SAND2011-3556C

# Toward portable programming of numerical linear algebra on manycore nodes

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, 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.

Sandia National Laboratories

# Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
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.
8. GPUs are a harbinger of CPU things to come.
9. Fortran Developers are in trouble in a manycore world.
10. Global SIMT is sufficient parallelism for scientific computing.

# Trilinos Contributors

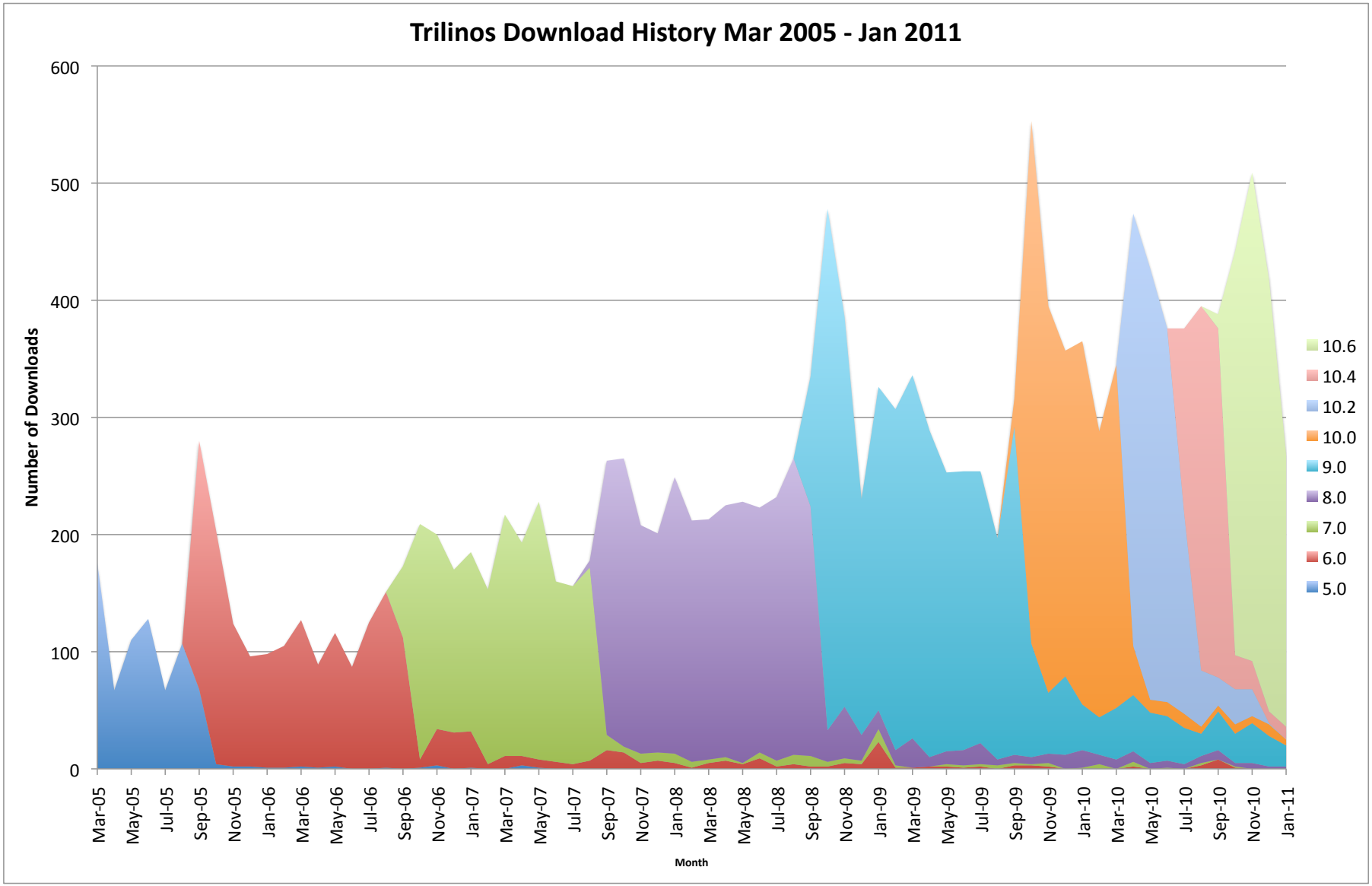
**Current Contributors**

**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

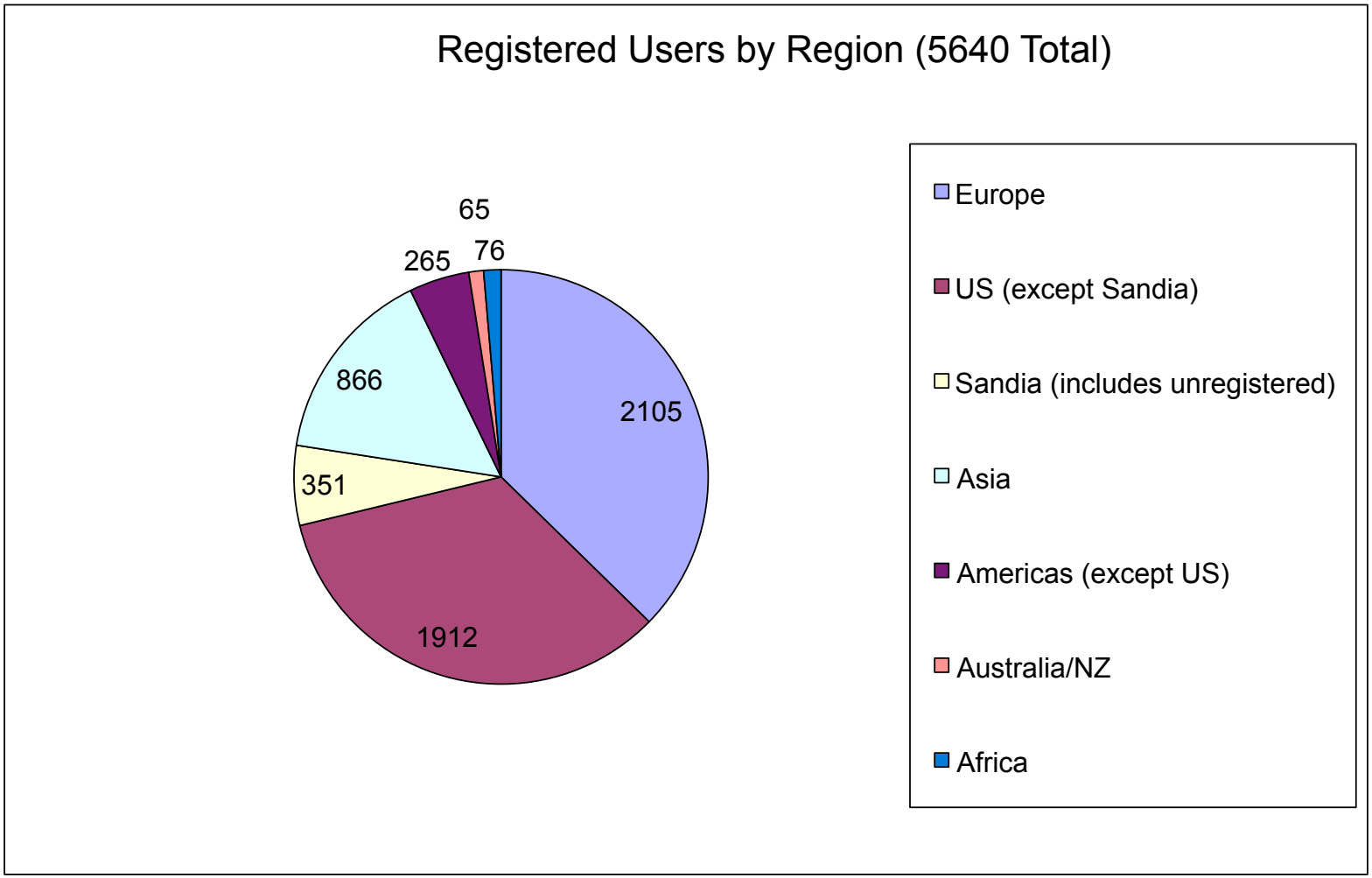
**Past Contributors**

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

# Trilinos Download History: 17600 Total

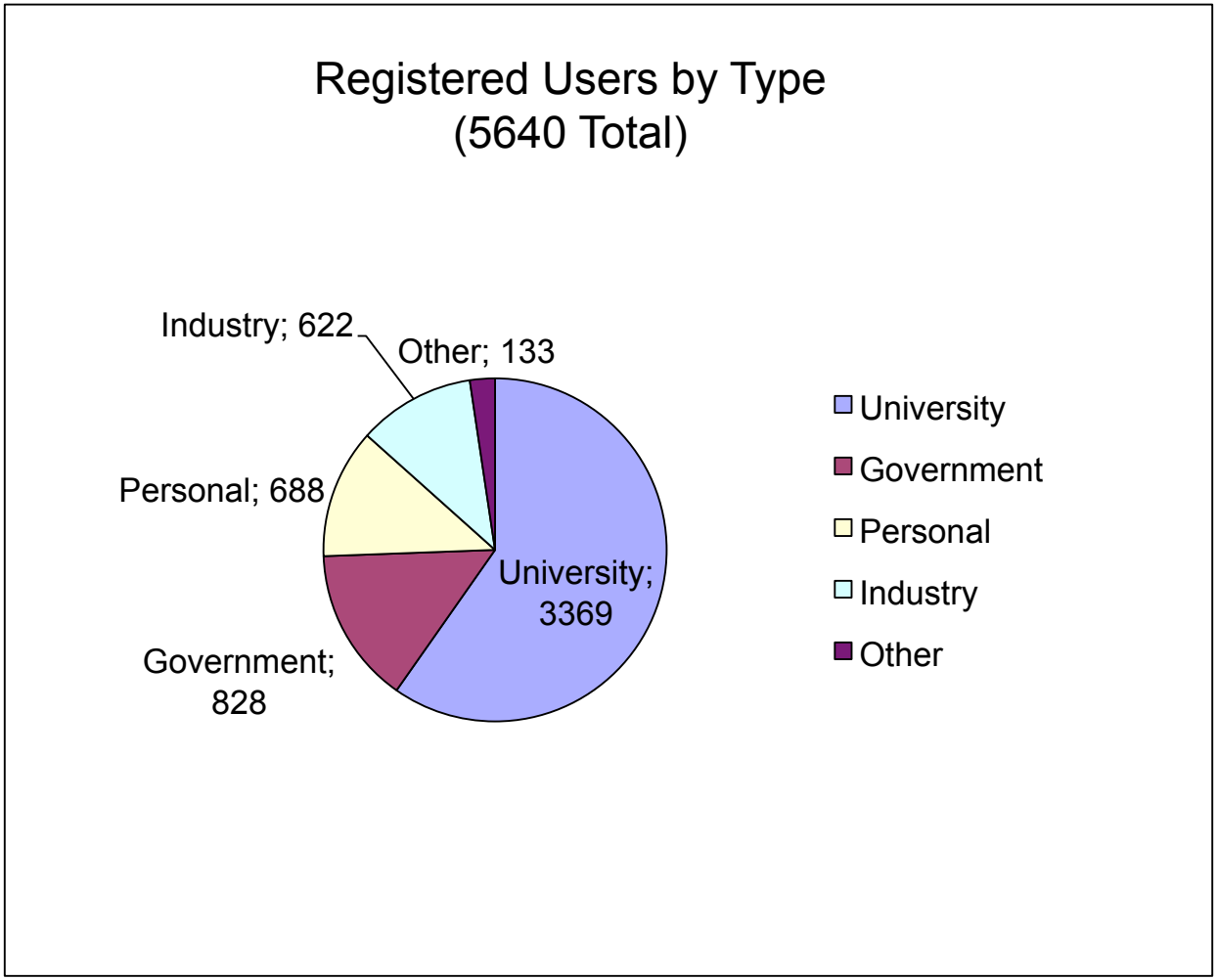


# Registered User by Region



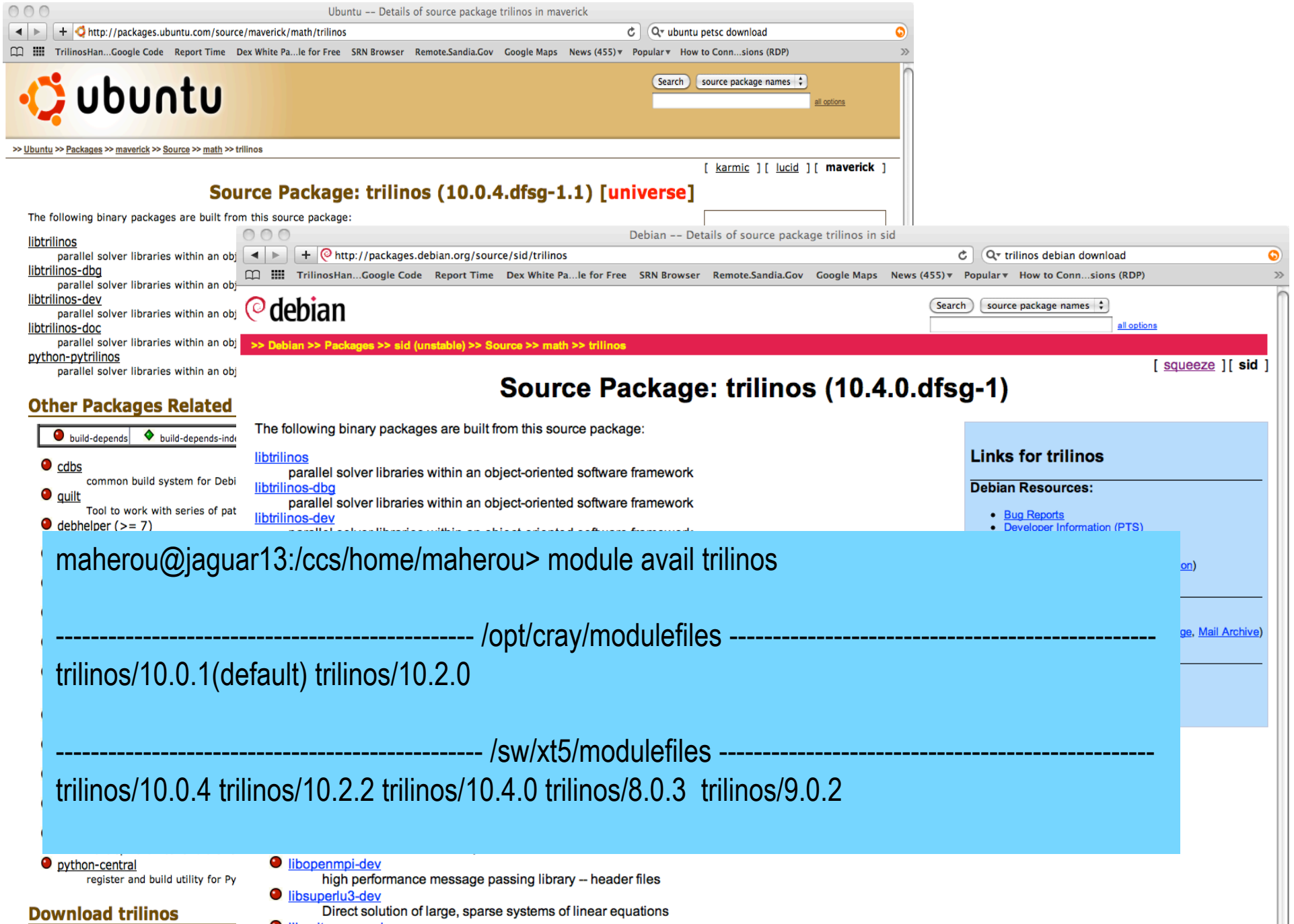

Sandia National Laboratories

# Registered Users by Type




Sandia National Laboratories

# Ubuntu/Debian: Other sources



# 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 |

# Three Design Points

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

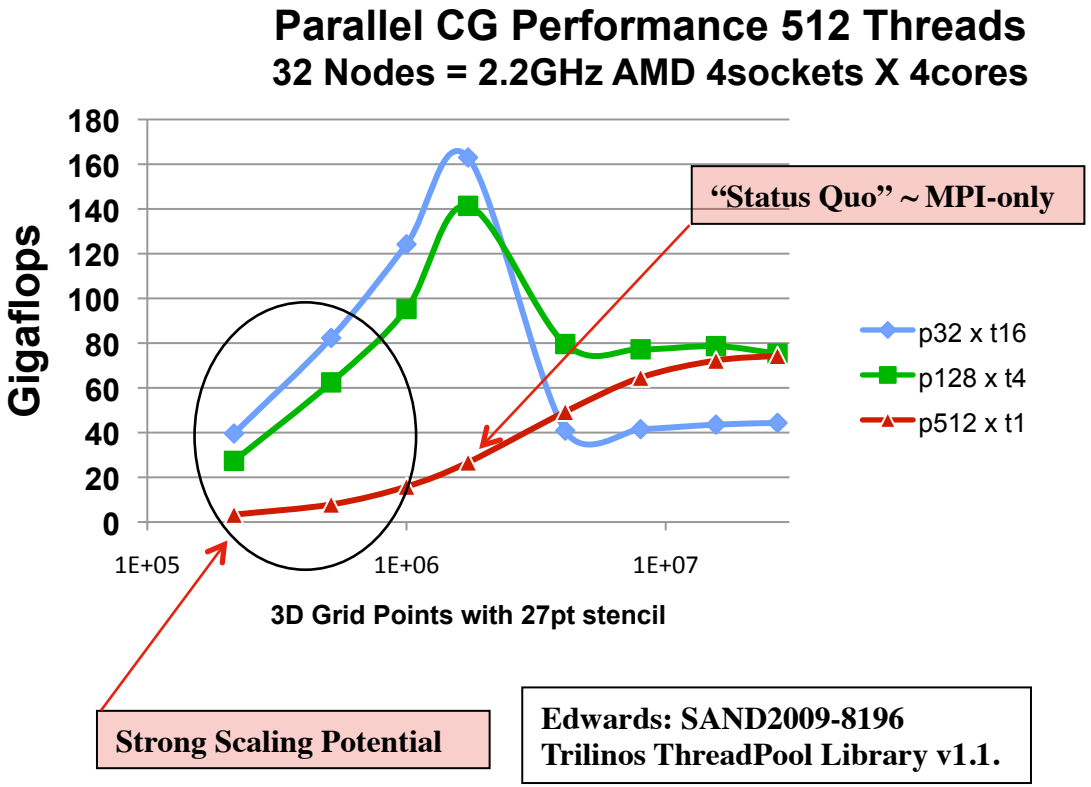
Sandia National Laboratories

# 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 paral
  - 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

Looking Forward to a New Age of Large-Scale Parallel Programming and the Demise of MPI

...hopes and dreams of an HPC educator

So What Would Life Be Like Without MPI?

$$F(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$


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

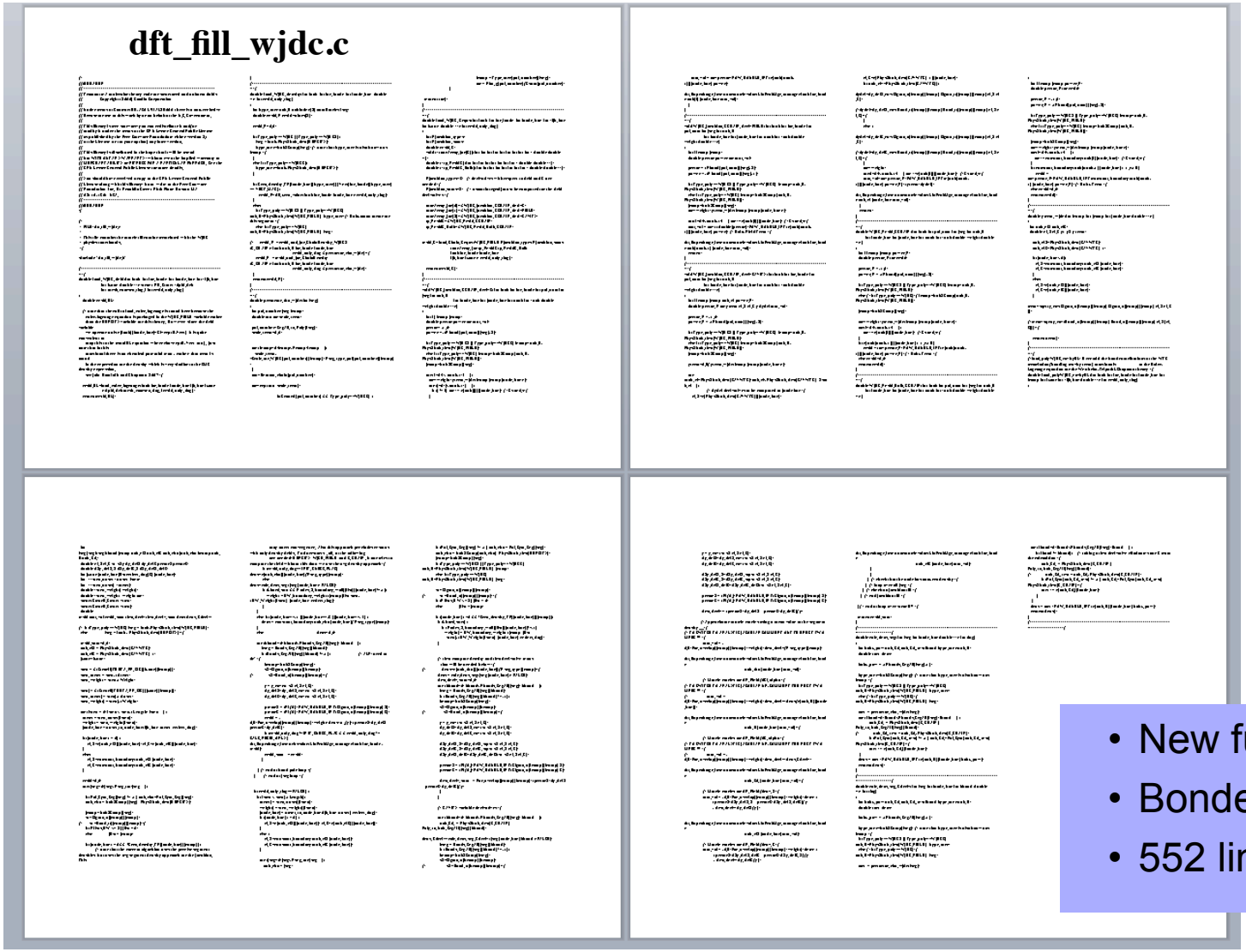
**Brad Chamberlain, Cray, 2000.**

# 3D Stencil in NAS MG



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

# *MPI Reality*

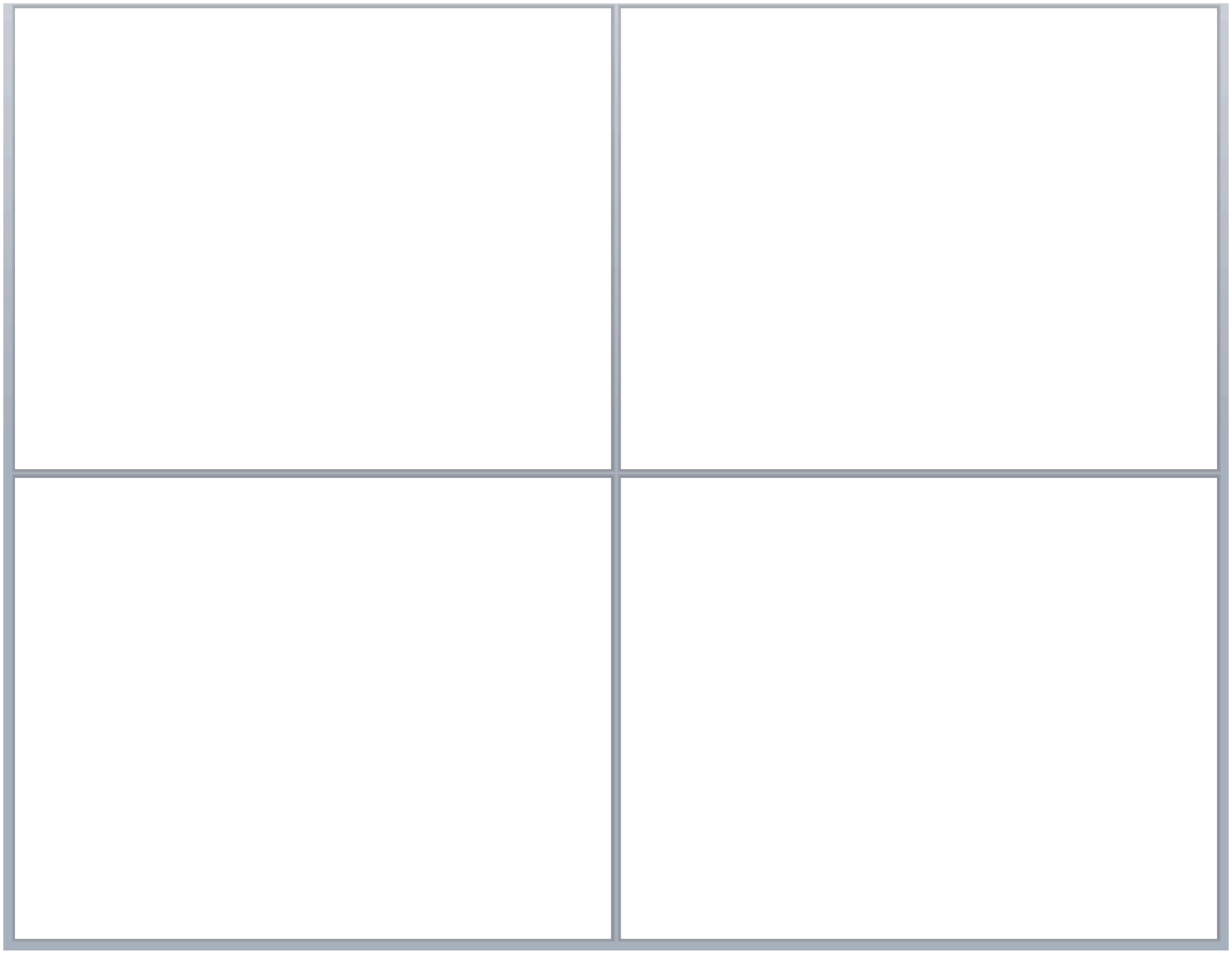



# Tramonto WJDC Functional

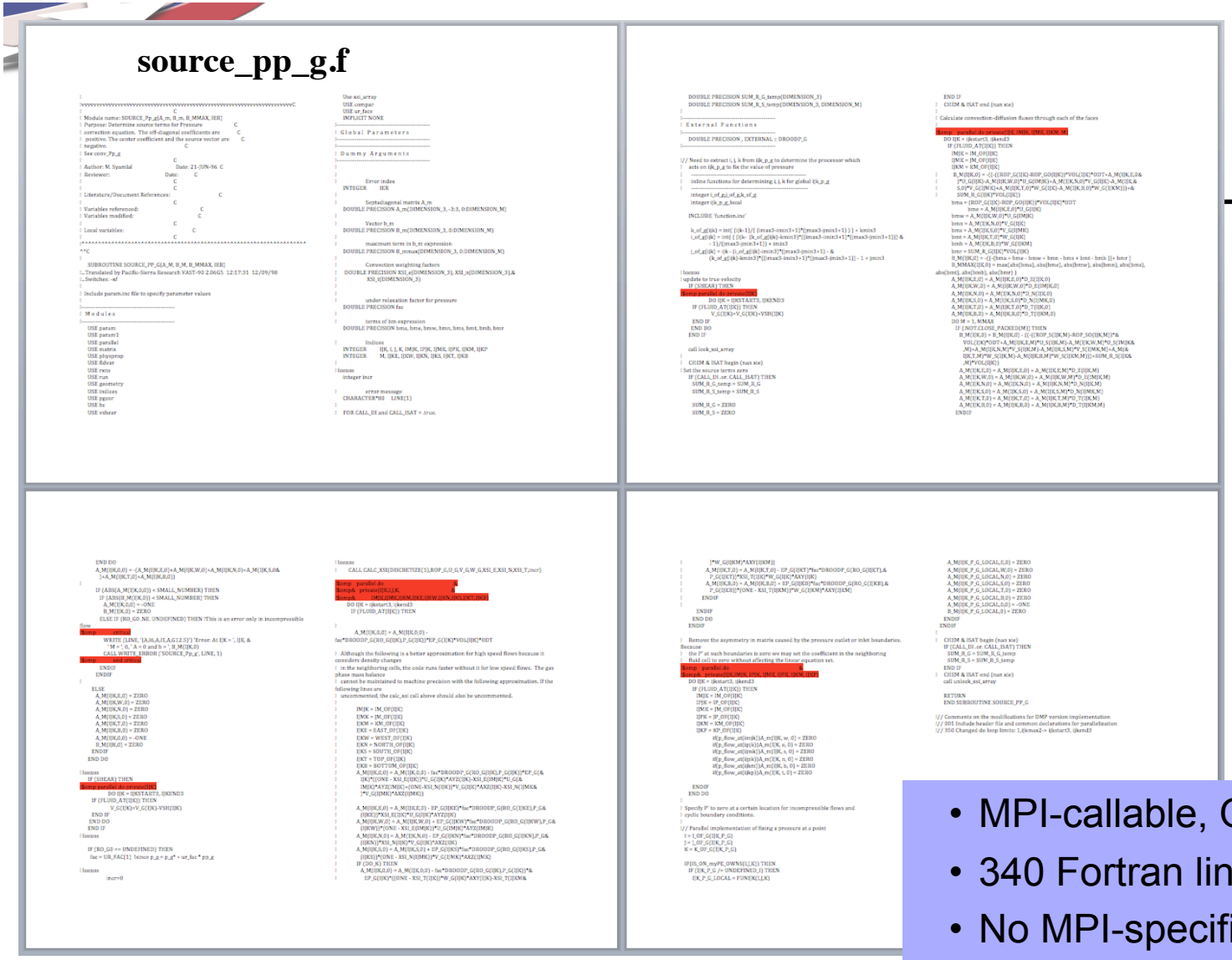
- New functional.
- Bonded systems.
- 552 lines C code.

**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).

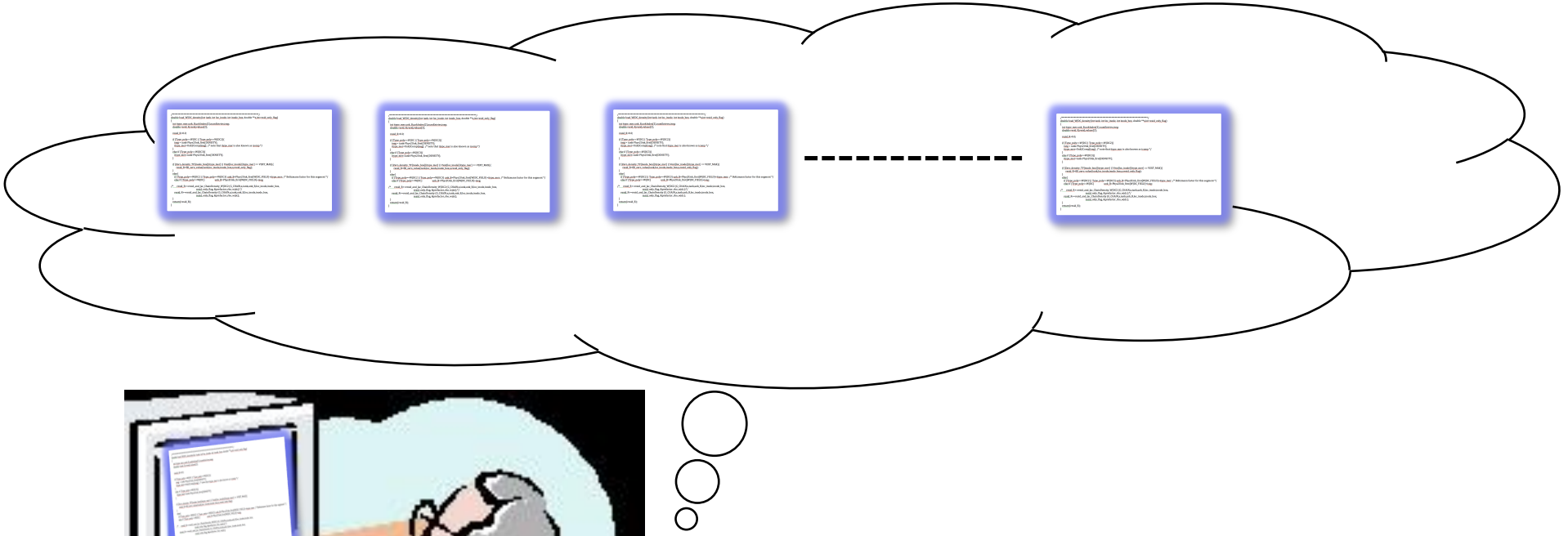
# Reasons for MPI Success?

- 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.

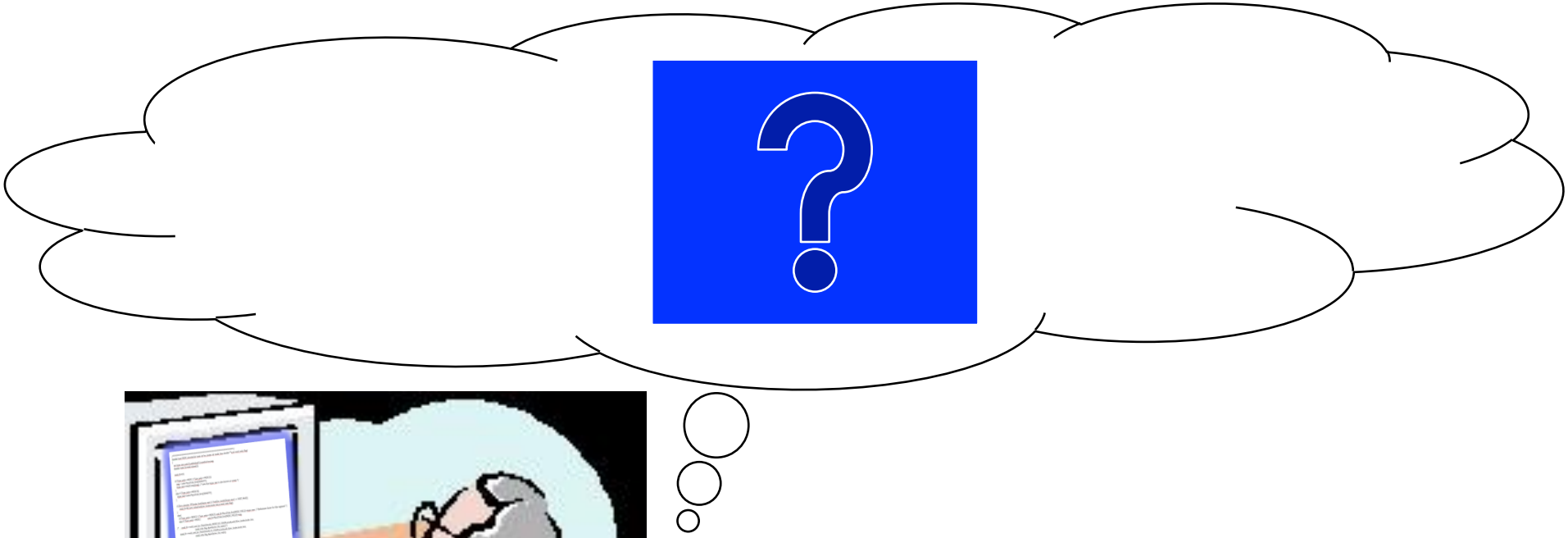
- Key goal for new parallel apps: Preserve this ratio

# Computational Domain Expert Writing MPI Code



Sandia National Laboratories

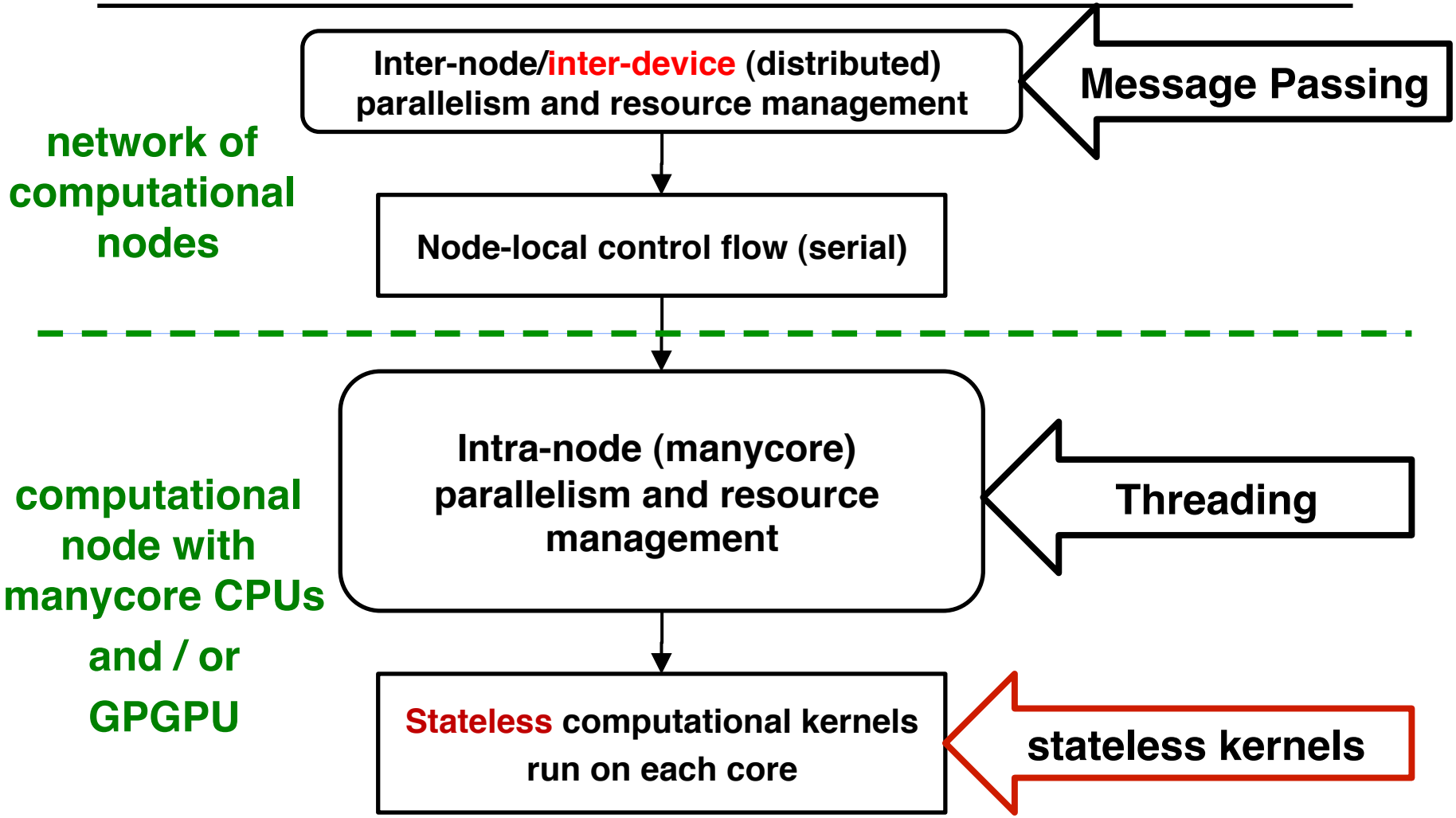
# Computational Domain Expert Writing Future Parallel Code



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

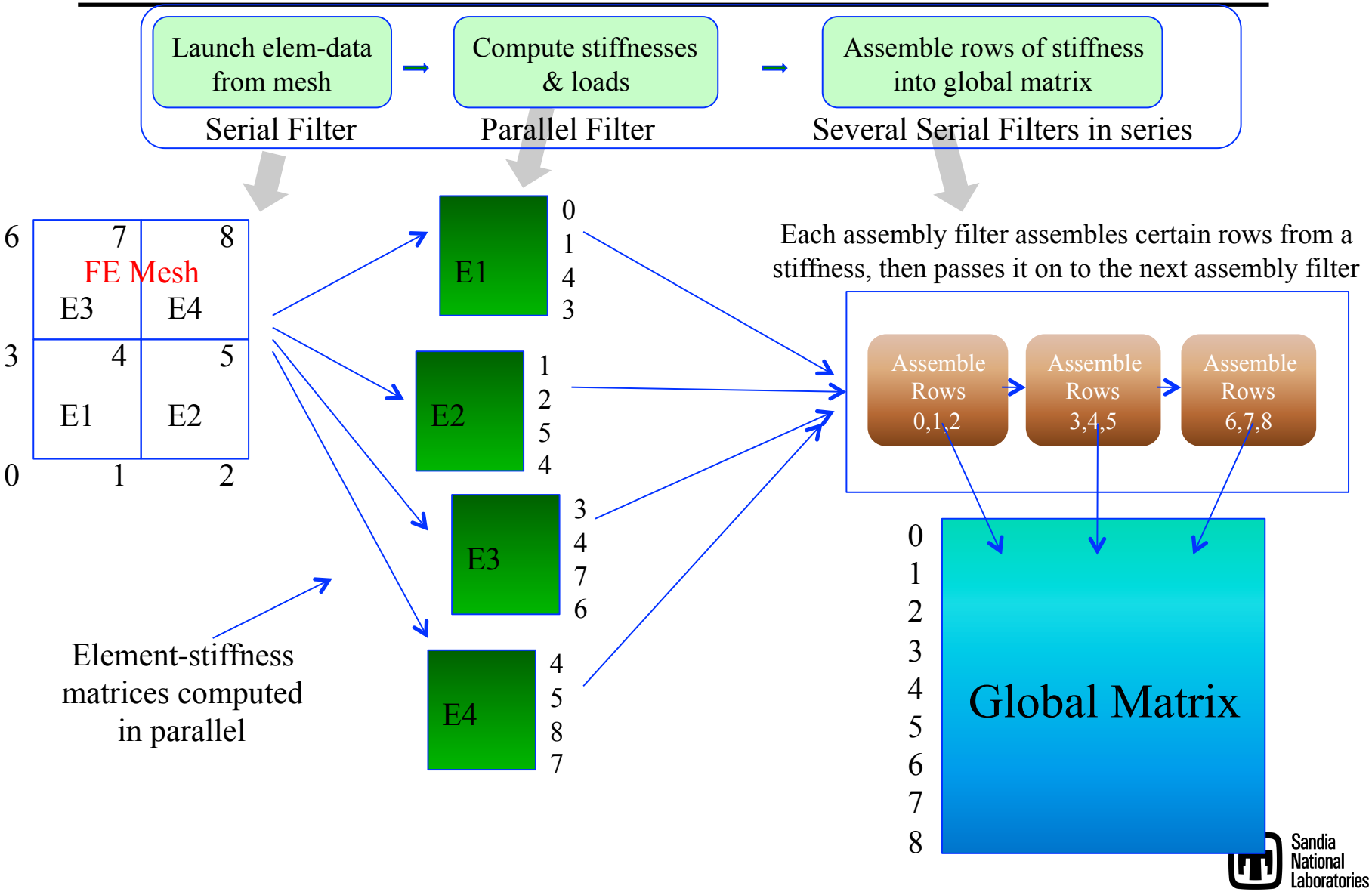
# Obvious Constructs/Concerns

- Parallel for:
  - No loop-carried dependence.
  - Rich loops.
  - Use of shared memory for temporal reuse, efficient device data transfers.
- Parallel reduce:
  - Couple with other computations.
  - Concern for reproducibility.

# Other construct: Pipeline

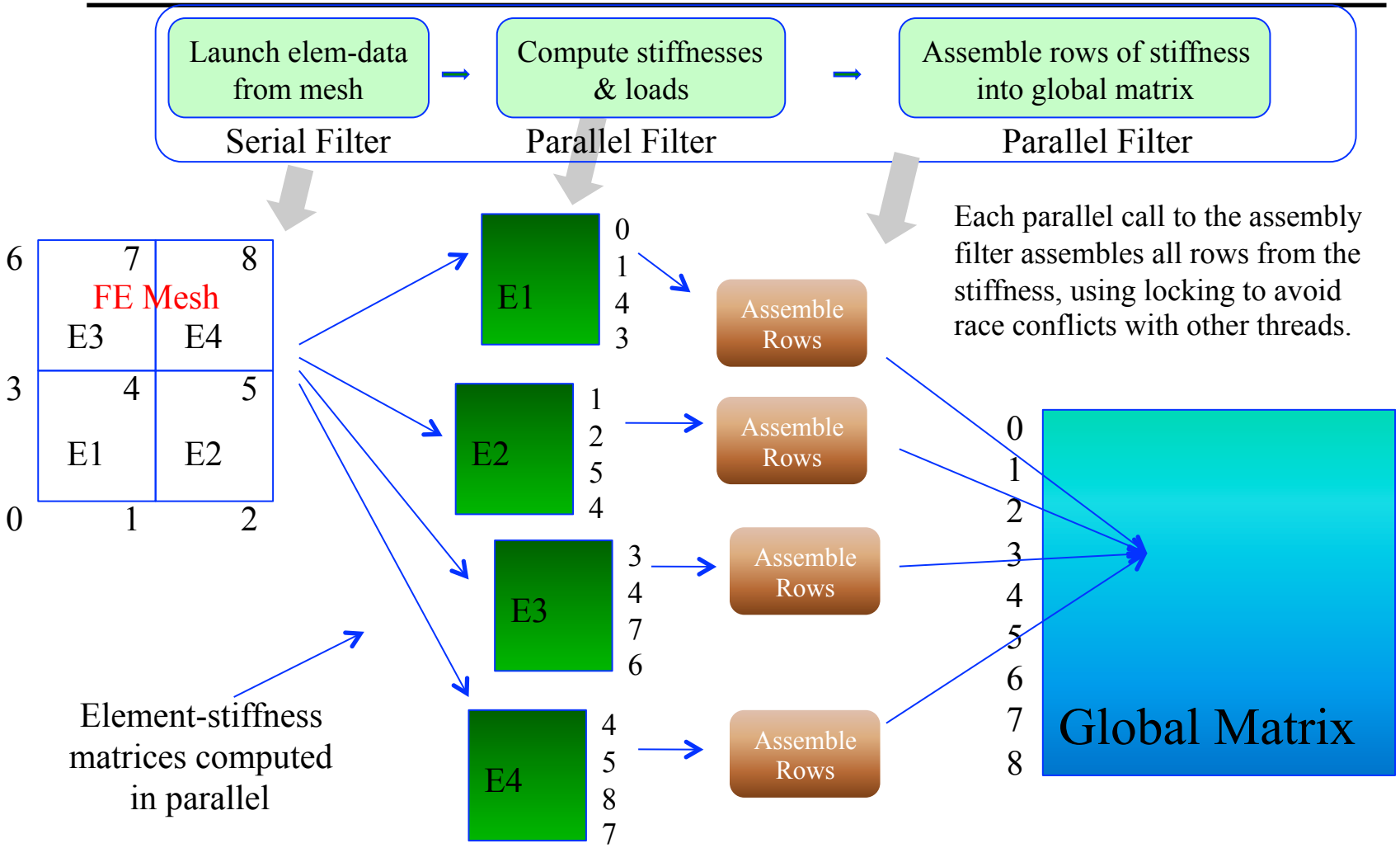
- Sequence of filters.
- Each filter is:
  - Sequential (grab element ID, enter global assembly) or
  - Parallel (fill element stiffness matrix).
- Filters executed in sequence.
- Programmer's concern:
  - Determine (conceptually): Can filter execute in parallel?
  - Write filter (serial code).
  - Register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

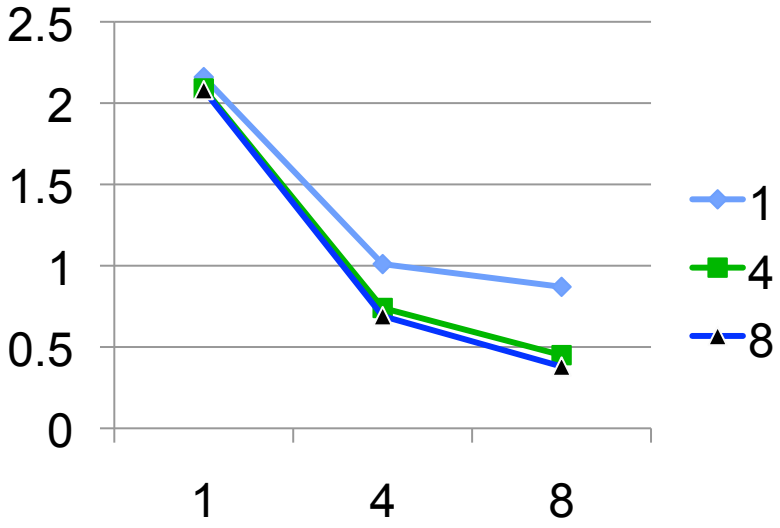
| Num-procs | Assembly-time Intel 11.1 | Assembly-time GCC 4.4.4 |
|---|---|---|
| 1 | 1.80s | 1.95s |
| 4 | 0.45s | 0.50s |
| 8 | 0.24s | 0.28s |

# 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.



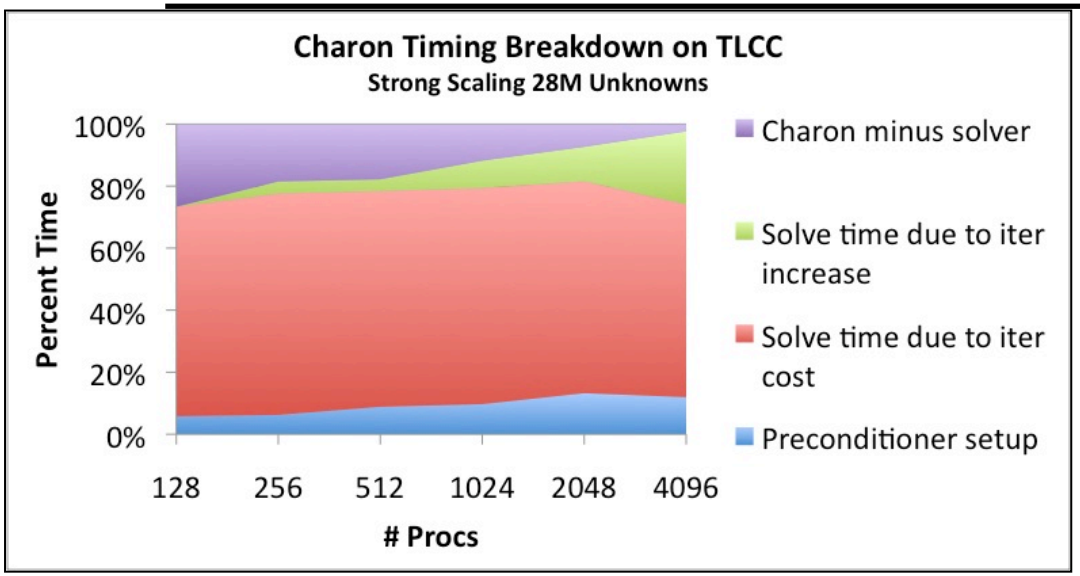

| Num-threads | Elem-group-size | Matrix-conflicts | Vector-conflicts | Assembly-time |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 2.16s |
| 1 | 4 | 0 | 0 | 2.09s |
| 1 | 8 | 0 | 0 | 2.08s |
| 4 | 1 | 95917 | 959 | 1.01s |
| 4 | 4 | 7938 | 25 | 0.74s |
| 4 | 8 | 3180 | 4 | 0.69s |
| 8 | 1 | 64536 | 1306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

Sandia National Laboratories

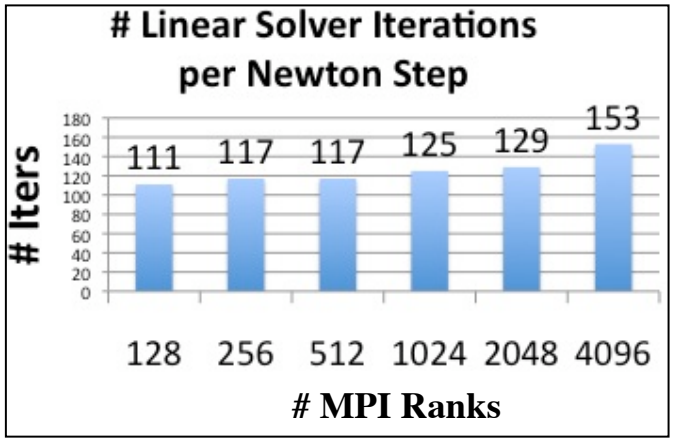
# Other construct: Thread team

- 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

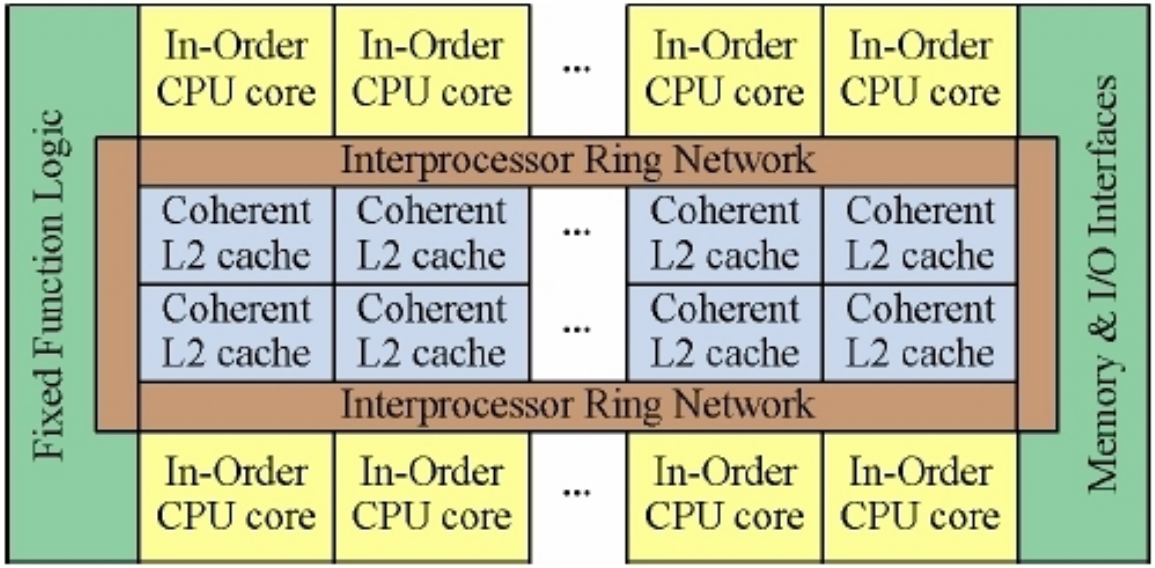
- Parallel for, reduce, pipeline:
  - Sufficient for vast majority of node level computation.
  - Supports:
    - Complex modeling expression.
    - Vanilla parallelism.
  - Must be “stencil-aware” for temporal locality.
- Thread team:
  - Complicated.
  - Requires true parallel algorithm knowledge.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

# *Portable Multi/Manycore Programming*
# *Trilinos/Kokkos Node API*

# Another Manycore architecture: Intel MIC

Knights Ferry:

- 32 x86 cores
  - 4-way hyperthreading
  - 128 threads total
- 512-bit vector unit
  - 16 floats, 8 doubles
- 1.20GHz
- PCI-E 2.0
- 2GB GDDR5 global mem
  - 8MB shared L2 cache
  - 64KB L1 Data, 64KB L1 Inst




Programming Env:

- OpenMP,
- TBB,
- Pthreads

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, awaiting my git push)*

# 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>`<br>`void Node::parallel_for(`<br>`  int beg, int end, WDP workdata);` | `template <class WDP>`<br>`WDP::ReductionType Node::parallel_reduce(`<br>`  int beg, int end, WDP workdata);` |
| Work-data pair (WDP) struct provides:<br>• loop body via `WDP::execute(i)` | Work-data pair (WDP) struct provides:<br>• reduction type `WDP::ReductionType`<br>• element generation via `WDP::generate(i)`<br>• reduction via `WDP::reduce(x,y)` |

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

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

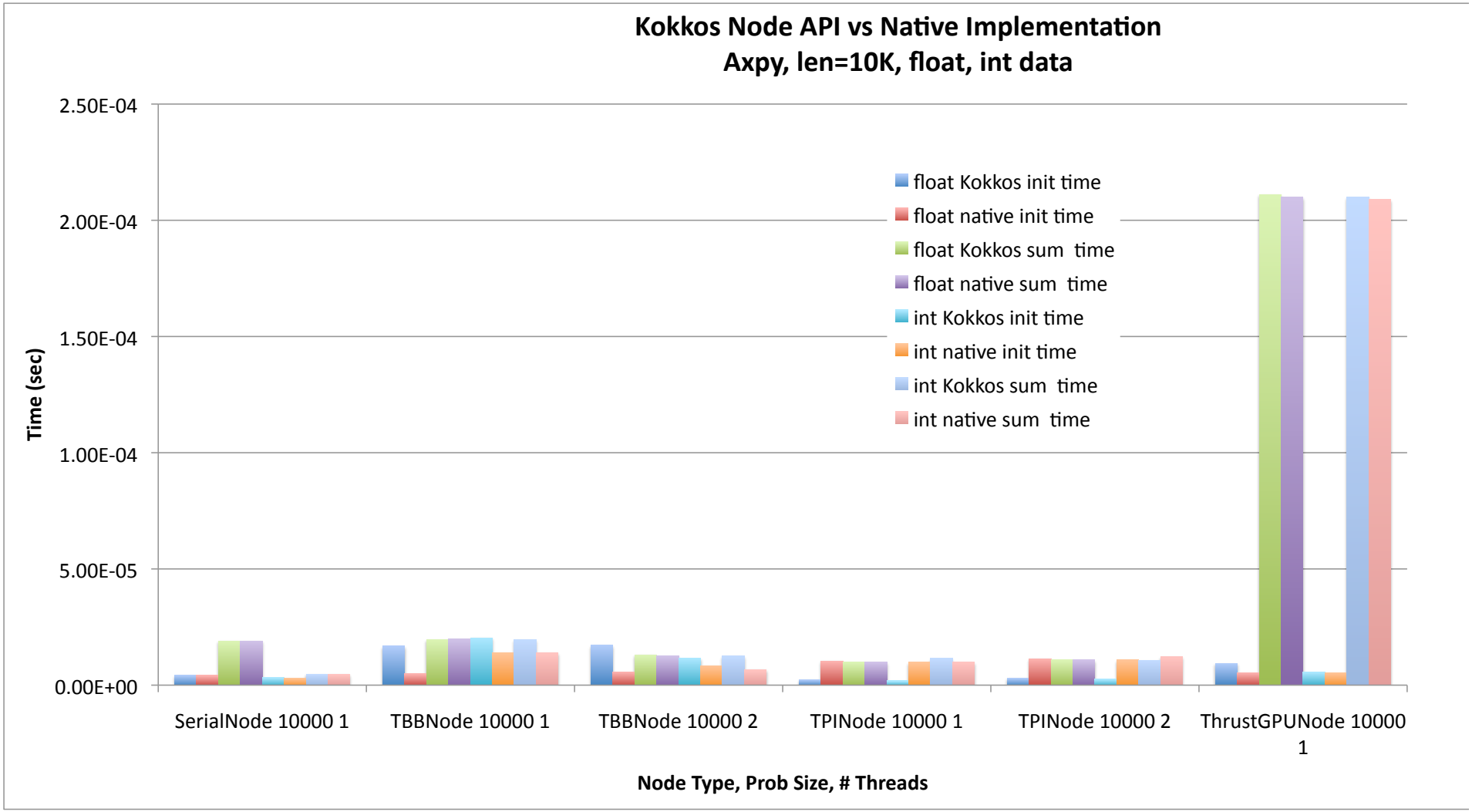
```
template <class WDP>
WDP::ReductionType
Node::parallel_reduce(int beg, int end,
                      WDP workdata    );
```

```
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]; }
};
```

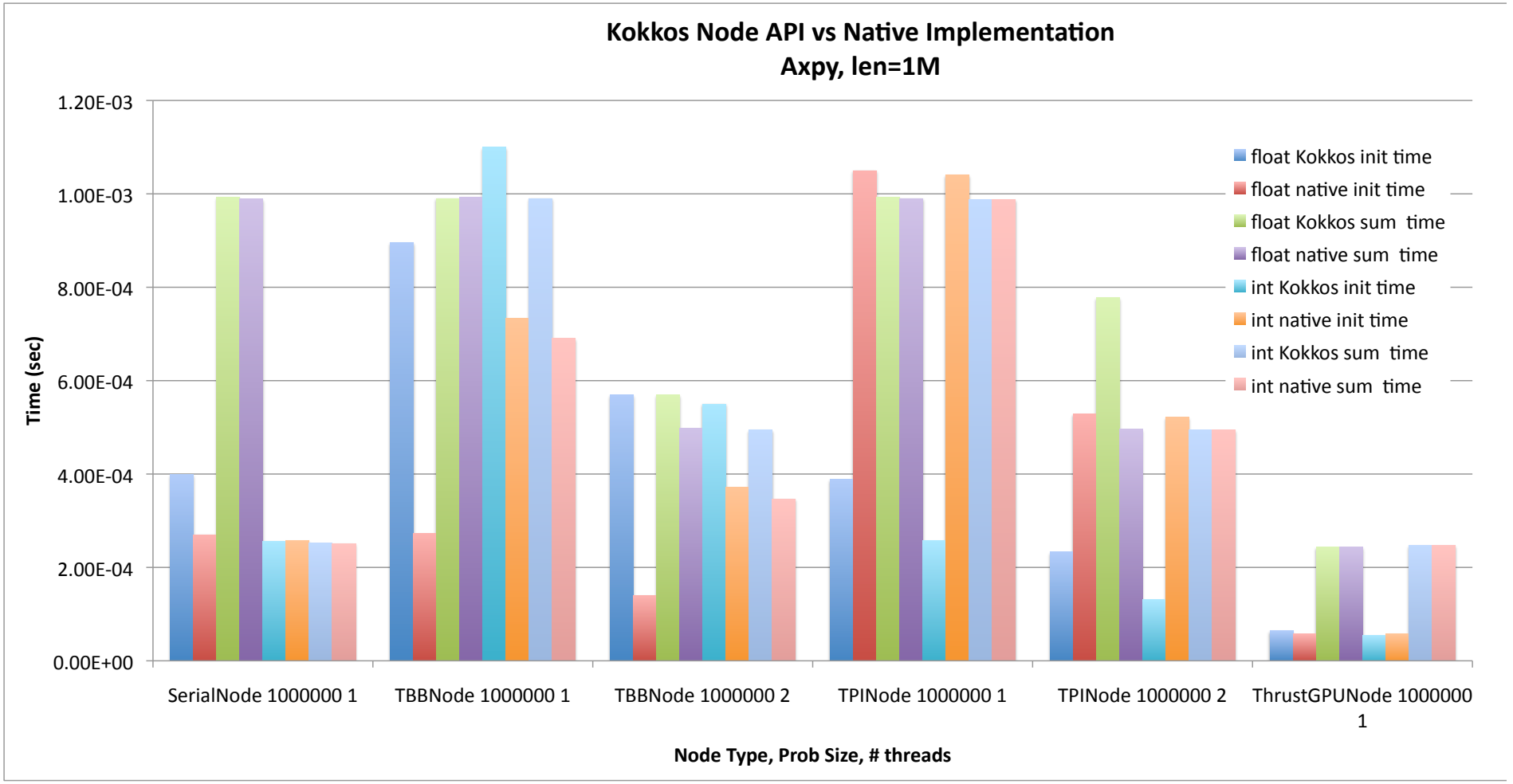
```
template <class T>
struct DotOp {
  typedef T ReductionType;
  const T * x, * y;
  T identity()       { return (T)0;      }
  T generate(int i)  { return x[i]*y[i]; }
  T reduce(T x, T y) { return x + y;     }
};
```

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

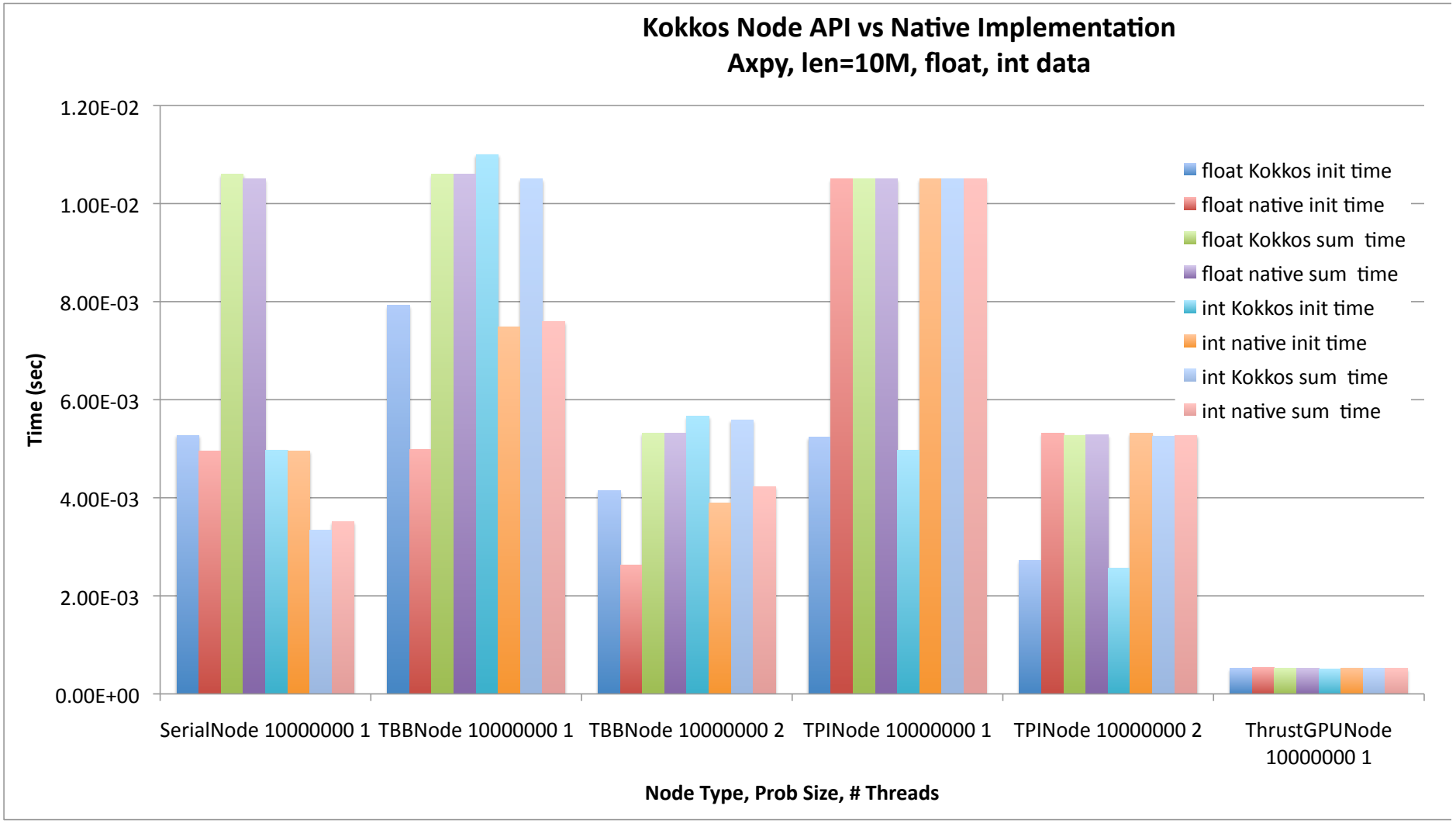
```
DotOp<float> op;
op.x = ...;  op.y = ...;
float dot;
dot = node.parallel_reduce< DotOp<float> >
                           (0, length, op);
```




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)**

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

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

**Example from IPOPT (Waechter)**

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

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

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>() ));
```

# 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

Sandia National Laboratories

# 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
```

Sandia National Laboratories

# *Hybrid CPU/GPU Computing*

# Writing and Launching Heterogeneous Jobs

- A node is a shared-memory domain.
- Multiple nodes are coupled via a communicator.
  - This requires launching multiple processes.
- In a heterogeneous cluster, this requires code written for multiple node types.
- It may be necessary to template large parts of the code and run the appropriate instantiation on each rank.
- For launching, two options are available:
  - Multiple single-node executables, complex dispatch
  - One diverse executable, early branch according to rank

# Tpetra::HybridPlatform

- Encapsulate main in a templated class method:

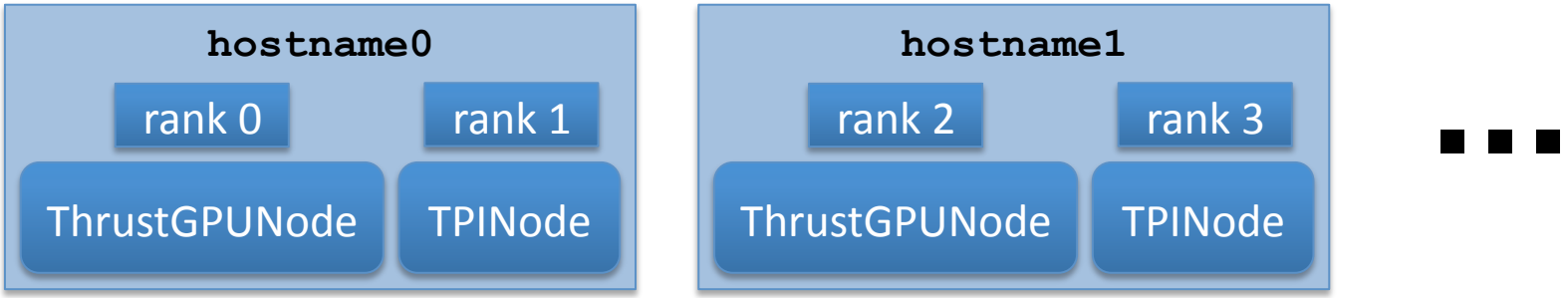
```
template <class Node>
class myMainRoutine {
  static void run(ParameterList &runParams,
                  const RCP<const Comm<int> > &comm,
                  const RCP<Node> &node)
  {
    // do something interesting
  }
};
```

- `HybridPlatform` maps the communicator rank to the Node type, instantiates a node and the run routine:

```
int main(...) {
   Comm<int>     comm         = ...
   ParameterList machine_file = ...
   // instantiate appropriate node and myMainRoutine
   Tpetra::HybridPlatform platform( comm , machine_file );
   platform.runUserCode< myMainRoutine >();
   return 0;
}
```

# HybridPlatform Machine File

| round-robin assignment | interval assignment | explicit assignment | default |
|---|---|---|---|
| %M=N | [M,N] | =N | default |




```
<ParameterList>
  <ParameterList name="%2=0">
    <Parameter name="NodeType"        type="string" value="Kokkos::ThrustGPUNode"/>
    <Parameter name="Verbose"         type="int"    value="1"/>
    <Parameter name="Device Number"   type="int"    value="0"/>
    <Parameter name="Node Weight"     type="int"    value="4"/>
  </ParameterList>
  <ParameterList name="%2=1">
    <Parameter name="NodeType"        type="string" value="Kokkos::TPINode"/>
    <Parameter name="Verbose"         type="int"    value="1"/>
    <Parameter name="Num Threads"     type="int"    value="15"/>
    <Parameter name="Node Weight"     type="int"    value="15"/>
  </ParameterList>
</ParameterList>
```

# HybridPlatformTest Output

```
[tpetra/example/HybridPlatform] mpirun -np 4 ./Tpetra_HybridPlatformTest.exe
                                              --machine-file=machines/G+15.xml

Every proc machine parameters from: machines/G+15.xml

Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens31 and rank 0!
Running test with Node == Kokkos::ThrustGPUNode on rank 0/4
ThrustGPUNode attached to device #0 "Tesla C1060", of compute capability 1.3

Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens31 and rank 1!
Running test with Node == Kokkos::TPINode on rank 1/4

Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens10 and rank 2!
Running test with Node == Kokkos::ThrustGPUNode on rank 2/4
TPINode initializing with numThreads == 15
ThrustGPUNode attached to device #0 "Tesla C1060", of compute capability 1.3

Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens10 and rank 3!
Running test with Node == Kokkos::TPINode on rank 3/4
TPINode initializing with numThreads == 15

...
```

See **HybridPlatformAnasazi.cpp** and **HybridPlatformBelos.cpp** for more fun!

# *Programming Today for Tomorrow's Machines*



Sandia National Laboratories

# 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?

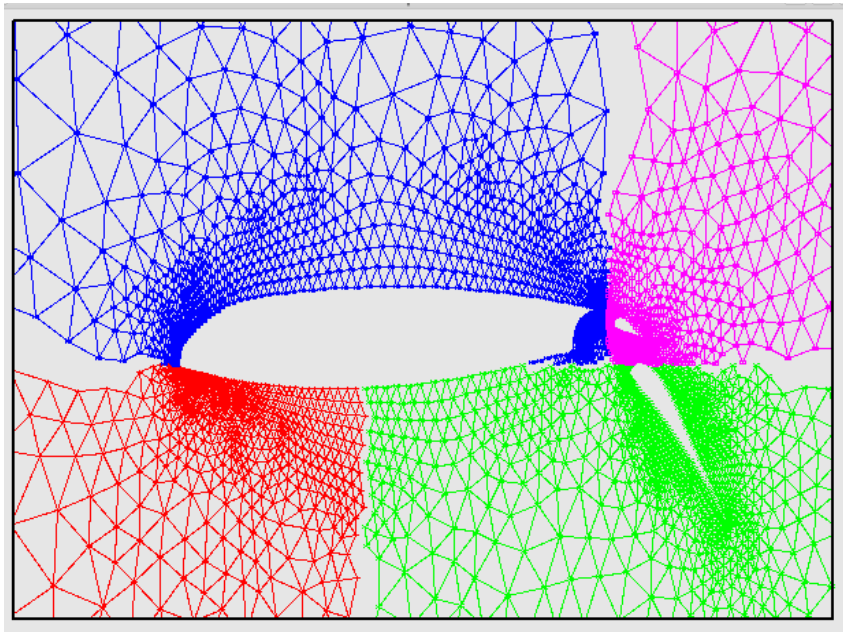
Sandia National Laboratories

# 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.

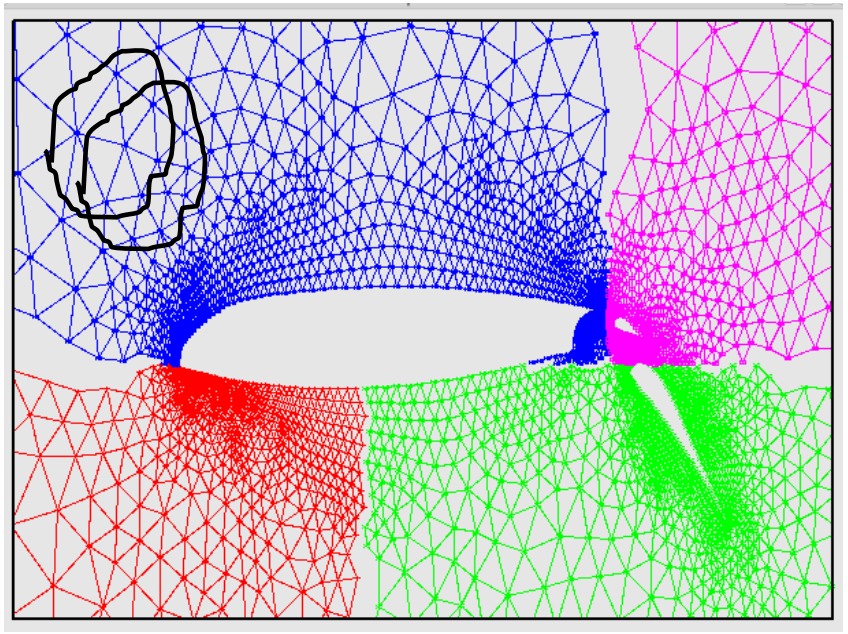


# FE/FV/FD Parallel Programming Tomorrow

```
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).



# *Resilient Algorithms*

# 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: 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

**M. Heroux, M. Hoemmen**

# *Software Development and Delivery*

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

# Compile-time Polymorphism

## Templates and Sanity upon a shifting foundation

Software delivery:

- Essential Activity

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.

# Solver Software Stack


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

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

**Sensitivities (Automatic Differentiation: Sacado)**

# Solver Software Stack


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

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

Sensitivities (Automatic Differentiation: Sacado)

# 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.
- Migration of Fortran apps to manycore will be painful.

Sandia National Laboratories

# Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
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.
8. GPUs are a harbinger of CPU things to come.
9. Fortran Developers are in trouble in a manycore world.
10. Global SIMT is sufficient parallelism for scientific computing.