories

Transforming Computational Analysis To Support High Consequence Decisions

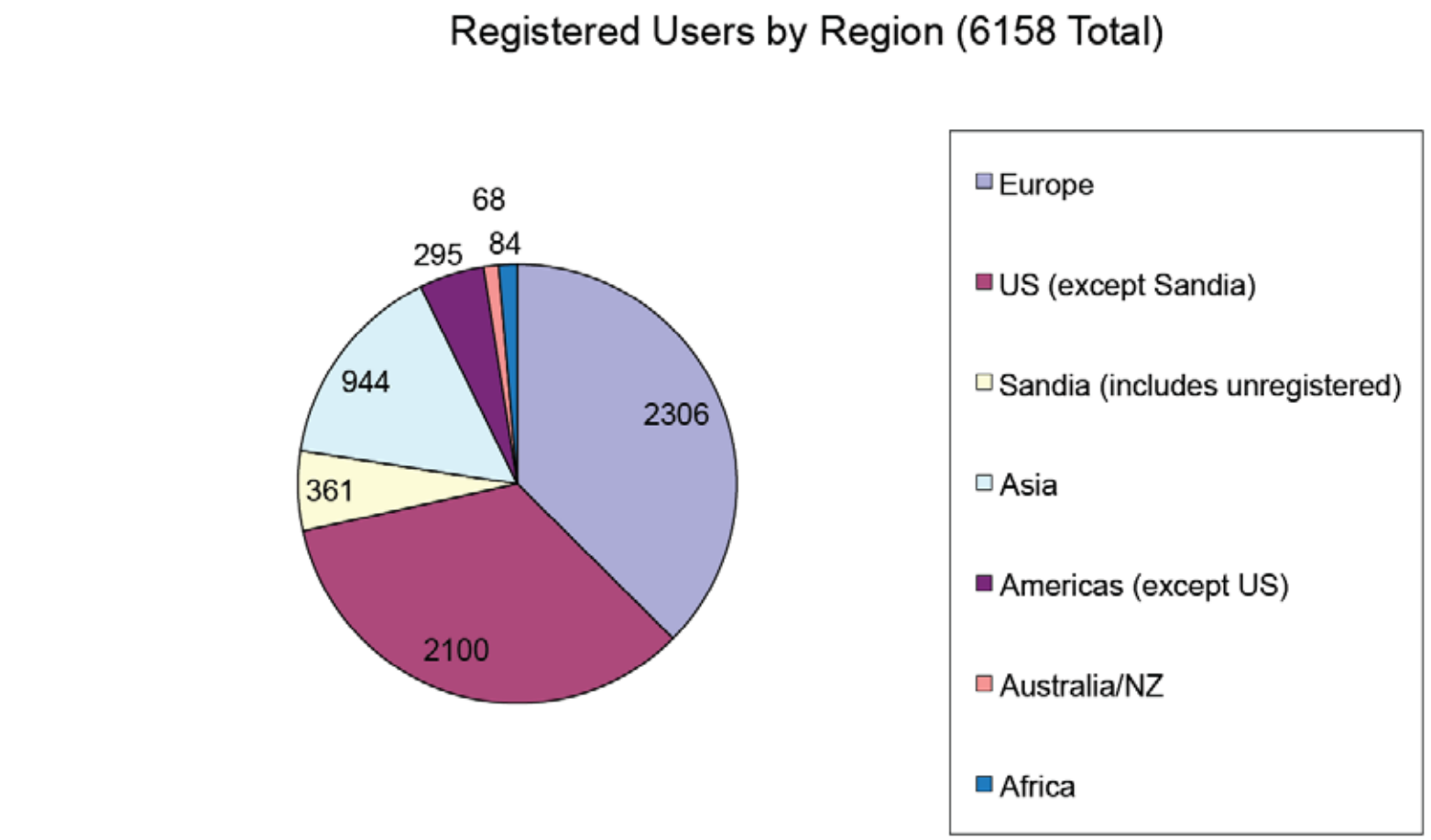


Each stage requires *greater performance* and *error control* of prior stages:
Always will need: more accurate and scalable methods.
more sophisticated tools.

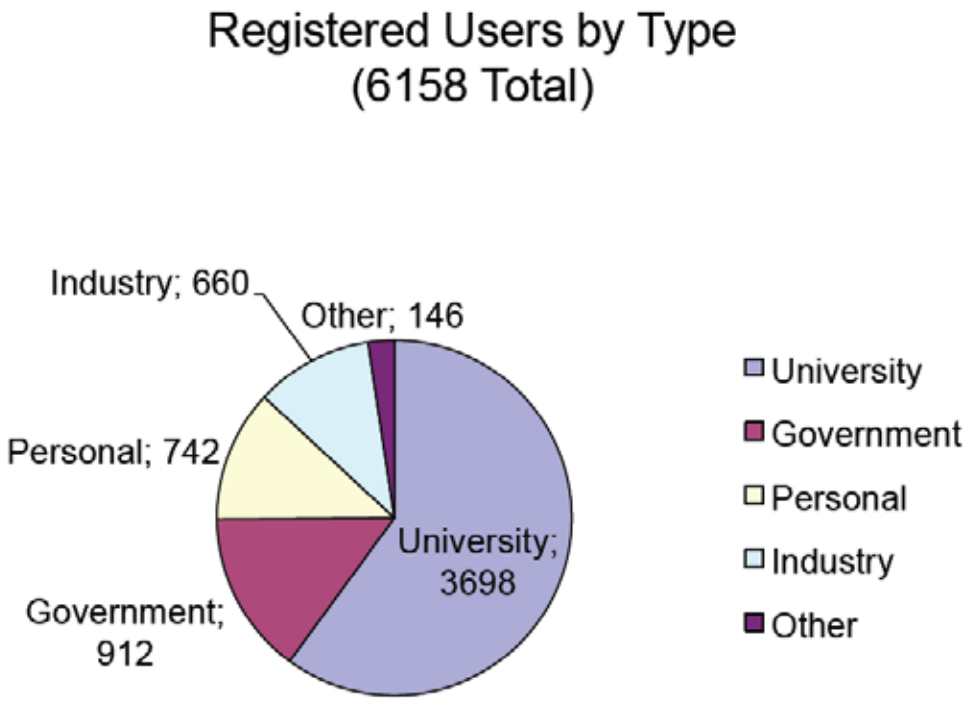
Trilinos Download History: 19525 Total



Registered User by Region

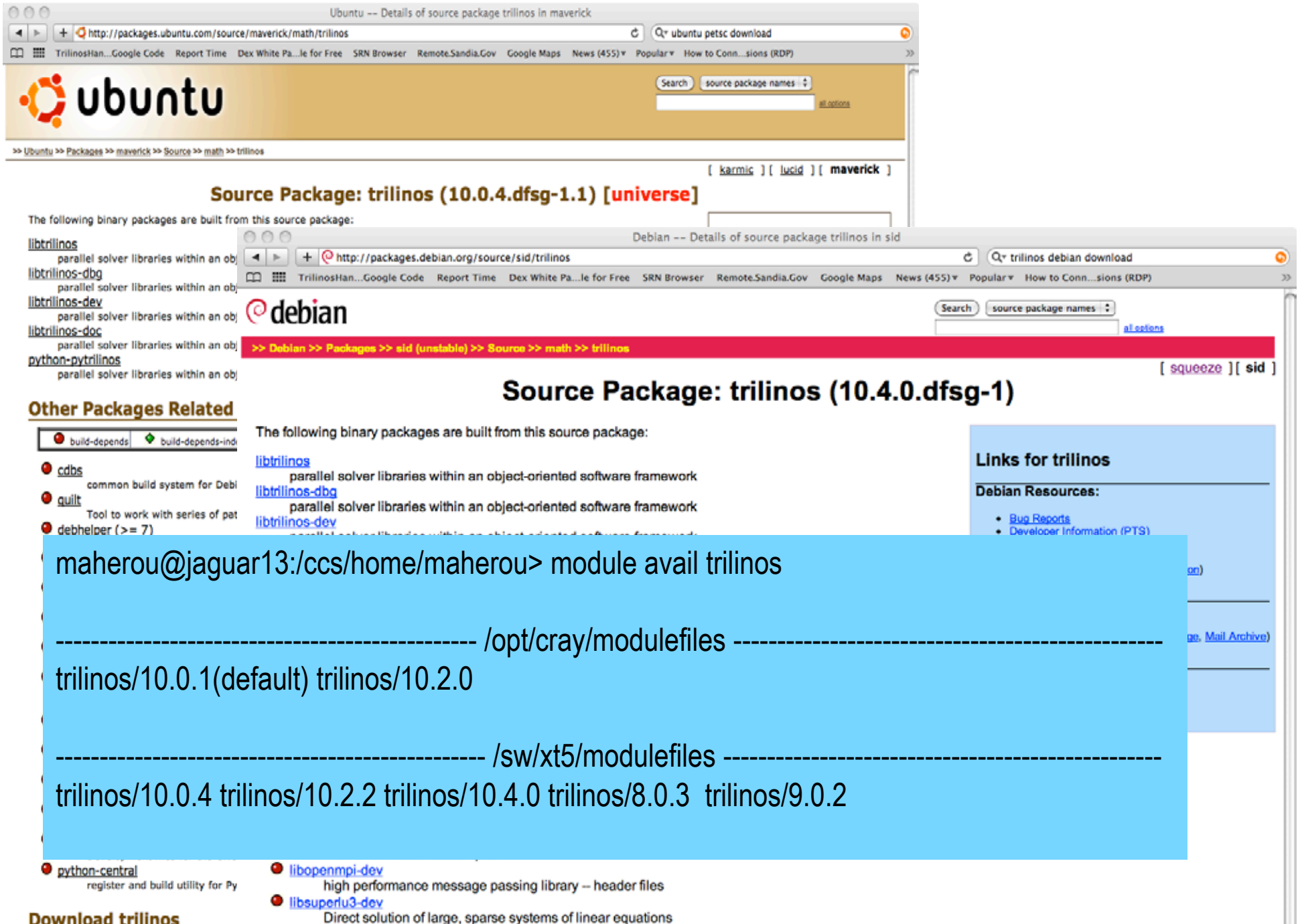


Registered Users by Type



Sandia National Laboratories

Ubuntu/Debian: Other sources




```
maherou@jaguar13:/ccs/home/maherou> module avail trilinos

---------------------------------------------- /opt/cray/modulefiles ----------------------------------------------
trilinos/10.0.1(default) trilinos/10.2.0

------------------------------------------------ /sw/xt5/modulefiles ------------------------------------------------
trilinos/10.0.4 trilinos/10.2.2 trilinos/10.4.0 trilinos/8.0.3  trilinos/9.0.2
```

Capability Leaders: Layer of Proactive Leadership

- Areas:
  - Framework, Tools & Interfaces (J. Willenbring).
  - Software Engineering Technologies and Integration (R. Bartlett).
  - Discretizations (P. Bochev).
  - Geometry, Meshing & Load Balancing (K. Devine).
  - Scalable Linear Algebra (M. Heroux).
  - Linear & Eigen Solvers (J. Hu).
  - Nonlinear, Transient & Optimization Solvers (A. Salinger).
  - Scalable I/O: (R. Oldfield)
- Each leader provides strategic direction across all Trilinos packages within area.

Trilinos Package Summary

	Objective	Package(s)
Discretizations	Meshing & Discretizations	STKMesh, Intrepid, Pamgen, Sundance, ITAPS, Mesquite
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Services	Linear algebra objects	Epetra, Jpetra, Tpetra, Kokkos
	Interfaces	Thyra, Stratimikos, RTOp, FEI, Shards
	Load Balancing	Zoltan, Isorropia
	“Skins”	PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika
	C++ utilities, I/O, thread API	Teuchos, EpetraExt, **Kokkos**, Triutils, ThreadPool, Phalanx
Solvers	Iterative linear solvers	AztecOO, Belos, Komplex
	Direct sparse linear solvers	Amesos, Amesos2
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi, Rbgen
	ILU-type preconditioners	AztecOO, IFPACK, Ifpack2
	Multilevel preconditioners	ML, CLAPS
	Block preconditioners	Meros, Teko
	Nonlinear system solvers	NOX, LOCA
	Optimization (SAND)	MOOCHO, Aristos, TriKota, Globipack, Optipack
	Stochastic PDEs	Stokhos

Observations and Strategies for Parallel Software Design

Three Design Points

- Terascale Laptop: Uninode-Manycore
- Petascale Deskside: Multinode-Manycore
- Exascale Center: Manynode-Manycore

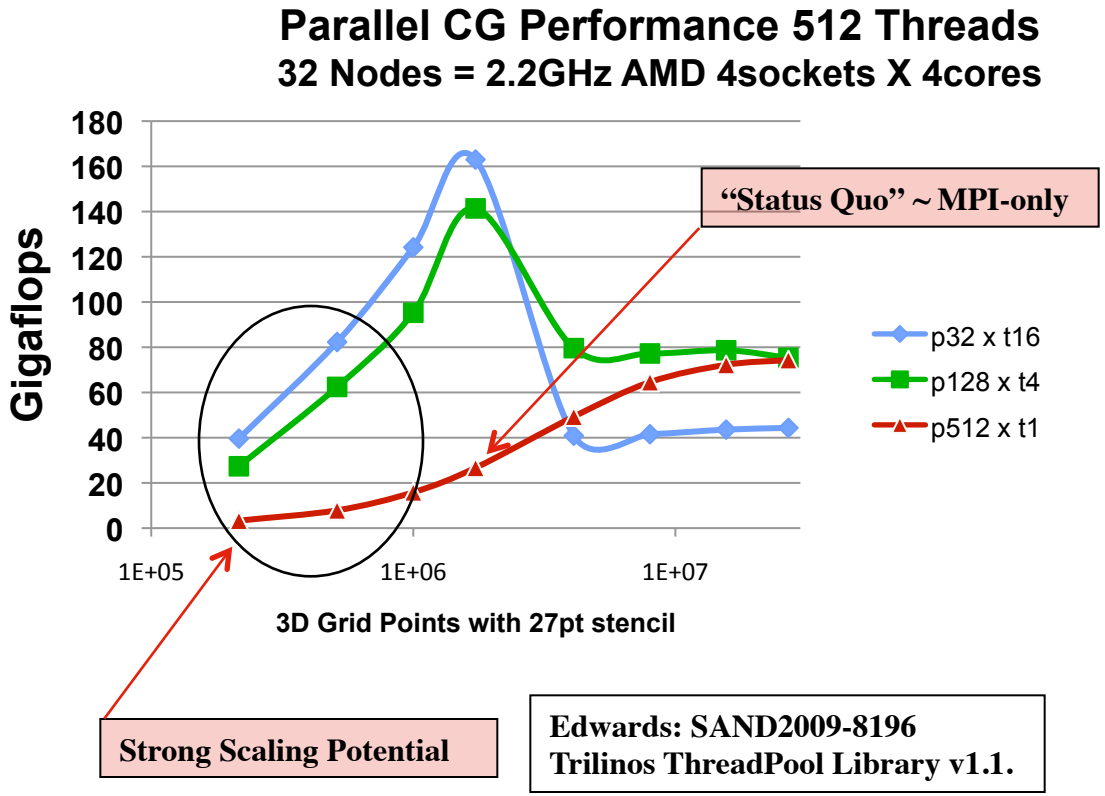
Sandia National Laboratories

Basic 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.
  - Resilience.



Sandia National Laboratories

Observations

- MPI-Only is not sufficient, except … much of the time.
- Near-to-medium term:
  - MPI+[OMP|TBB|Pthreads|CUDA|OCL|MPI]
  - Long term, too?
- Concern:
  - Best hybrid performance: 1 MPI rank per UMA core set.
  - UMA core set size growing slowly ➔ Lots of MPI tasks.
- Long- term:
  - Something hierarchical, global in scope.
- Conjecture:
  - Data-intensive apps need non-SPDM model.
  - Will develop new programming model/env.
  - Rest of apps will adopt over time.
  - Time span: 10-20 years.

What Can we Do Right Now?

- Study why MPI was successful.
- Study new parallel landscape.
- Try to cultivate an approach similar to MPI (and others).

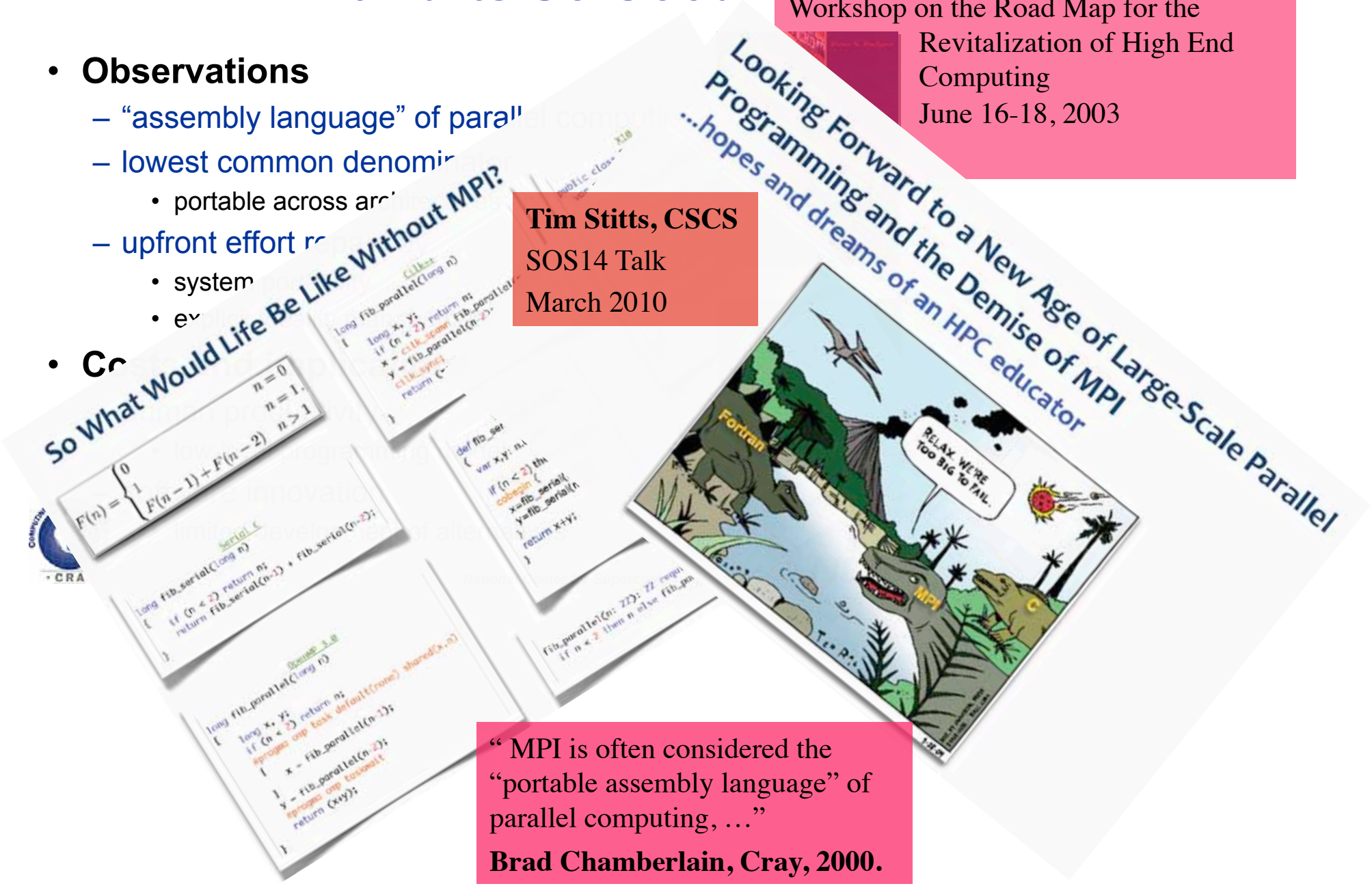
MPI Impresssions

MPI: It Hurts So Good

- **Observations**
  - "assembly language" of parallel comput
  - lowest common denomi
    - portable across ar
  - upfront effort r
    - system
    - ex
- **Co**

Dan Reed, Microsoft
Workshop on the Road Map for the Revitalization of High End Computing
June 16-18, 2003

Tim Stitts, CSCS
SOS14 Talk
March 2010



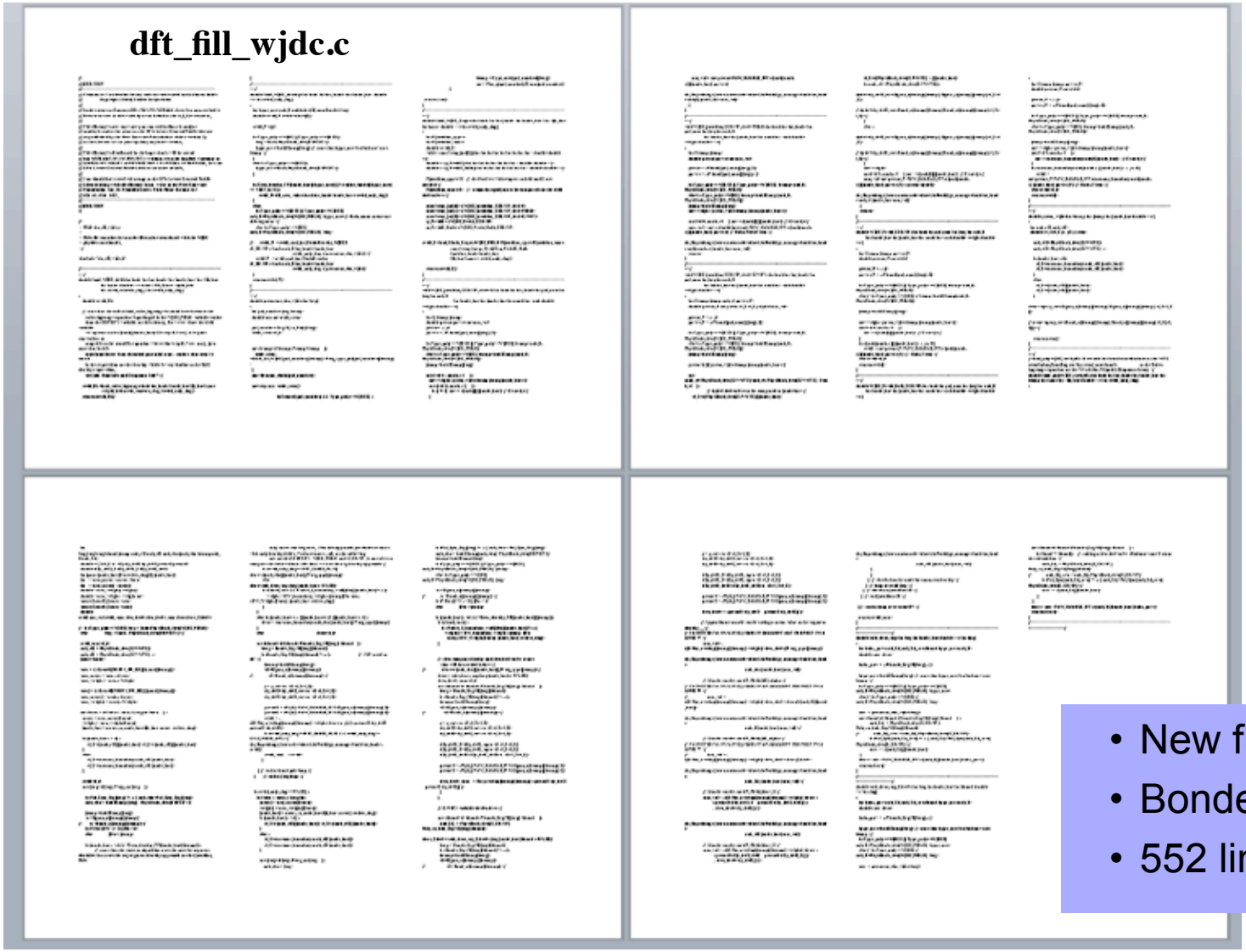
" MPI is often considered the "portable assembly language" of parallel computing, …"

Brad Chamberlain, Cray, 2000.

3D Stencil in NAS MG

Brad Chamberlain, Cray, PPOPP'06, http://chapel.cray.com/publications/ppopp06-slides.pdf

MPI Reality

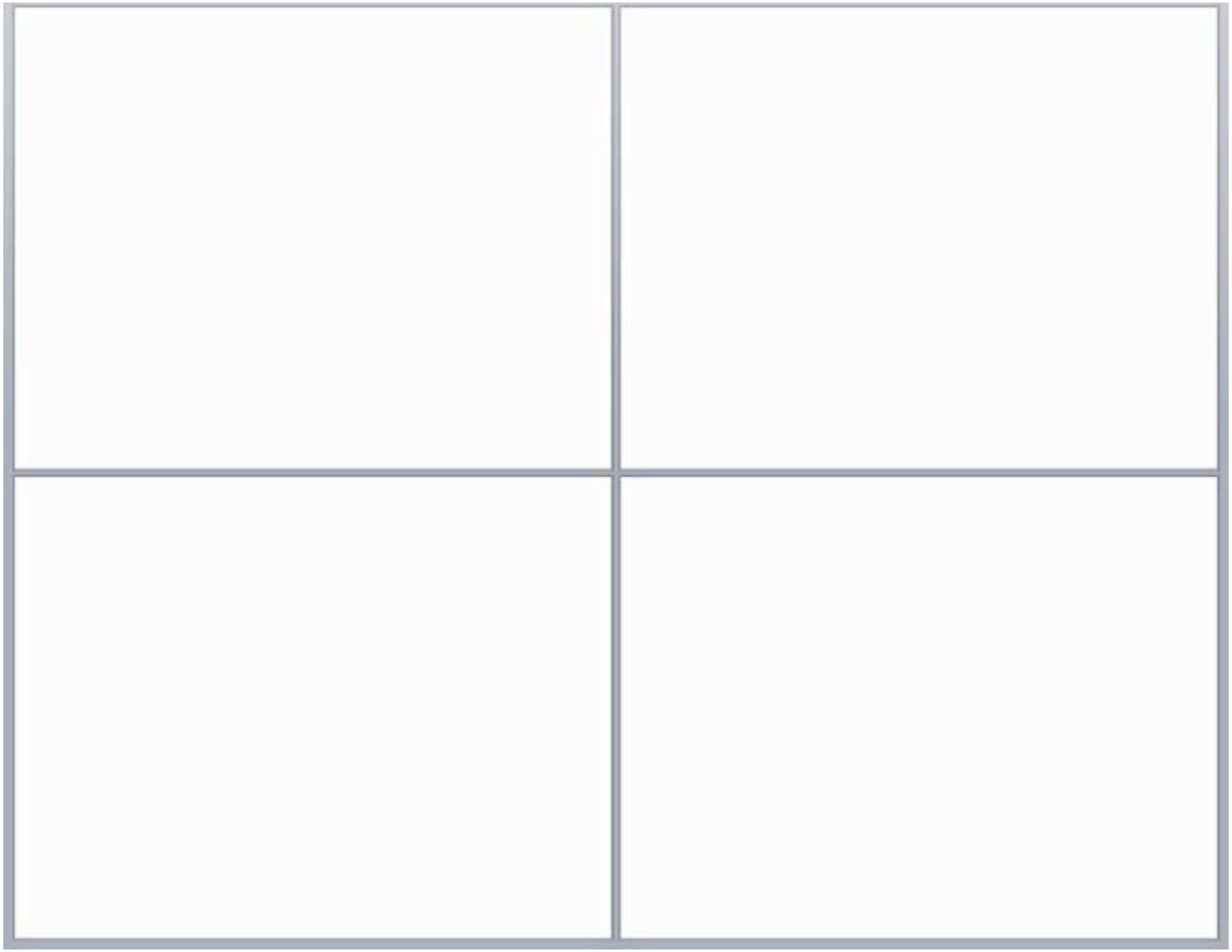


Tramonto WJDC Functional

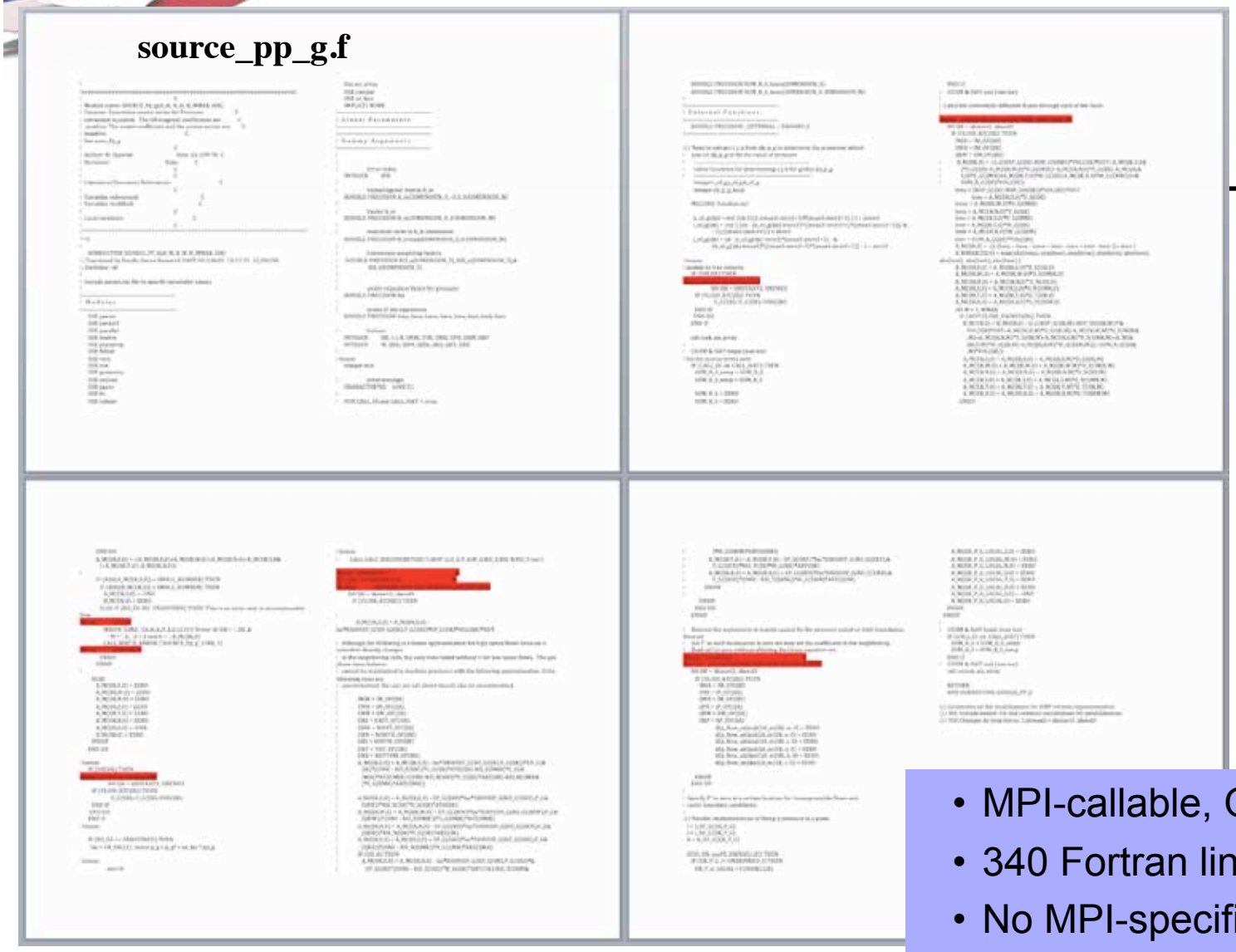
- New functional.
- Bonded systems.
- 552 lines C code.

WJDC-DFT (Werthim, Jain, Dominik, and Chapman) theory for bonded systems. ***(S. Jain, A. Dominik, and W.G. Chapman. Modified interfacial statistical associating fluid theory: A perturbation density functional theory for inhomogeneous complex fluids. J. Chem. Phys., 127:244904, 2007.)*** **Models stoichiometry constraints inherent to bonded systems.**

How much MPI-specific code?



dft_fill_wjdc.c
MPI-specific
code



MFIX

Source term for pressure correction

- MPI-callable, OpenMP-enabled.
- 340 Fortran lines.
- No MPI-specific code.
- Ubiquitous OpenMP markup (red regions).

Reasons for MPI Success?

- Portability? Yes.
- Standardized? Yes.
- Momentum? Yes.
- Separation of many Parallel & Algorithms concerns? Big Yes.

- Once framework in place:
  - Sophisticated physics added as serial code.
  - Ratio of science experts vs. parallel experts: 10:1.

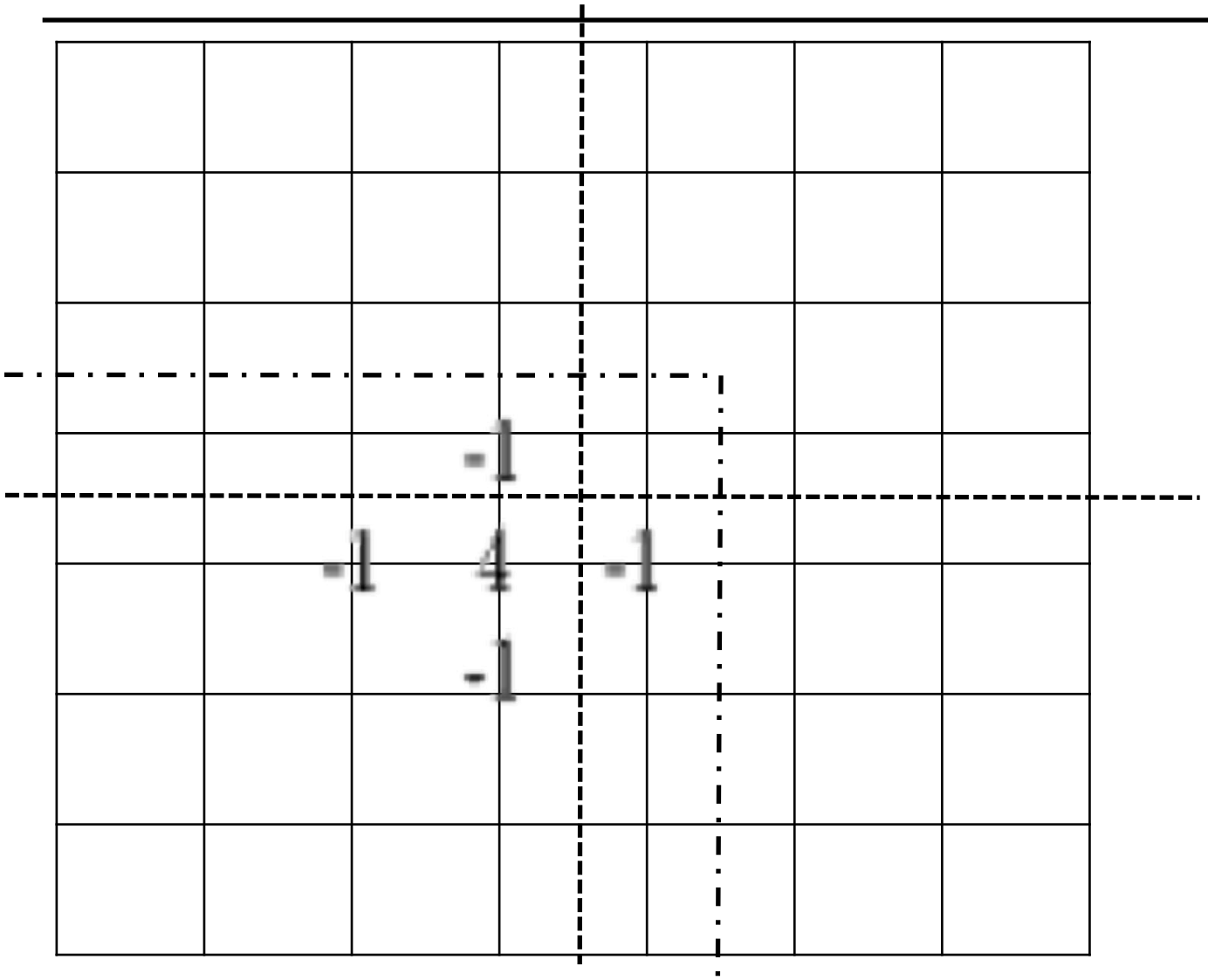
- Key goal for new parallel apps: Preserve this ratio

Sandia National Laboratories

Single Program Multiple Data (SPMD) 101

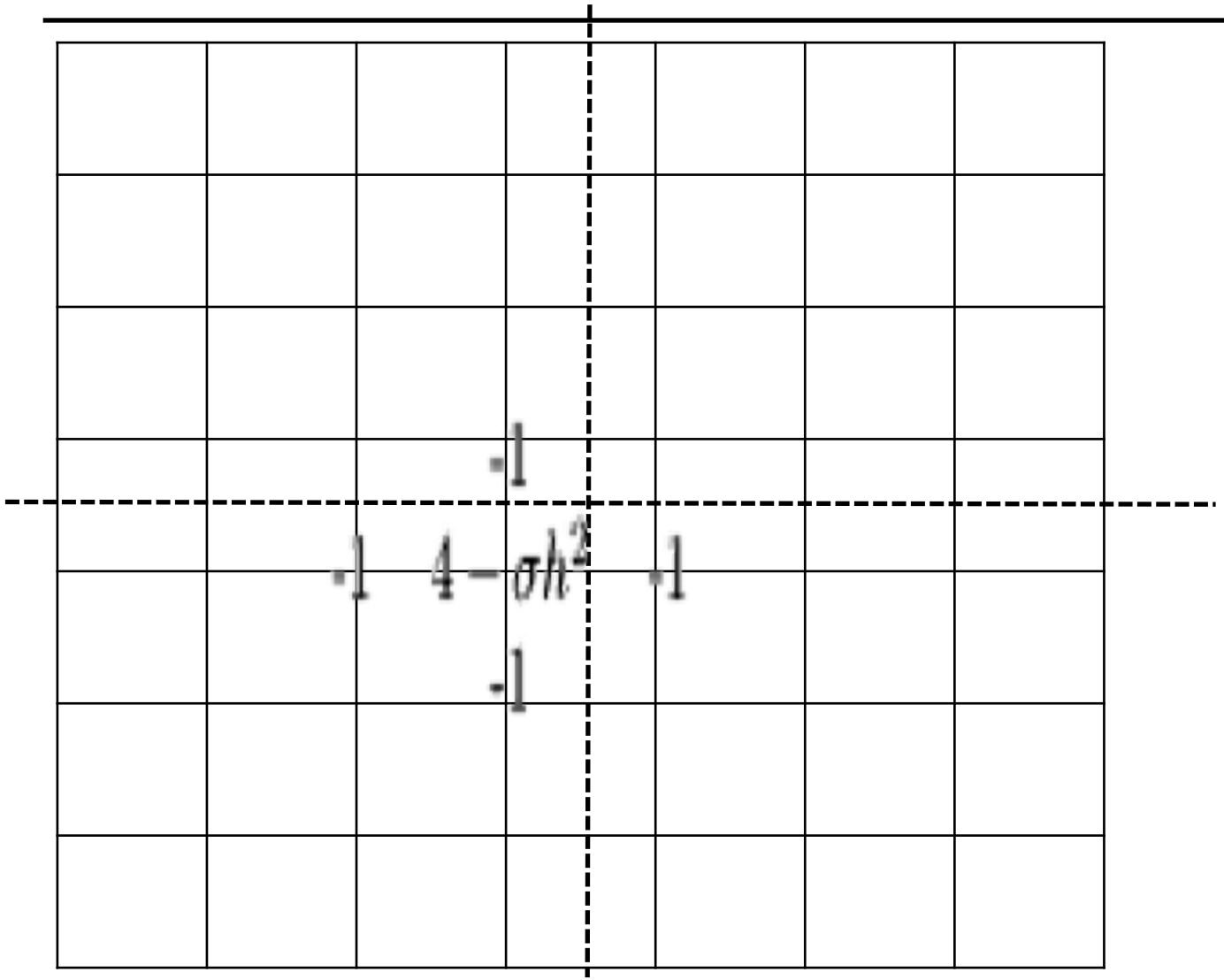
Sandia National Laboratories

2D PDE on Regular Grid (Standard Laplace)



Sandia National Laboratories

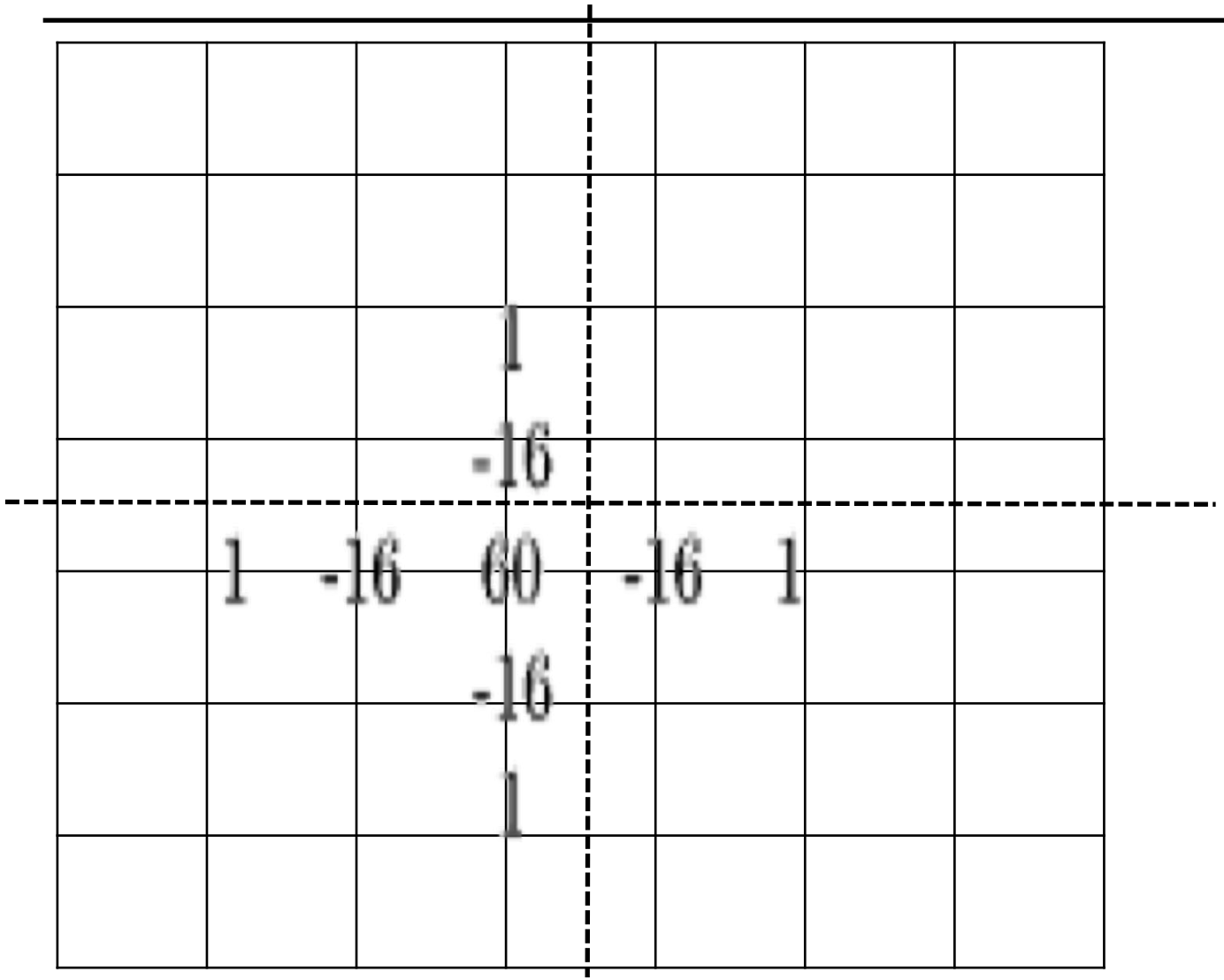
2D PDE on Regular Grid (Helmholtz)



$$-\nabla u - \sigma u = f \qquad (\sigma \geq 0)$$

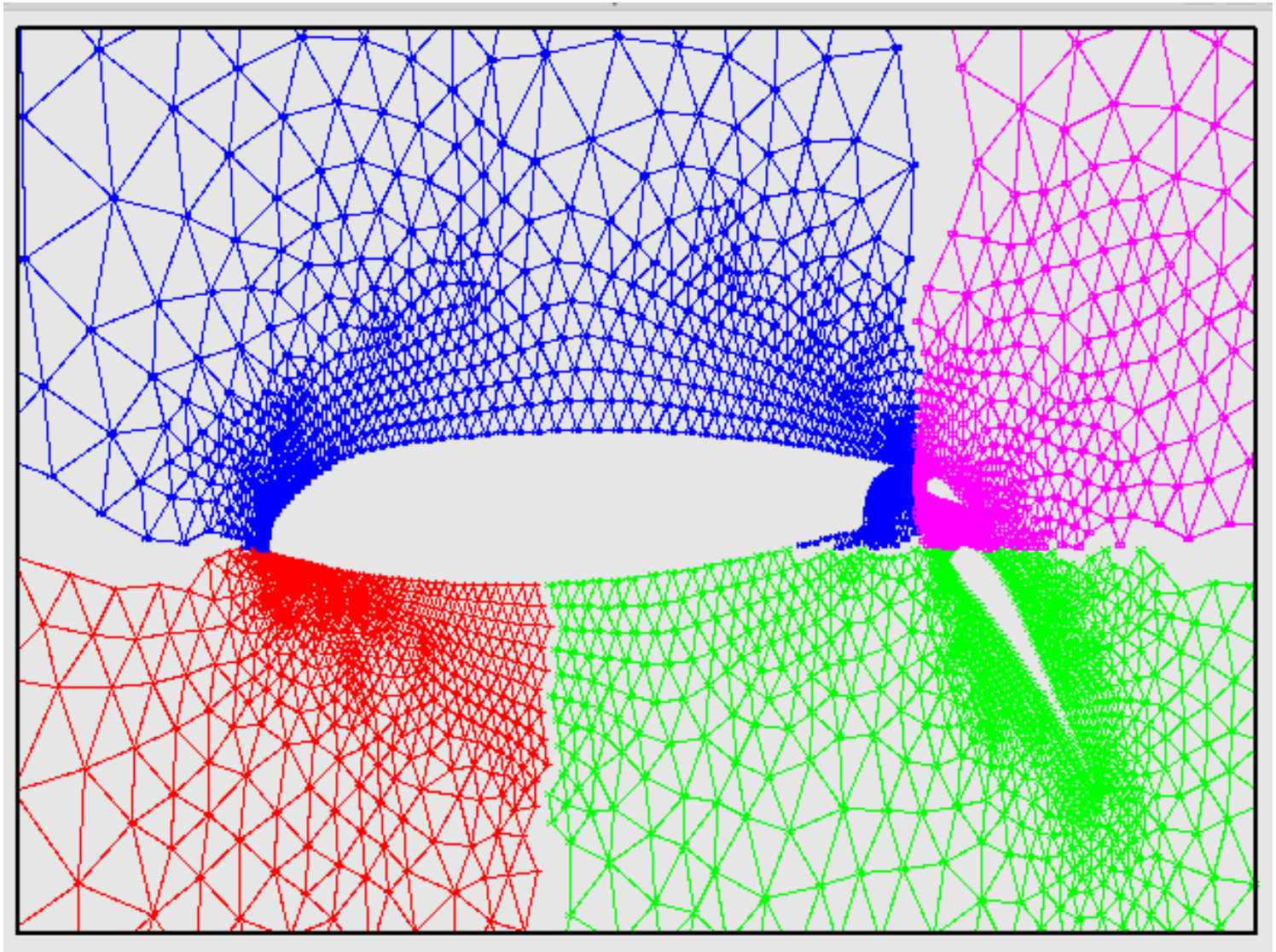
Sandia National Laboratories

2D PDE on Regular Grid (4th Order Laplace)



Sandia National Laboratories

More General Mesh and Partitioning

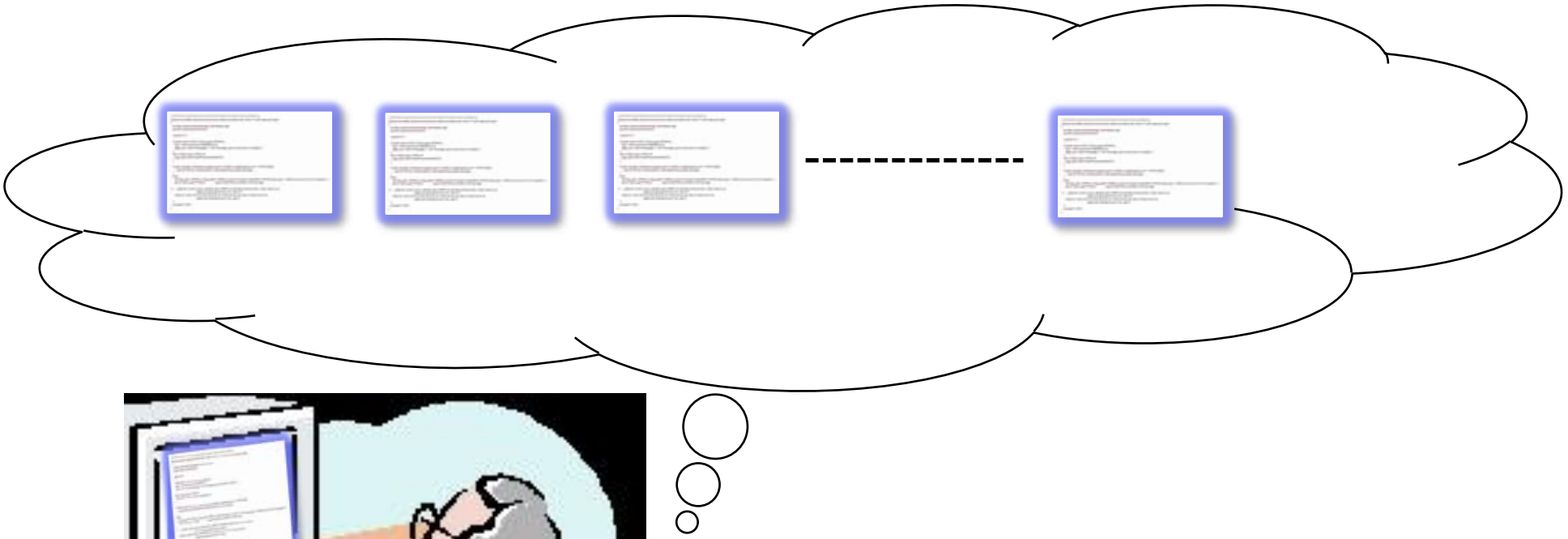


Sandia National Laboratories

SPMD Patterns for Domain Decomposition

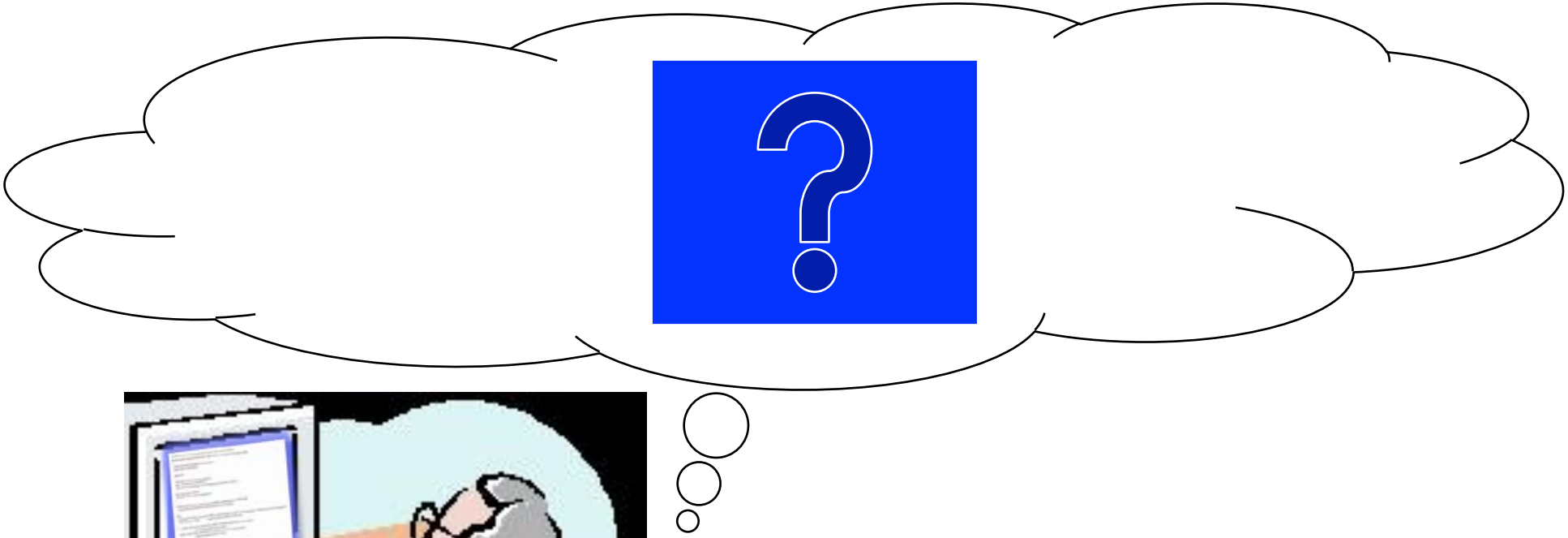
- Halo Exchange:
  - Conceptual.
  - Needed for any partitioning, halo layers.
  - MPI is simply portability layer.
  - Could be replace by PGAS, one-sided, …
- Collectives:
  - Dot products, norms.
- All other programming:
  - Sequential!!!

Computational Domain Expert Writing MPI Code



Sandia National Laboratories

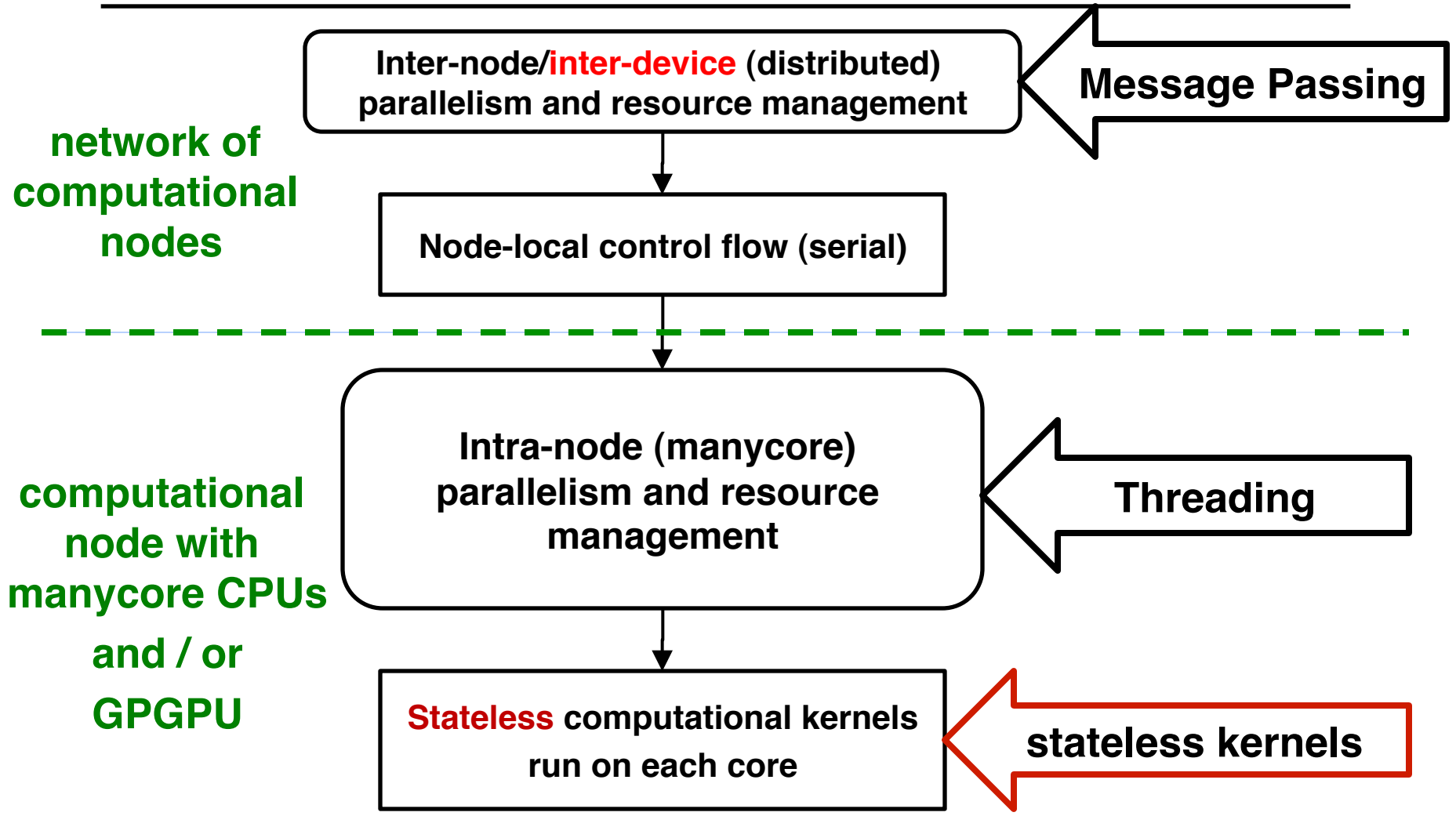
Computational Domain Expert Writing Future Parallel Code



Sandia National Laboratories

Evolving Parallel Programming Model

Parallel Programming Model: Multi-level/Multi-device



Adapted from slide of H. Carter Edwards

Domain Scientist's Parallel Palette

- MPI-only (SPMD) apps:
  - Single parallel construct.
  - Simultaneous execution.
  - Parallelism of even the messiest serial code.
- MapReduce:
  - Plug-n-Play data processing framework - 80% Google cycles.
- Pregel: Graph framework (other 20%)
- Next-generation PDE and related applications:
  - Internode:
    - MPI, yes, or something like it.
    - Composed with intranode.
  - Intranode:
    - Much richer palette.
    - More care required from programmer.
- What are the constructs in our new palette?

Sandia National Laboratories

Obvious Constructs/Concerns

- Parallel for:
  forall (i, j) in domain {…}
  - No loop-carried dependence.
  - Rich loops.
  - Use of shared memory for temporal reuse, efficient device data transfers.
- Parallel reduce:

```
forall (i, j) in domain {
      xnew(i, j) = …;
      delx+= abs(xnew(i, j) - xold(i, j));
}
```

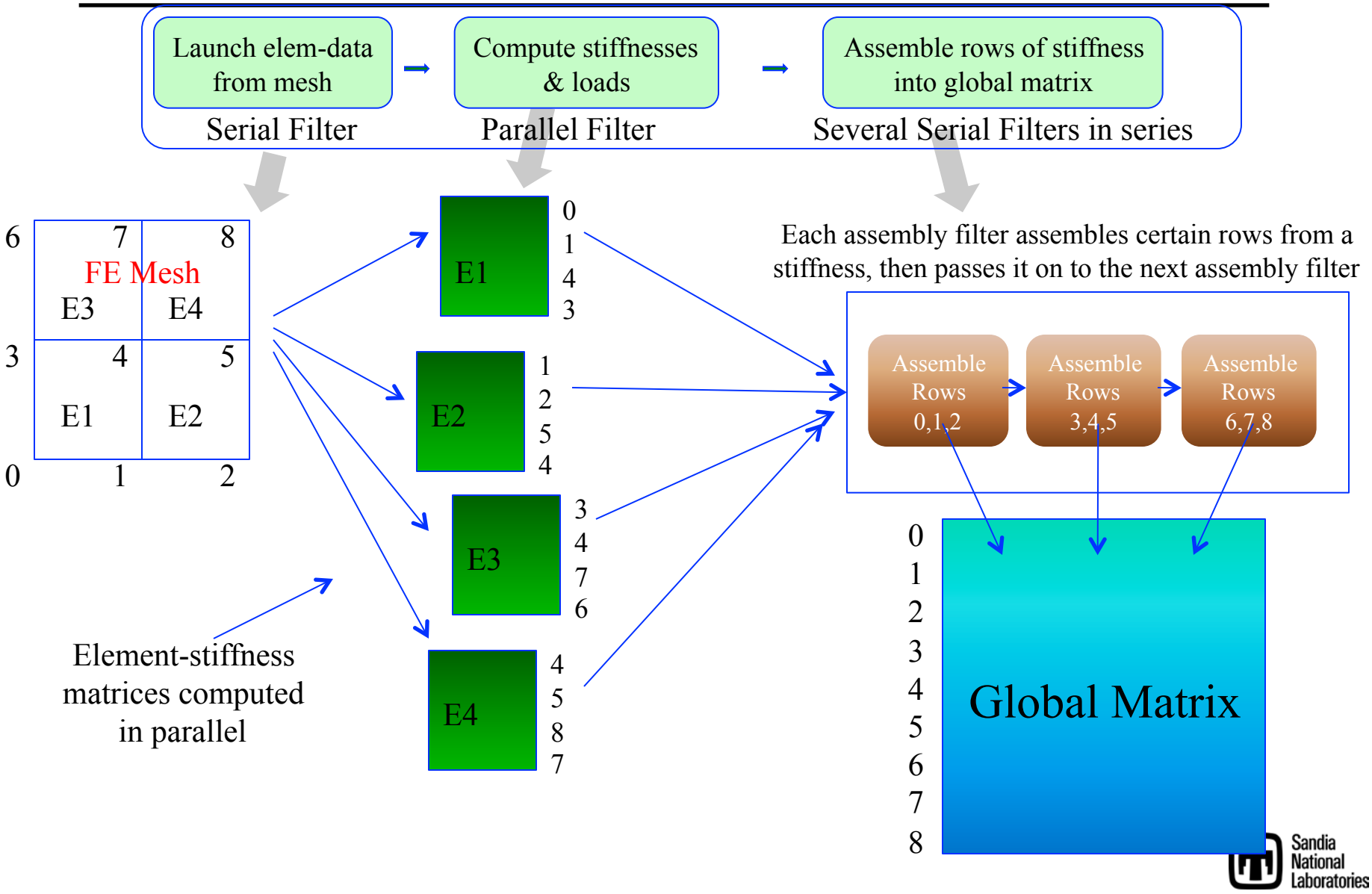
  - Couple with other computations.
  - Concern for reproducibility.

Sandia National Laboratories

Other construct: Pipeline

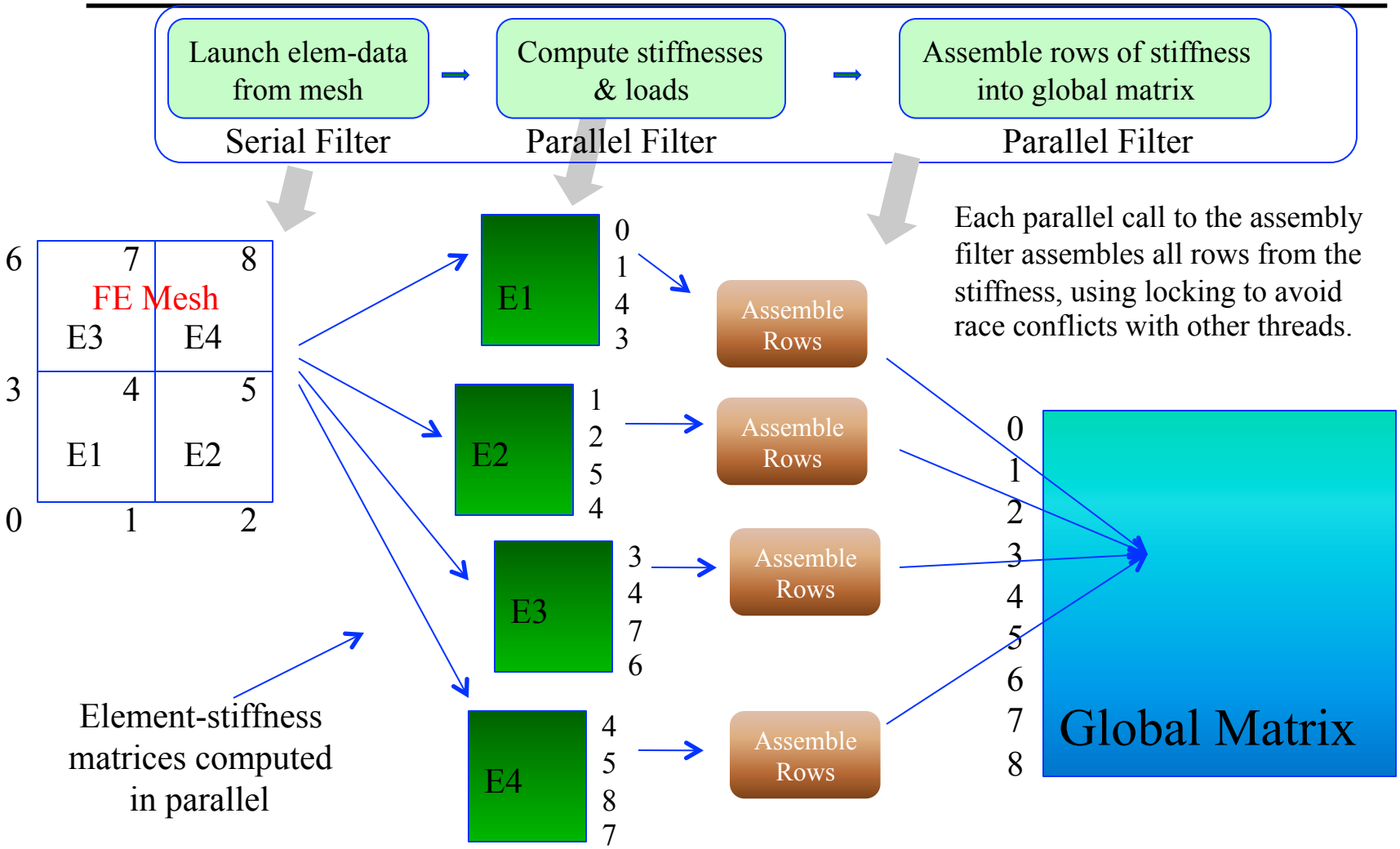
- Sequence of filters.
- Each filter is:
  - Sequential (grab element ID, enter global assembly) or
  - Parallel (fill element stiffness matrix).
- Filters executed in sequence.
- Programmer's concern:
  - Determine (conceptually): Can filter execute in parallel?
  - Write filter (serial code).
  - Register it with the pipeline.
- Extensible:
  - New physics feature.
  - New filter added to pipeline.

TBB Pipeline for FE assembly



Sandia National Laboratories

Alternative
TBB Pipeline for FE assembly



Sandia National Laboratories

Base-line FE Assembly Timings

Problem size: 80x80x80 == 512000 elements, 531441 matrix-rows
The finite-element assembly performs 4096000 matrix-row sum-into operations
(8 per element) and 4096000 vector-entry sum-into operations.

MPI-only, no threads. Linux dual quad-core workstation.

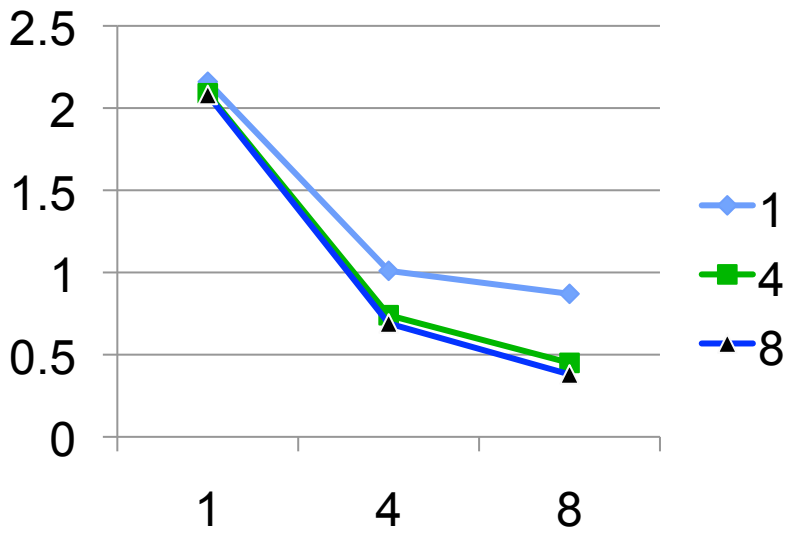
Num-procs	Assembly-time Intel 11.1	Assembly-time GCC 4.4.4
1	1.80s	1.95s
4	0.45s	0.50s
8	0.24s	0.28s

Sandia National Laboratories

FE Assembly Timings

Problem size: 80x80x80 == 512000 elements, 531441 matrix-rows
The finite-element assembly performs 4096000 matrix-row sum-into operations (8 per element) and 4096000 vector-entry sum-into operations.

No MPI, only threads. Linux dual quad-core workstation.



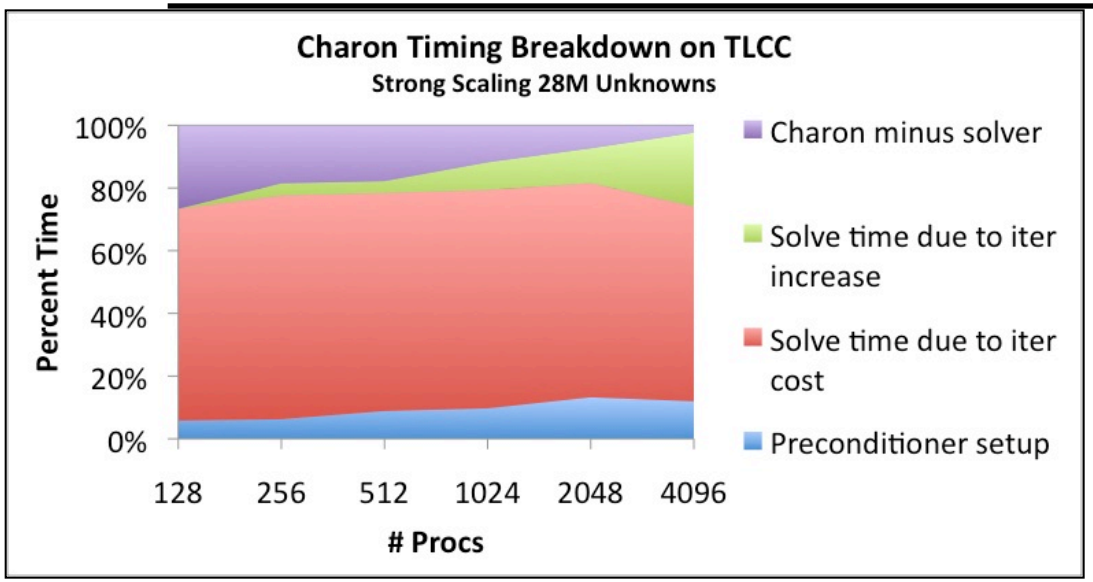
Num-threads	Elem-group-size	Matrix-conflicts	Vector-conflicts	Assembly-time
1	1	0	0	2.16s
1	4	0	0	2.09s
1	8	0	0	2.08s
4	1	95917	959	1.01s
4	4	7938	25	0.74s
4	8	3180	4	0.69s
8	1	64536	1306	0.87s
8	4	5892	49	0.45s
8	8	1618	1	0.38s

Sandia National Laboratories

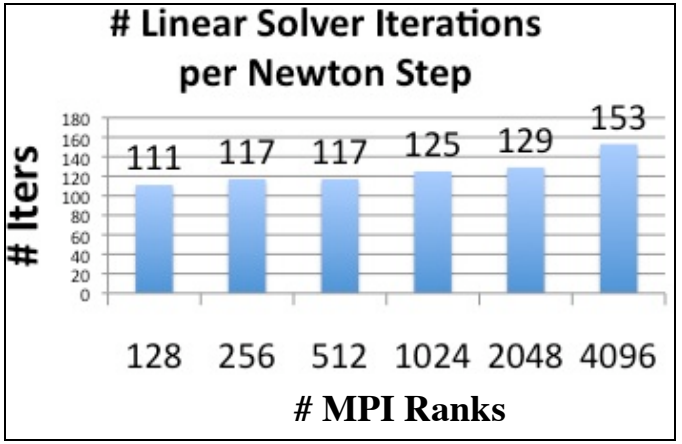
Other construct: Thread team

- Multiple threads.
- Fast barrier.
- Shared, fast access memory pool.
- Example: Nvidia SM
- X86 more vague, emerging more clearly in future.

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.**

Sandia National Laboratories

Thread Team Advantanges

- Qualitatively better algorithm:
  - Threaded triangular solve scales.
  - Fewer MPI ranks means fewer iterations, better robustness.
- Exploits:
  - Shared data.
  - Fast barrier.
  - Data-driven parallelism.

Finite Elements/Volumes/Differences and parallel node constructs

- Parallel for, reduce, pipeline:
  - Sufficient for vast majority of node level computation.
  - Supports:
    - Complex modeling expression.
    - Vanilla parallelism.
  - Must be “stencil-aware” for temporal locality.
- Thread team:
  - Complicated.
  - Requires true parallel algorithm knowledge.
  - Useful in solvers.

Programming Today for Tomorrow's Machines

Programming Today for Tomorrow's Machines

- Parallel Programming in the small:
  - Focus: writing sequential code fragments.
  - Programmer skills:
    - 10%: Pattern/framework experts (domain-aware).
    - 90%: Domain experts (pattern-aware)
- Languages needed are already here.
  - Exception: Large-scale data-intensive graph?

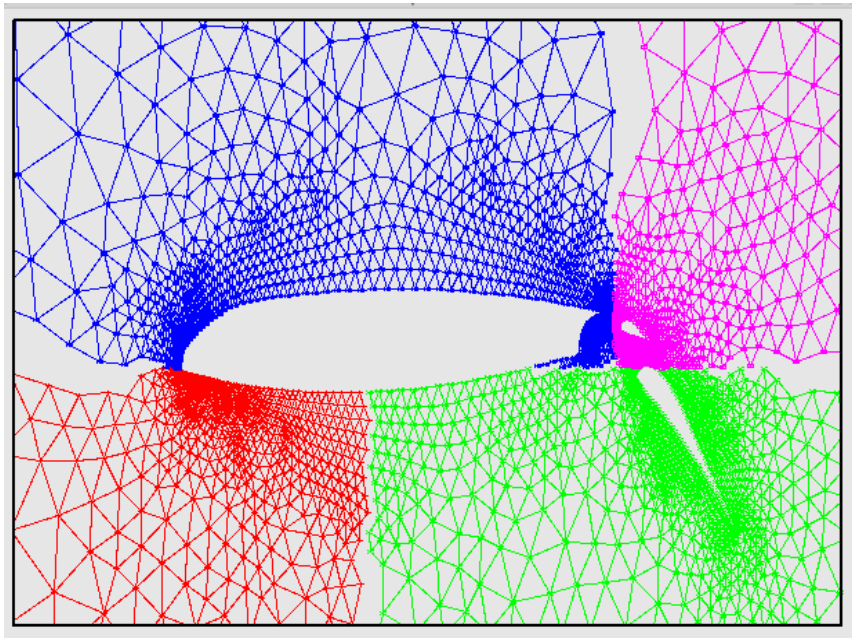
Sandia National Laboratories

FE/FV/FD Parallel Programming Today

```
for ((i,j,k) in points/elements on subdomain) {
  compute coefficients for point (i,j,k)
  inject into global matrix
  }
```

Notes:

- User in charge of:
  - Writing physics code.
  - Iteration space traversal.
  - Storage association.
- Pattern/framework/runtime in charge of:
  - SPMD execution.

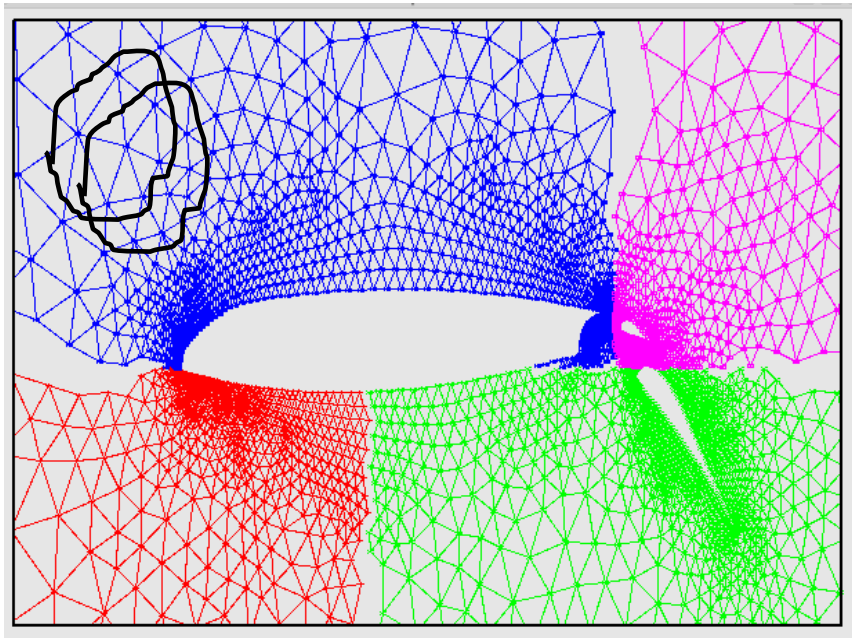


FE/FV/FD Parallel Programming Tomorrow

```
pipeline <i,j,k> {
  filter(addPhysicsLayer1<i,j,k)>);
  ...
  filter(addPhysicsLayern<i,j,k>);
  filter(injectIntoGlobalMatrix<i,j,k>);
 }
```

Notes:

- User in charge of:
  - Writing physics code (filter).
  - Registering filter with framework.
- Pattern/framework/runtime in charge of:
  - SPMD execution.
  - Iteration space traversal.
    - Sensitive to temporal locality.
  - Filter execution scheduling.
  - Storage association.
- Better assignment of responsibility (in general).



Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
4. All future programmers will need to write parallel code.

Portable Multi/Manycore Programming
Trilinos/Kokkos Node API

Generic Node Parallel Programming via C++ Template Metaprogramming

- Goal: Don't repeat yourself (DRY).
- Every parallel programming environment supports basic patterns: parallel_for, parallel_reduce.
  - OpenMP:
    ```
    #pragma omp parallel for
    for (i=0; i<n; ++i) {y[i] += alpha*x[i];}
    ```
  - Intel TBB:
    ```
    parallel_for(blocked_range<int>(0, n, 100), loopRangeFn(…));
    ```
  - CUDA:
    ```
    loopBodyFn<<< nBlocks, blockSize >>> (…);
    ```
- How can we write code once for all these (and future) environments?

Sandia National Laboratories

Tpetra and Kokkos

- Tpetra is an implementation of the Petra Object Model.
  - Design is similar to Epetra, with appropriate deviation.
  - Fundamental differences:
    - heavily exploits templates
    - utilizes hybrid (distributed + shared) parallelism via Kokkos Node API
- Kokkos is an API for shared-memory parallel nodes
  - Provides parallel_for and parallel_reduce skeletons.
  - Support shared memory APIs:
    - ThreadPool Interface (TPI; Carter Edwards's pthreads Trilinos package)
    - Intel Threading Building Blocks (TBB)
    - NVIDIA CUDA-capable GPUs (via Thrust)
    - *OpenMP (implemented by Radu Popescu/EPFL)*

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
- Practical goal: Cover 80% kernels with generic code.

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);
```

Sandia National Laboratories

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    );
```

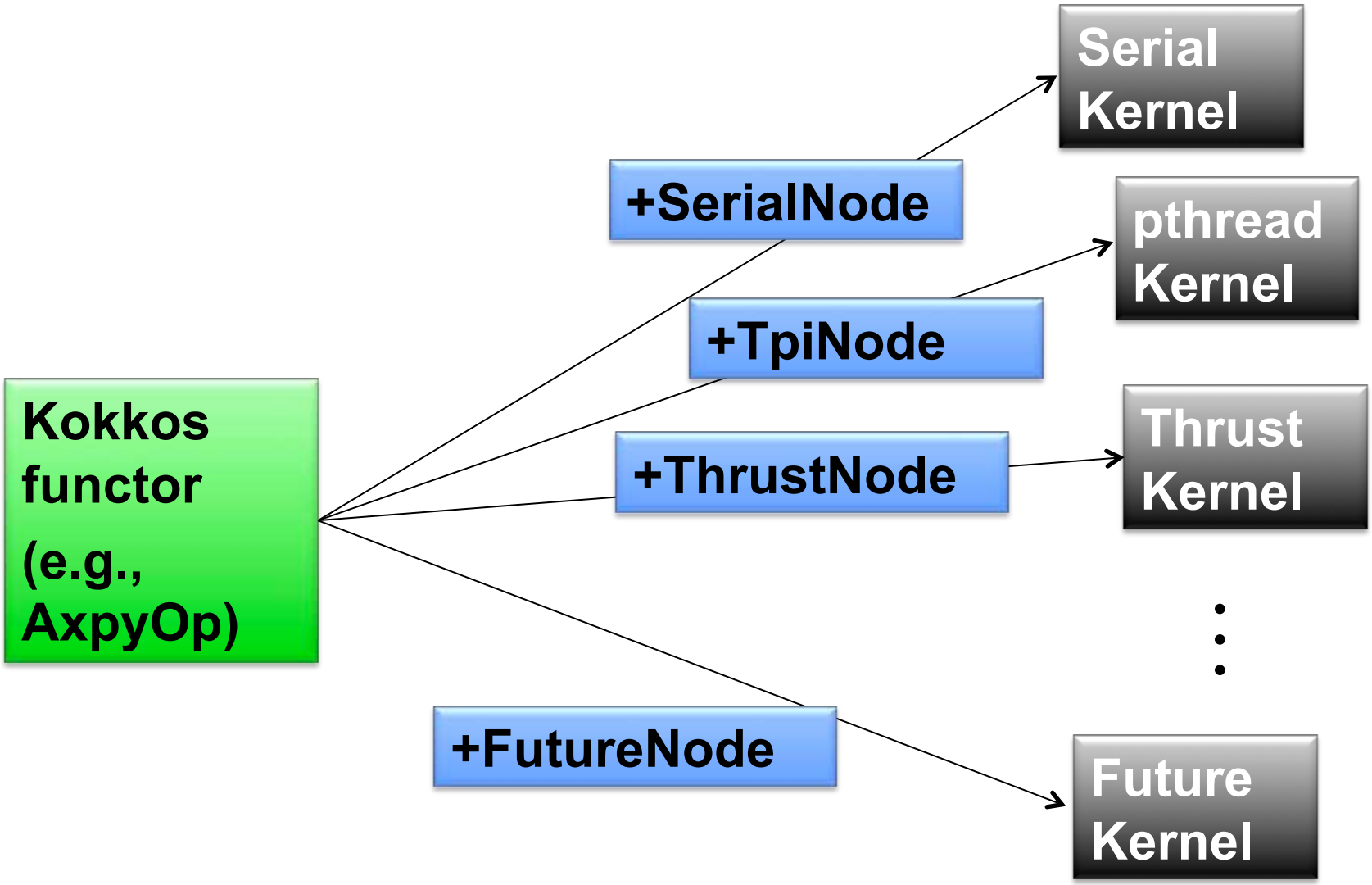
```
template <class T>
struct AxpyOp {
  const T * x;
  T * y;
  T alpha, beta;
  void execute(int i)
  { y[i] = alpha*x[i] + beta*y[i]; }
};
```

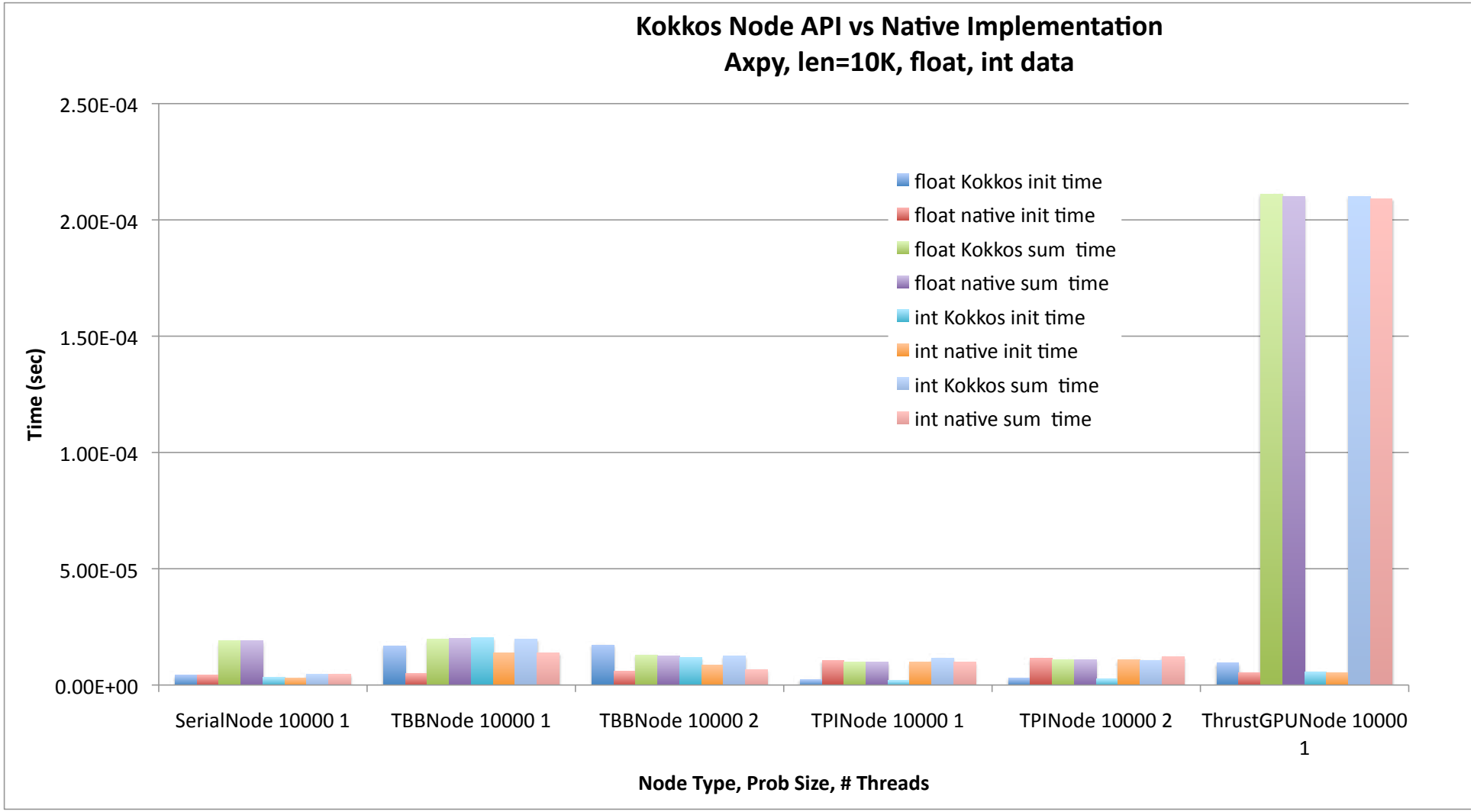
```
template <class T>
struct DotOp {
  typedef T ReductionType;
  const T * x, * y;
  T identity()       { return (T)0;      }
  T generate(int i)  { return x[i]*y[i]; }
  T reduce(T x, T y) { return x + y;     }
};
```

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

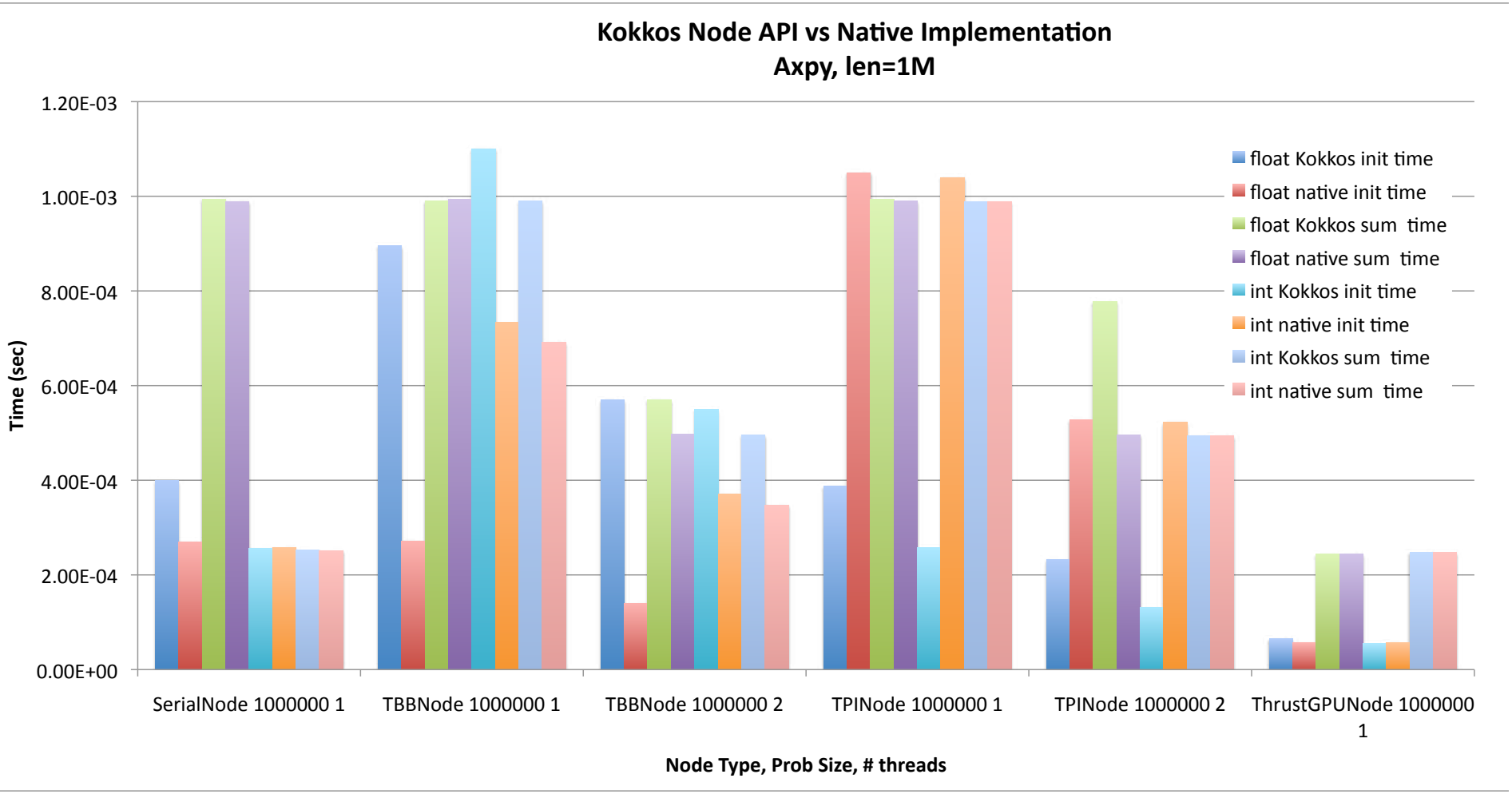
```
DotOp<float> op;
op.x = ...;  op.y = ...;
float dot;
dot = node.parallel_reduce< DotOp<float> >
                          (0, length, op);
```

Compile-time Polymorphism

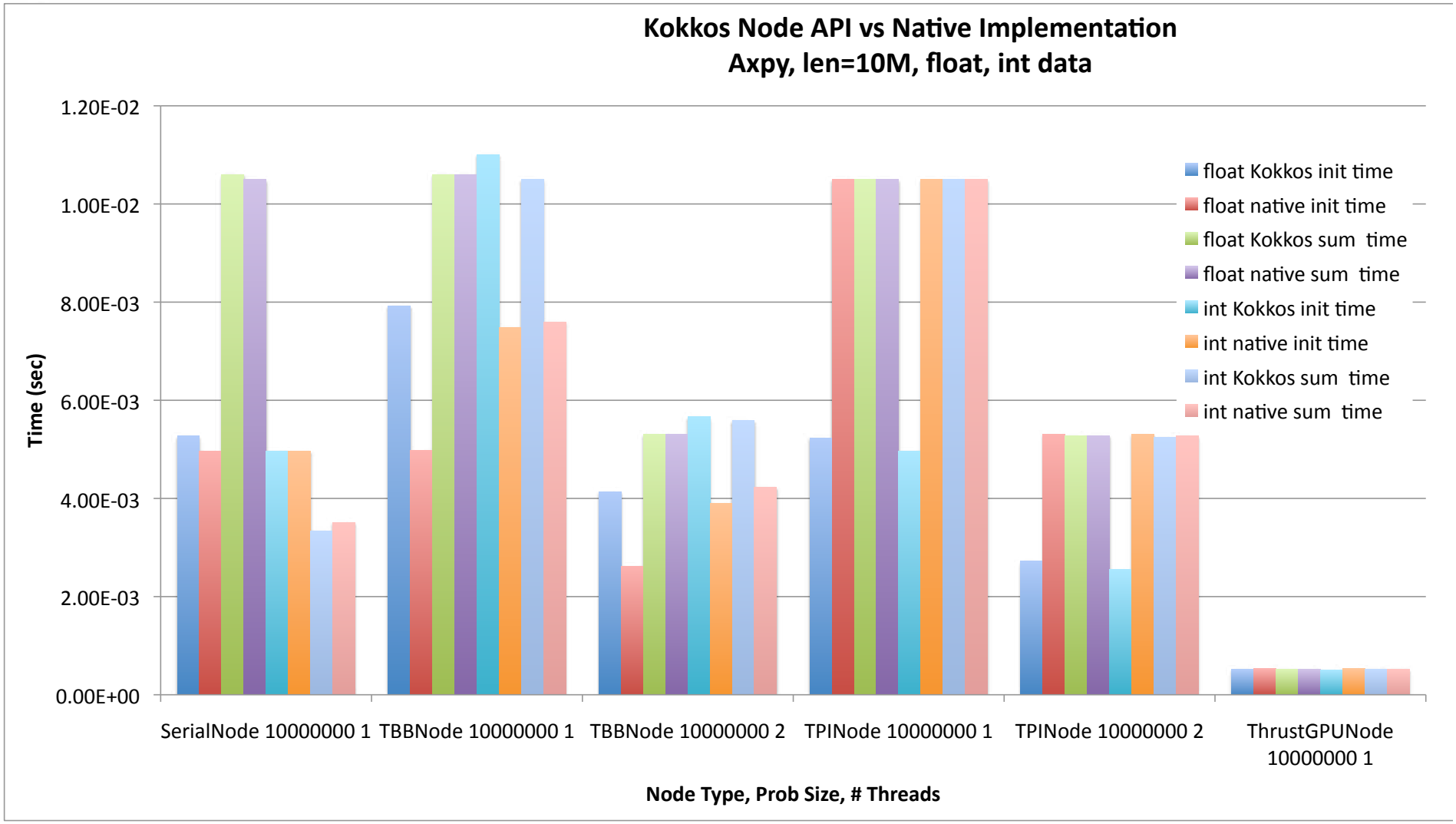




Sandia National Laboratories



Sandia National Laboratories



Sandia National Laboratories

What's the Big Deal about Vector-Vector Operations?

Examples from OOQP (Gertz, Wright)

$$y_i \leftarrow y_i + \alpha x_i z_i \quad ,i = 1\ldots n$$

$$y_i \leftarrow y_i / x_i \quad ,i = 1\ldots n$$

$$y_i \leftarrow \begin{cases} y^{\min} - y_i & \text{if } y_i < y^{\min} \\ y^{\max} - y_i & \text{if } y_i > y^{\max} \\ 0 & \text{if } y^{\min} \le y_i \le y^{\max} \end{cases} \quad ,i = 1\ldots n$$

$$\alpha \leftarrow \{\max \alpha : x + \alpha d \ge \beta\}$$

Example from TRICE (Dennis, Heinkenschloss, Vicente)

$$d_i \leftarrow \begin{cases} (b-u)_i^{1/2} & \text{if } w_i < 0 \text{ and } b_i < +\infty \\ 1 & \text{if } w_i < 0 \text{ and } b_i = +\infty \\ (u-a)_i^{1/2} & \text{if } w_i \ge 0 \text{ and } a_i . > -\infty \\ 1 & \text{if } w_i \ge 0 \text{ and } a_i . = -\infty \end{cases} \quad ,i = 1\ldots n$$

Example from IPOPT (Waechter)

$$x_i \leftarrow \begin{cases} \left(x_i^L + \dfrac{\left(x_i^U - x_i^L \right)}{2} \right) & \text{if } \ddot{o}^L{}_i > \ddot{o}^U{}_i \\ \ddot{o}^L{}_i & \text{if } x_i < \ddot{o}^L{}_i \\ \ddot{o}^U{}_i & \text{if } x_i > \ddot{o}^U{}_i \end{cases} \quad ,i = 1\ldots n$$

$$where: \quad \begin{aligned} \ddot{o}^L{}_i &= \min\left(x_i^L + \eta\left(x_i^U - x_i^L \right) x_i^L + \delta \right) \\ \ddot{o}^U{}_i &= \max\left(x_i^L - \eta\left(x_i^U - x_i^L \right) x_i^U - \delta \right) \end{aligned}$$

Many different and unusual vector operations are needed by interior point methods for optimization!

Currently in MOOCHO : > 40 vector operations!

Sandia National Laboratories

Tpetra RTI Components

- Set of stand-alone non-member methods:
  - `unary_transform<UOP>(Vector &v, UOP op)`
  - `binary_transform<BOP>(Vector &v1, const Vector &v2, BOP op)`
  - `reduce<G>(const Vector &v1, const Vector &v2, G op_glob)`
  - `binary_pre_transform_reduce<G>( Vector &v1, const Vector &v2, G op_glob)`
- These are non-member methods of Tpetra::RTI which are loosely coupled with Tpetra::MultiVector and Tpetra::Vector.
- Tpetra::RTI also provides Operator-wrappers:
  - `class KernelOp<..., Kernel > : Tpetra::Operator<...>`
  - `class BinaryOp<...,BinaryOp> : Tpetra::Operator<...>`

Tpetra RTI Example

```
// isn't this nicer than a bunch of typedefs?
auto &platform = Tpetra::DefaultPlatform::getDefaultPlatform();
auto comm = platform.getComm();
auto node = platform.getNode();

// create Map and some Vector objects
Tpetra::global_size_t numGlobalRows = ...;
auto map = createUniformContigMapWithNode<int,int>(numGlobalRows, comm, node);
const size_t numLocalRows = map->getNodeNumElements();
auto x = Tpetra::createVector<float>(map),
     y = Tpetra::createVector<float>(map);
auto z = Tpetra::createVector<double>(map),
     w = Tpetra::createVector<double>(map);

// parallel initialization of x[i] = 1.0 using C++-0x lambda function
Tpetra::RTI::unary_transform(  *x,      [](float xi){return 1.0f;} );
// parallel initialization of y[i] = x[i]
Tpetra::RTI::binary_transform( *y, *x, [](float, float xi) {return xi;} );
// parallel y[i] = x[i] + y[i]
Tpetra::RTI::binary_transform( *y, *x, std::plus<float>() );
// parallel single precision dot(x,y)
fresult = Tpetra::RTI::reduce( *x, *y, reductionGlob<ZeroOp<float>>(
                                         std::multiplies<float>(),
                                         std::plus<float>() ));
```

Future Node API Trends

- TBB provides very rich pattern-based API.
  - It, or something very much like it, will provide environment for sophisticated parallel patterns.
- Simple patterns: FutureNode may simply be OpenMP.
  - OpenMP handles parallel_for, parallel_reduce fairly well.
  - Deficiencies being addressed.
  - Some evidence it can beat CUDA.
- OpenCL practically unusable?
  - Functionally portable.
  - Performance not.
  - Breaks the DRY principle.

Additional Benefits of Templates

Multiprecision possibilities

- 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

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

```
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;}
```

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
```

```
#ifndef TPETRA_POWER_METHOD_HPP
#define TPETRA_POWER_METHOD_HPP

#include <Tpetra_Operator.hpp>
#include <Tpetra_Vector.hpp>
#include <Teuchos_ScalarTraits.hpp>

namespace TpetraExamples {

 /** \brief Simple power iteration eigensolver for a Tpetra::Operator.
  */
 template <class Scalar, class Ordinal>
 Scalar powerMethod(const Teuchos::RCP<const Tpetra::Operator<Scalar,Ordinal> > &A,
                    int niters, typename Teuchos::ScalarTraits<Scalar>::magnitudeType tolerance,
                    bool verbose)
 {
   typedef typename Teuchos::ScalarTraits<Scalar>::magnitudeType Magnitude;
   typedef Tpetra::Vector<Scalar,Ordinal> Vector;

   if ( A->getRangeMap() != A->getDomainMap() ) {
     throw std::runtime_error("TpetraExamples::powerMethod(): operator must have domain and range maps that
are equivalent.");
   }
```

```
// create three vectors, fill z with random numbers
    Teuchos::RCP<Vector> z, q, r;
    q = Tpetra::createVector<Scalar>(A->getRangeMap());
    r = Tpetra::createVector<Scalar>(A->getRangeMap());
    z = Tpetra::createVector<Scalar>(A->getRangeMap());
    z->randomize();
    //
    Scalar lambda = 0.0;
    Teuchos::ScalarTraits<Scalar>::magnitudeType normz, residual = 0.0;
    // power iteration
    for (int iter = 0; iter < niters; ++iter) {
      normz = z->norm2();                    // Compute 2-norm of z
      q->scale(1.0/normz, *z);               // Set q = z / normz
      A->apply(*q, *z);                      // Compute z = A*q
      lambda = q->dot(*z);                   // Approximate maximum eigenvalue: lamba = dot(q,z)
      if ( iter % 100 == 0 || iter + 1 == niters ) {
        r->update(1.0, *z, -lambda, *q, 0.0);      // Compute A*q - lambda*q
        residual = Teuchos::ScalarTraits<Scalar>::magnitude(r->norm2() / lambda);
        if (verbose) {
          std::cout << "Iter = " << iter << "  Lambda = " << lambda
                    << "  Residual of A*q - lambda*q = " << residual   << std::endl;  }
      }
      if (residual < tolerance) {  break;  }
    }    return lambda;
  }
} // end of namespace TpetraExamples
```

Placement and Migration

Placement and Migration

- MPI:
  - Data/work placement clear.
  - Migration explicit.
- Threading:
  - It's a mess (IMHO).
  - Some platforms good.
  - Many not.
  - Default is bad (but getting better).
  - Some issues are intrinsic.

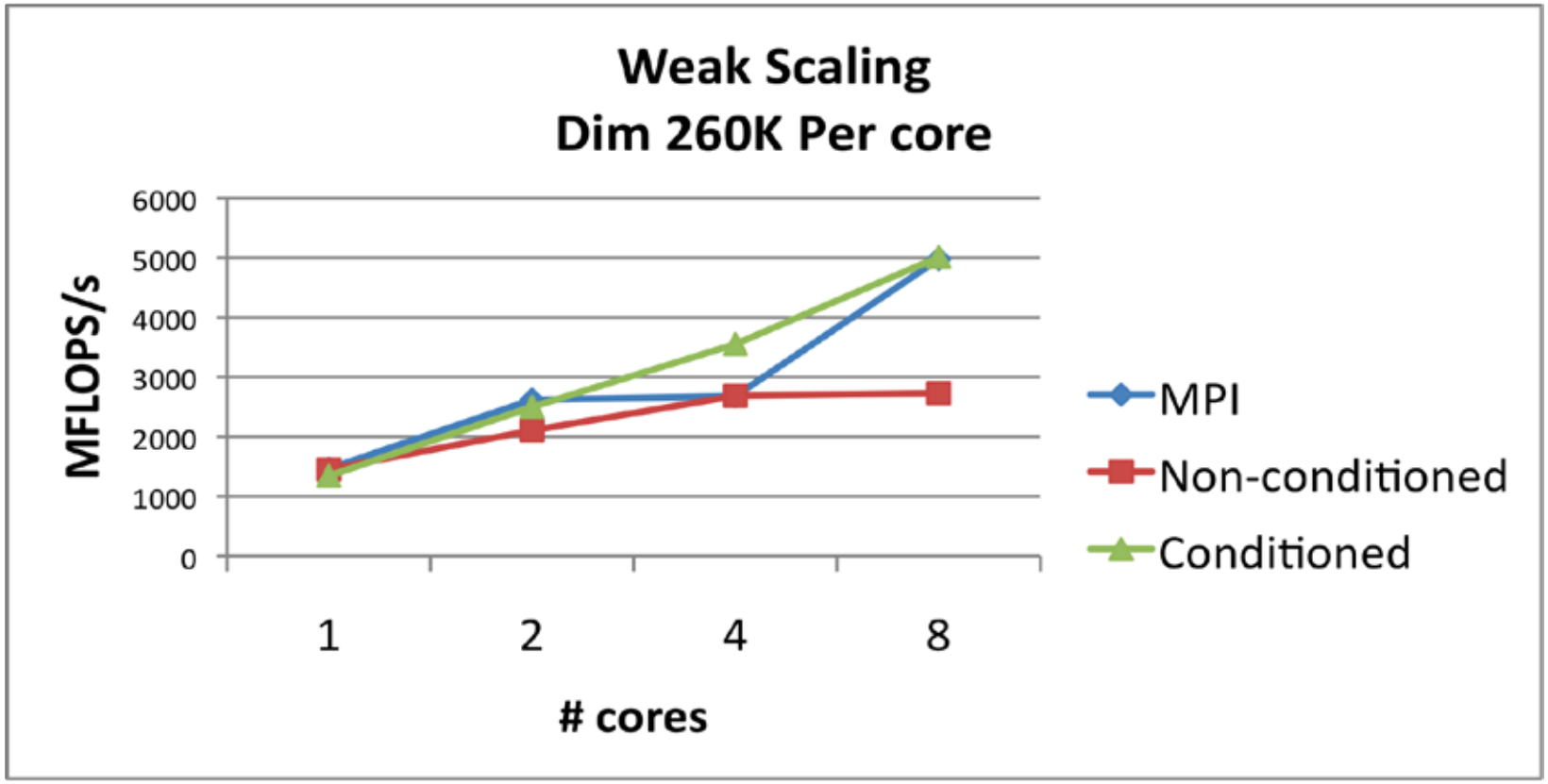
Data Placement on NUMA

- Memory Intensive computations: Page placement has huge impact.
- Most systems: First touch (except LWKs).
- 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).
- Note: This phenomenon especially damaging to OpenMP common usage.

Resilient Algorithms:

A little reliability, please.

My Luxury in Life (wrt FT/Resilience)

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

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

(W. Camp)

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.

Hard Error Futures

- C/R will continue as dominant approach:
  - Global state to global file system OK for small systems.
  - Large systems: State control will be localized, use SSD.
- Checkpoint-less restart:
  - Requires full vertical HW/SW stack co-operation.
  - Very challenging.
  - Stratified research efforts not effective.

Soft Error Futures

- Soft error handling: A legitimate algorithms issue.
- Programming model, runtime environment play role.

Sandia National Laboratories

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?

- Many apps are implicit.
- Most popular (nonsymmetric) linear solver is preconditioned GMRES.
- Only small subset of calculations need to be reliable.
  - GMRES is iterative, but also direct.

Every calculation matters

Soft Error Resilience

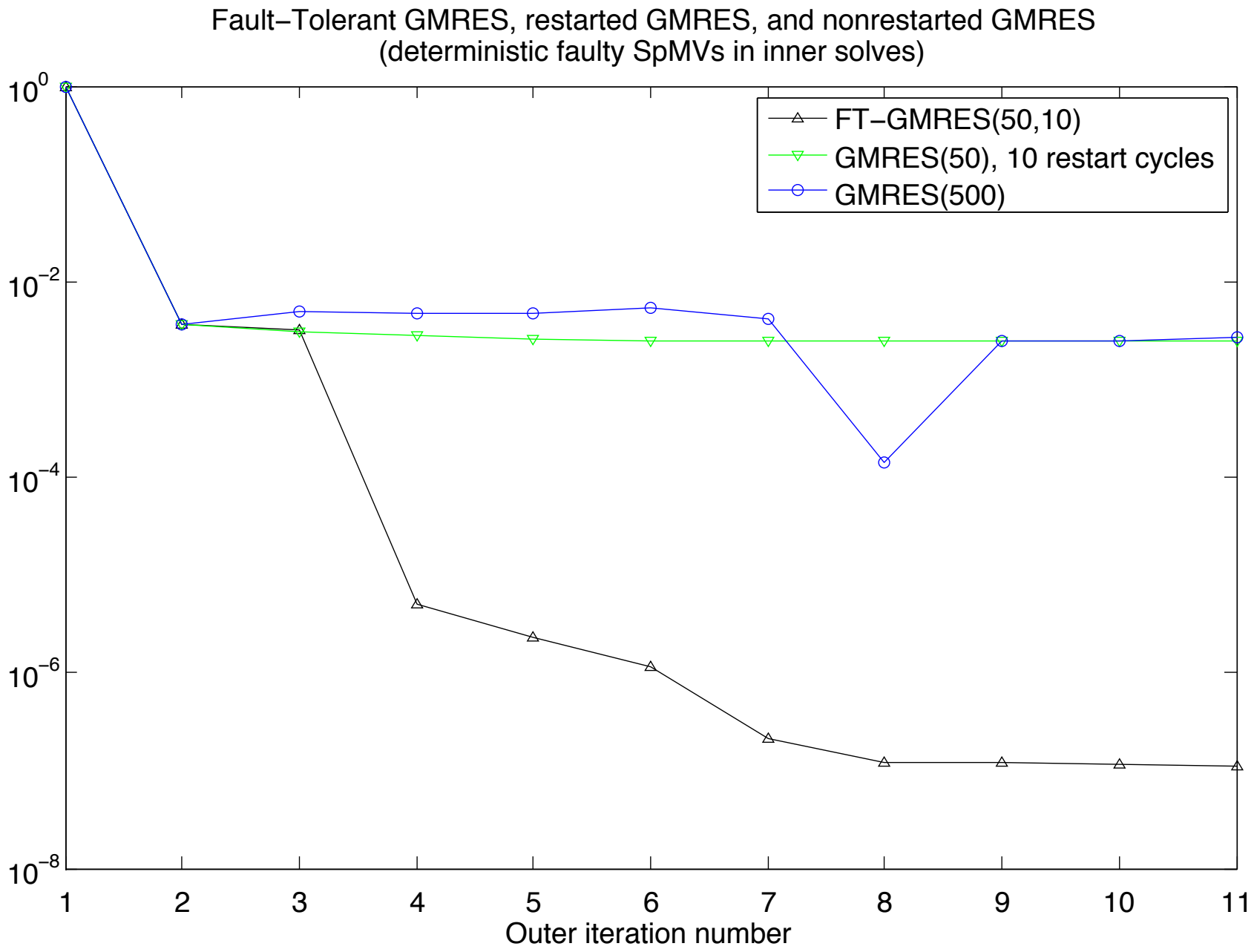
Description	Iters	FLOPS	Recursive Residual Error	Solution Error
All Correct Calcs	35	343M	4.6e-15	1.0e-6
Iter=2, y[1] += 1.0 SpMV incorrect Ortho subspace	35	343M	6.7e-15	3.7e+3
Q[1][1] += 1.0 Non-ortho subspace	N/C	N/A	7.7e-02	5.9e+5

- 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.

- 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: FT-GMRES.
  - Resilient to soft errors.
  - Outer solve: Highly Reliable
  - Inner solve: “bulk” reliability.
- General approach applies to many algorithms.

M. Heroux, M. Hoemmen

FTGMRES Results

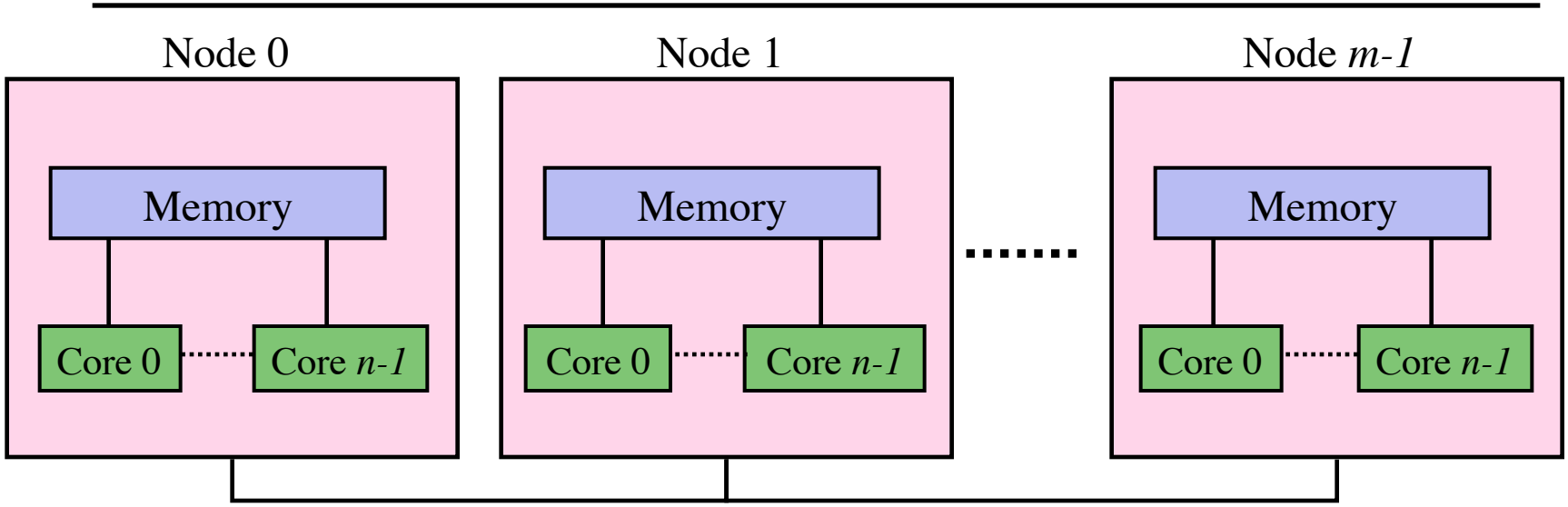


Quiz (True or False)

5. DRY is not possible across CPUs and GPUs.
6. Extended precision is too expensive to be useful.
7. Resilience will be built into algorithms.

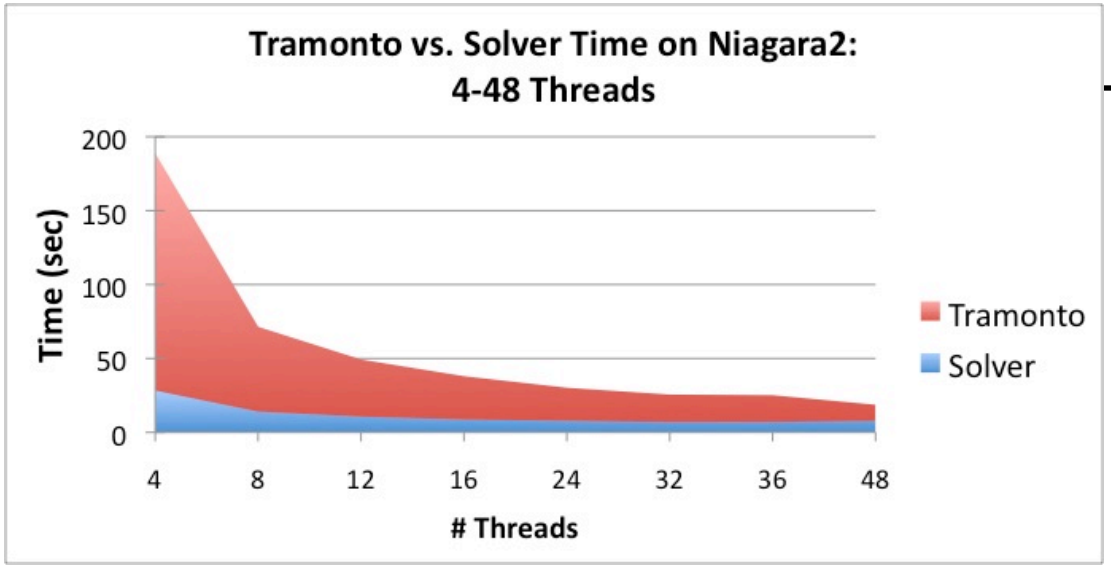
Bi-Modal: MPI-only and MPI+[X|Y|Z]

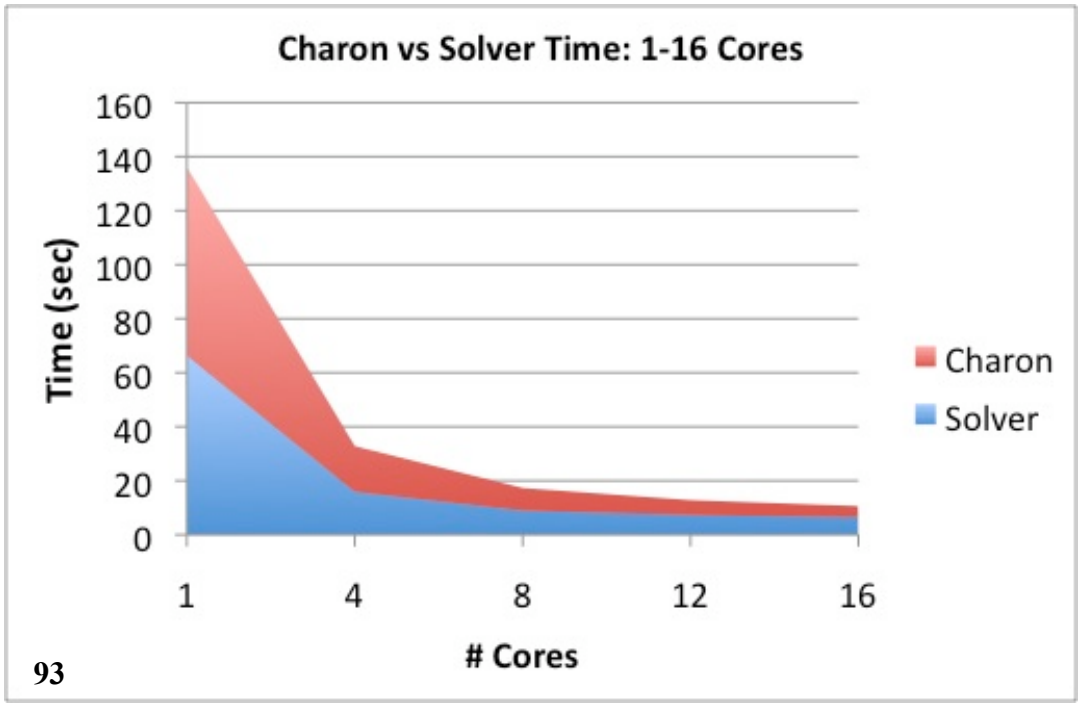
Parallel Machine Block Diagram



- 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).

Multicore Scaling: App vs. Solver





Application:

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

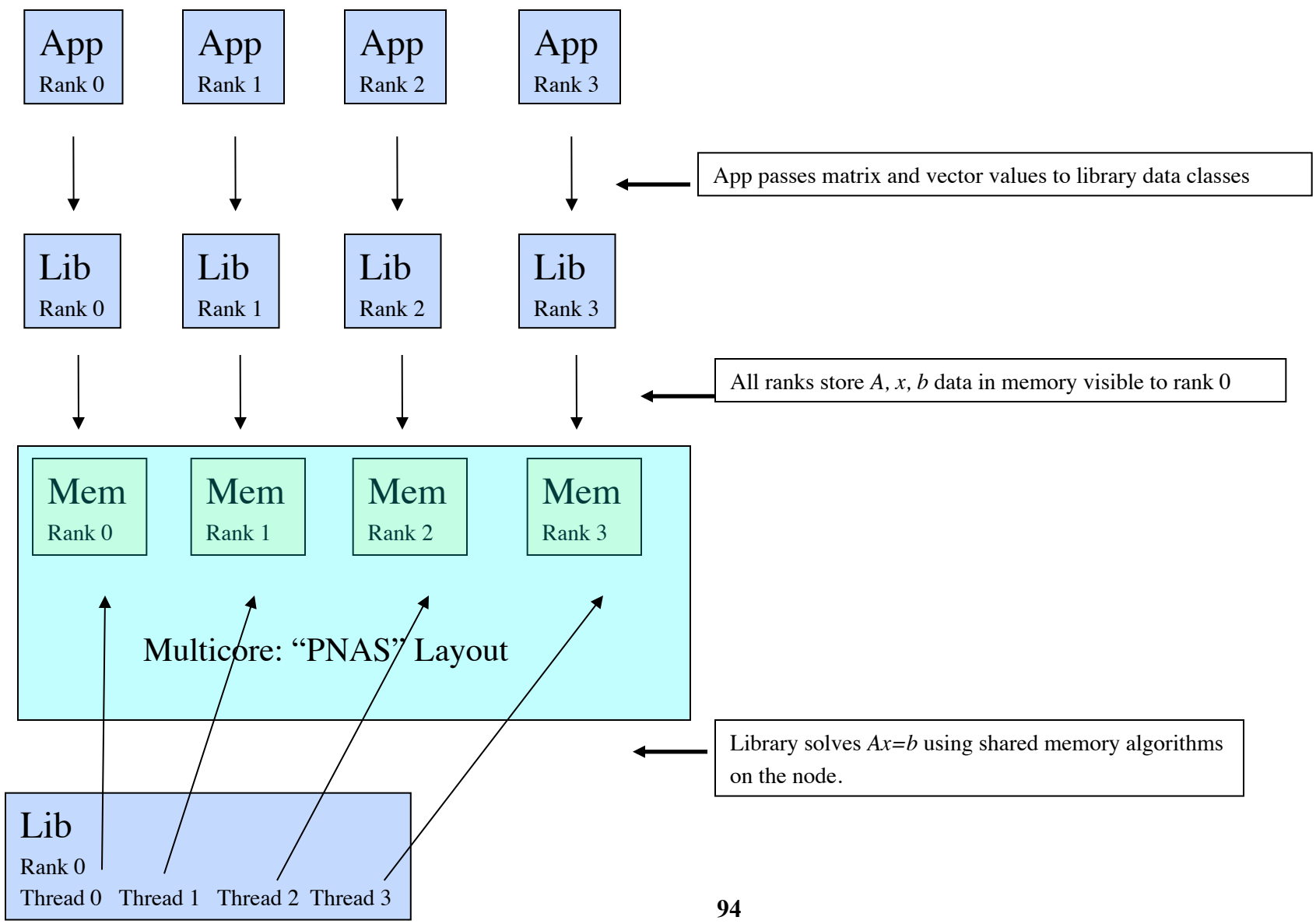
Solver:

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

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

Sandia National Laboratories

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



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);
```

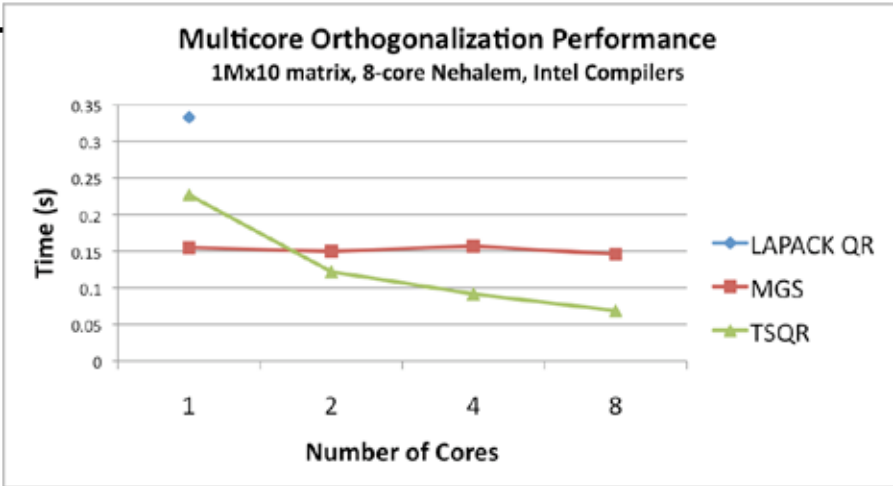
Collaborators: B. Barrett, Brightwell, Wolf - SNL; Vallee, Koenig - ORNL

Algorithms and Meta-Algorithms

Sandia National Laboratories

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:

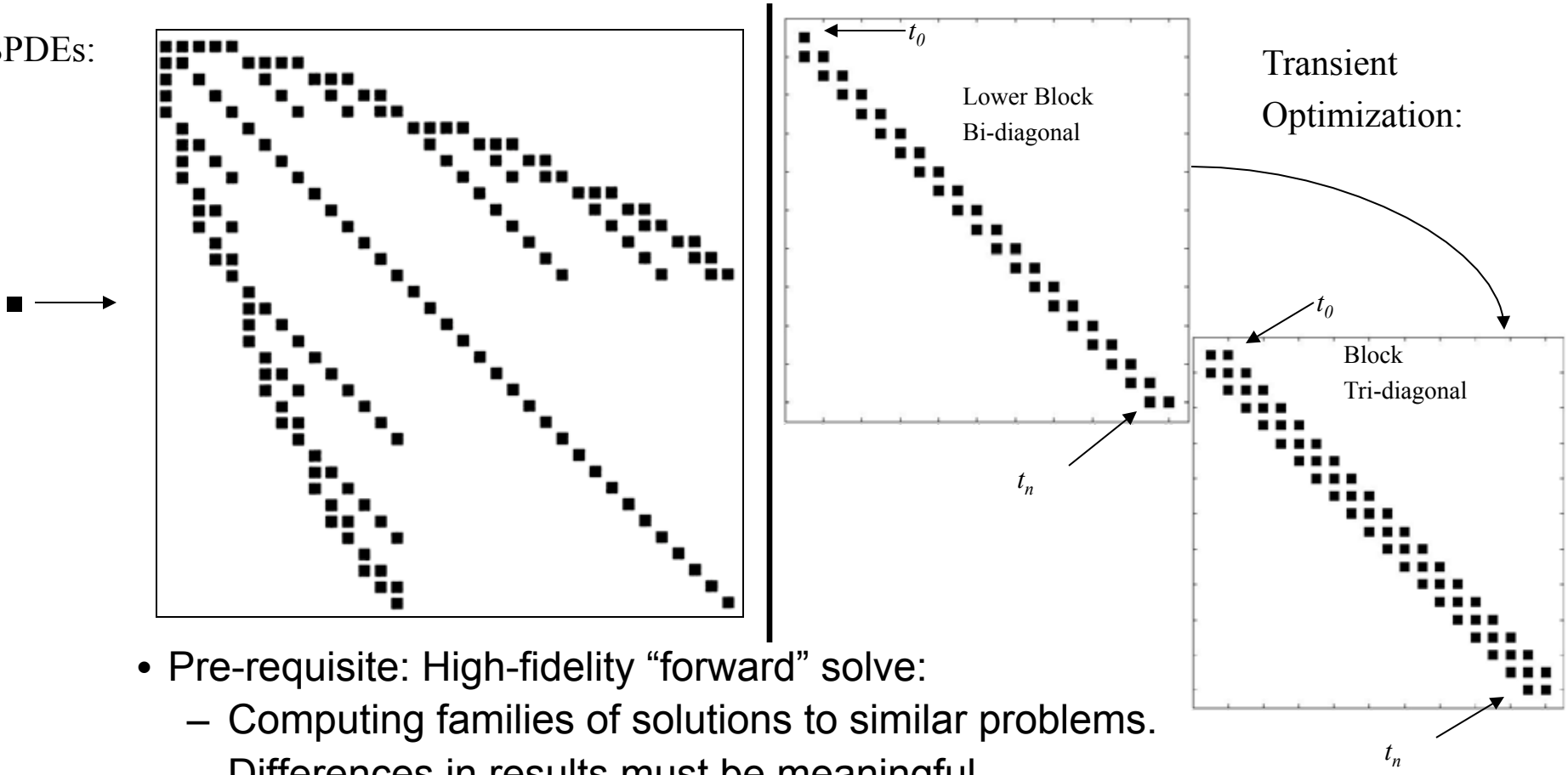
- Critical for exascale solvers.
- Part of the Trilinos scalable multicore capabilities.
- Helps all iterative solvers in Trilinos (available to external libraries, too).
- Staffing: Mark Hoemmen (lead, post-doc, UC-Berkeley), M. Heroux
- Part of Trilinos 10.6 release, Sep 2010.

Sandia National Laboratories

Advanced Modeling and Simulation Capabilities: Stability, Uncertainty and Optimization

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

SPDEs:



- 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.

Sandia National Laboratories

Accelerator-based Scalability Concerns

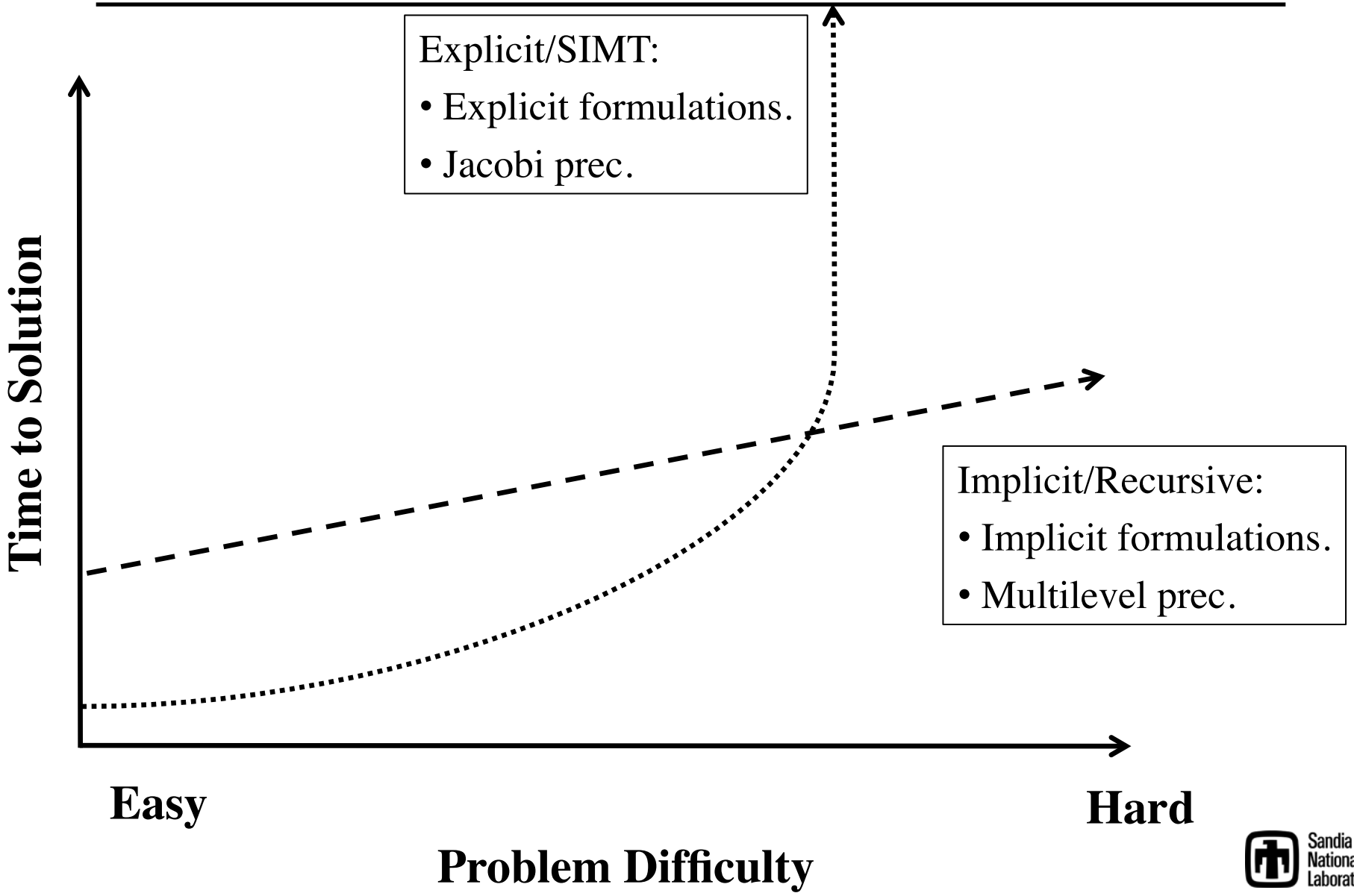
Global Scope Single Instruction Multiple Thread (SIMT) is too Restrictive

If FLOPS are free,
why are we making them cheaper?

Larry Wall:
Easy things should be easy, hard things should be possible.

Why are we making easy things easier and hard things impossible?

Explicit/SIMT vs. Implicit/Recursive Algorithms



Sandia National Laboratories

Problems with Accelerator-based Scalability

- Global SIMT is the only approach that really works well on GPUs, but:
  - Many of our most robust algorithms have no apparent SIMT replacement.
  - Working on it, but a lot to do, and fundamental issues at play.
- SMs might be useful to break SIMT mold, but:
  - Local store is way too small.
  - No market reason to make it bigger.
- Could consider SIMT approaches, but:
  - Broader apps community moving the other way:
    - Climate: Looking at implicit formulations.
    - Embedded UQ: Coupled formulations.
- Accelerator-based apps at risk?
  - Isolation from the broader app trends.
  - Accelerators good, but in combination with strong multicore CPU.

Summary

- Some app targets will change:
  - Advanced modeling and simulation: Gives a better answer.
  - Kernel set changes (including redundant computation).
- Resilience requires an integrated strategy:
  - Most effort at the system/runtime level.
  - C/R (with localization) will continue at the app level.
  - Resilient algorithms will mitigate soft error impact.
  - Use of validation in solution hierarchy can help.
- Building the next generation of parallel applications requires enabling domain scientists:
  - Write sophisticated methods.
  - Do so with serial fragments.
  - Fragments hoisted into scalable, resilient fragment.
- Success of manycore will require breaking out of global SIMT-only.

Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
4. All future programmers will need to write parallel code.
5. DRY is not possible across CPUs and GPUs
6. CUDA and OpenCL may be footnotes in computing history.
7. Extended precision is too expensive to be useful.
8. Resilience will be built into algorithms.
9. A solution with error bars complements architecture trends.
10. Global SIMT is sufficient parallelism for scientific computing.