Emerging Architectures and UQ: Implications and Opportunities

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.

Sandia National Laboratories

Outline

1. Brief Introduction to Trilinos (Why I think the way I do).
2. Why you should care about parallelism (if you don't already).
3. Why SPMD (think MPI) is successful.
4. Most future programmers won't need to write parallel code.
5. Extended precision is not too expensive to be useful.
6. Resilience will be built into algorithms.
7. A solution with error bars complements architecture trends.

Trilinos Background & Motivation

Sandia National Laboratories

Trilinos Contributors

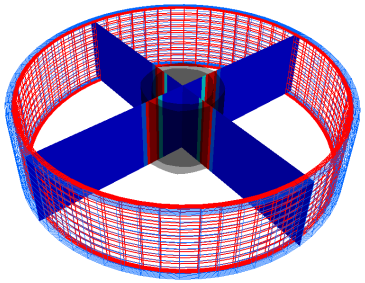
Current Contributors

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

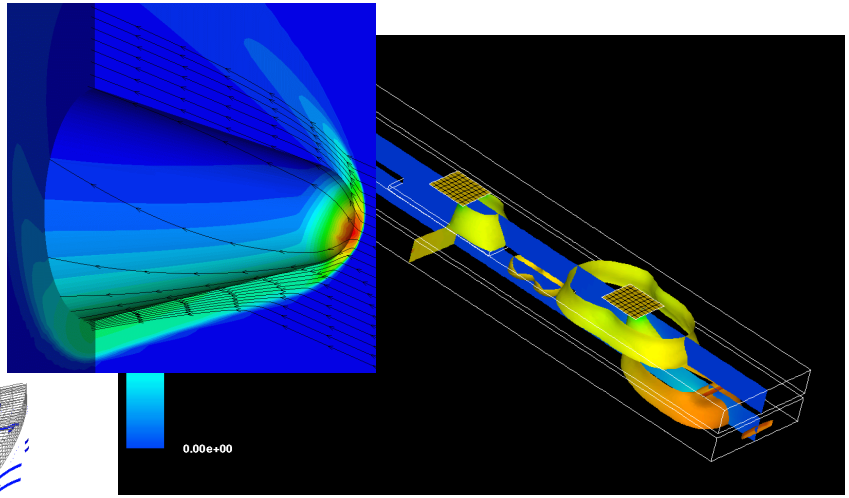
Past Contributors

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

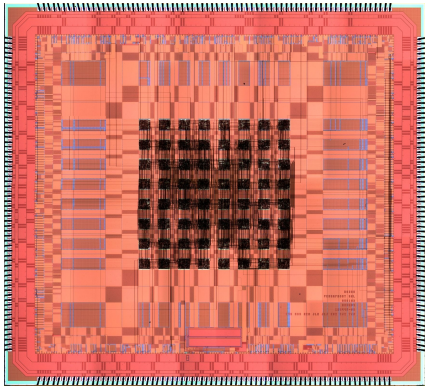
Target Problems: PDES and more…

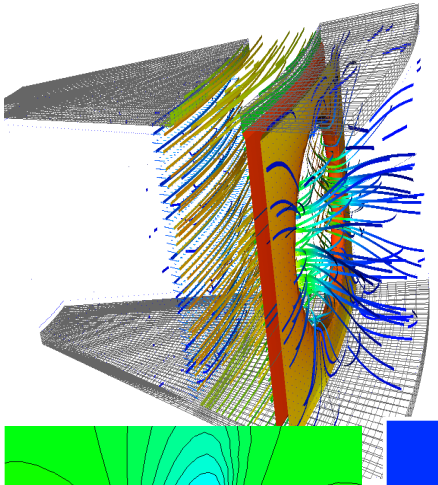


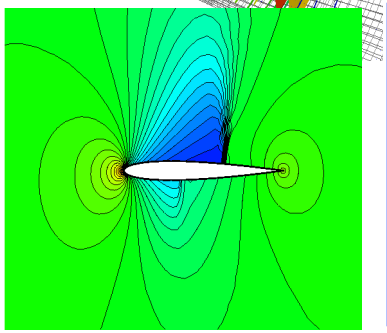
PDES

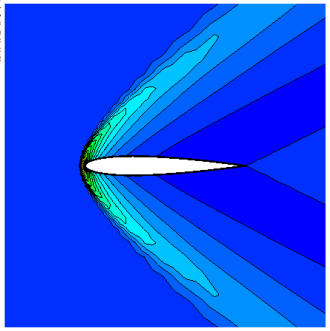


Circuits



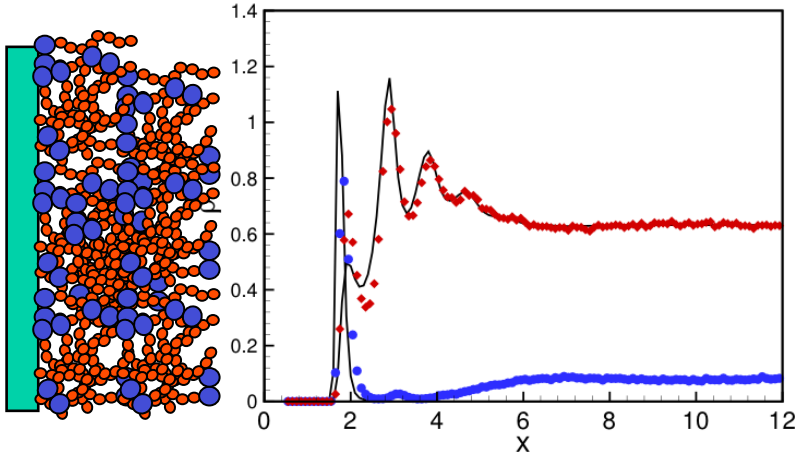






And More…

Inhomogeneous Fluids



Target Platforms: Any and All
(Now and in the Future)

- Desktop: Development and more…
- Capability machines:
  - Cielo (XE6), JaguarPF (XT5), Clusters
  - Titan (Hybrid CPU/GPU).
  - Multicore nodes.
- Parallel software environments:
  - MPI of course.
  - threads, vectors, CUDA OpenCL, …
  - Combinations of the above.
- User "skins":
  - C++/C, Python
  - Fortran.
  - Web.

Sandia National Laboratories

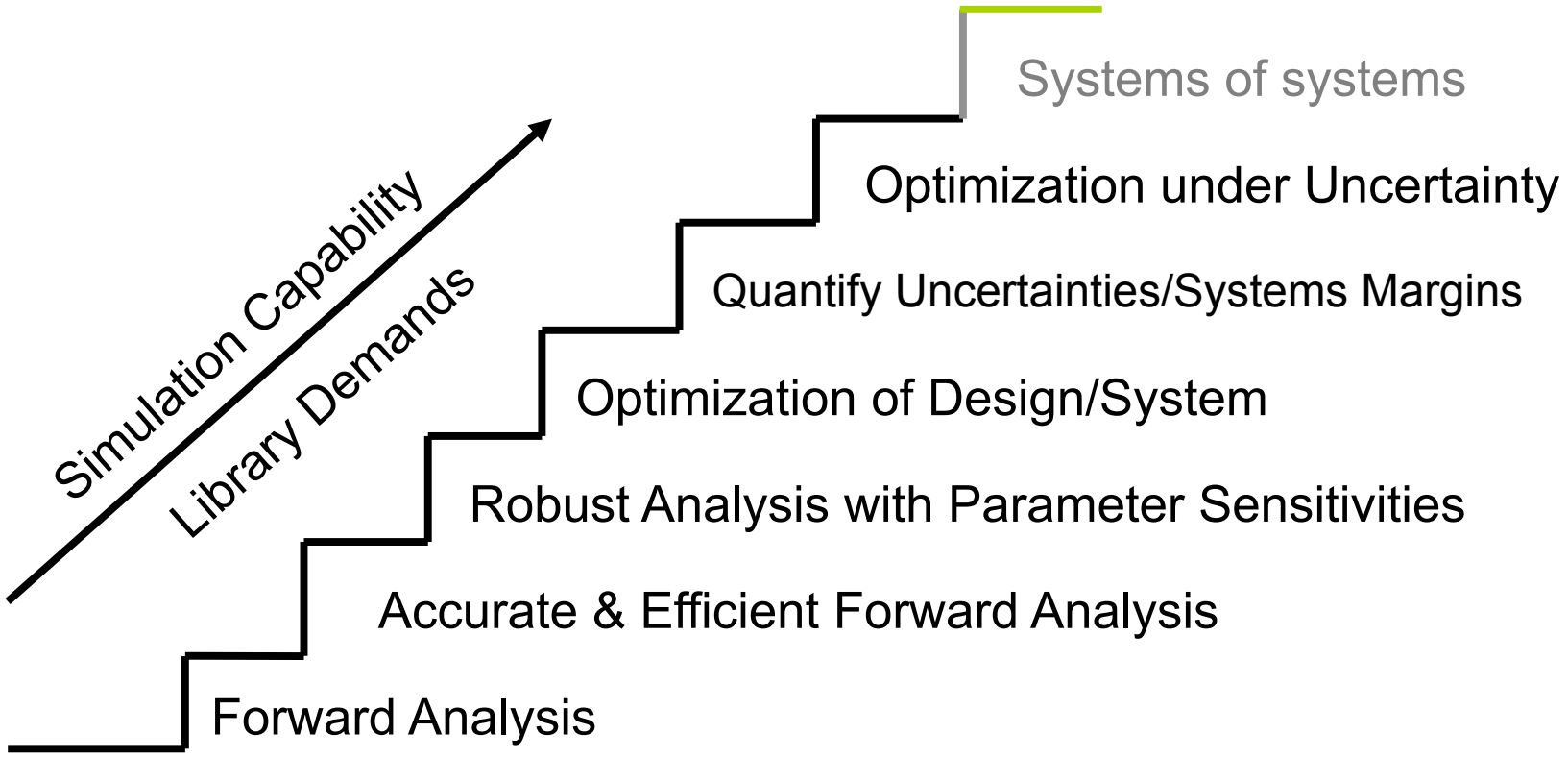
Evolving Trilinos Solution

- 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

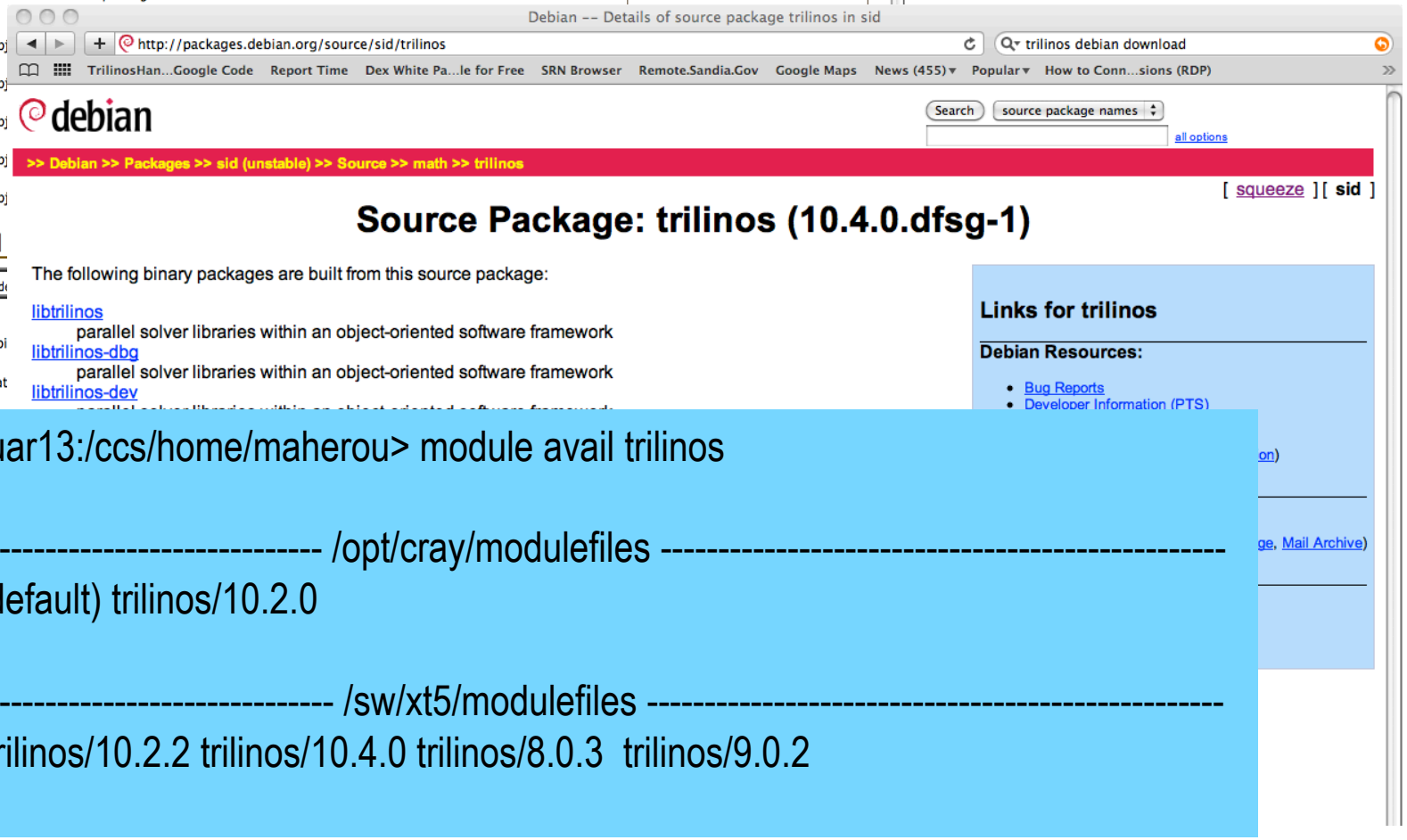
A Solutions Capability Maturity Model



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

Availability

- 6,200 Registered Users
- 20,400 Source Downloads.




```
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**

Emerging Architecture Algorithms & Programming Challenges

Sandia National Laboratories

Why care about parallelism: Riding the commodity curve

- In the past:
  - Clock speed, ILP.
  - Number of nodes.
- Now and future:
  - Core count, vector length.
  - Number of nodes (somewhat).
- But I have ensembles, all the parallelism I need.
  - Yes, but,
  - Fine grain parallelism (SIMD, SIMT) is part of the commodity curve formula.

Three Parallel Computing Design Points

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

Common Element

Goal: Make
Petascale = Terascale + more
Exascale = Petascale + more

Sandia National Laboratories

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

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

Factoring 1K to1B-Way Parallelism

- Why 1K to 1B?
  - Clock rate: O(1GHz) → O(10^9) ops/sec sequential
  - Terascale: 10^{12} ops/sec → O(10^3) simultaneous ops
    - 1K parallel intra-node.
  - Petascale: 10^{15} ops/sec → O(10^6) simultaneous ops
    - 1K-10K parallel intra-node.
    - 100-1K parallel inter-node.
  - Exascale: 10^{18} ops/sec → O(10^9) simultaneous ops
    - 1K-10K parallel intra-node.
    - 100K-1M parallel inter-node.
- Current nodes:
  - SPARC64™ VIIIfx: **128GF** (at 2.2GHz). "K" machine
  - NVIDIA Fermi: **500GF** (at 1.1GHz). Tianhe-1A.

Sandia National Laboratories

Data Movement: Locality

- Locality always important:
  - Caches: CPU
  - L1$ vs L2$ vs DRAM: Order of magnitude latency.
- Newer concern:
  - NUMA affinity.
  - Initial data placement important (unless FLOP rich).
  - Example:
    - 4-socket AMD with dual six-core per socket (48 cores).
    - BW of owner-compute: 120 GB/s.
    - BW of neighbor-compute: 30 GB/s.
- GPUs: Not so much a concern.

Memory Size

- Current “healthy” memory/core:
  - 512 MB/core (e.g. MD computations).
  - 2 GB/core (e.g. Implicit CFD).
- Future:
  - 512 MB/core “luxurious”.

Sandia National Laboratories

Resilience

- Individual component reliability:
  - Tuned for “acceptable” failure rate.
- Aggregate reliability:
  - Function of all components not failing.
  - May decline.
- Size of data sets may limit usage of standard checkpoint/restart.

Summary of Algorithms Challenge

- Realize node parallelism of O(1K-10K).
- Do so
  - Within a more complicated memory system and
  - With reduced relative memory capacity and
  - With decreasing reliability.

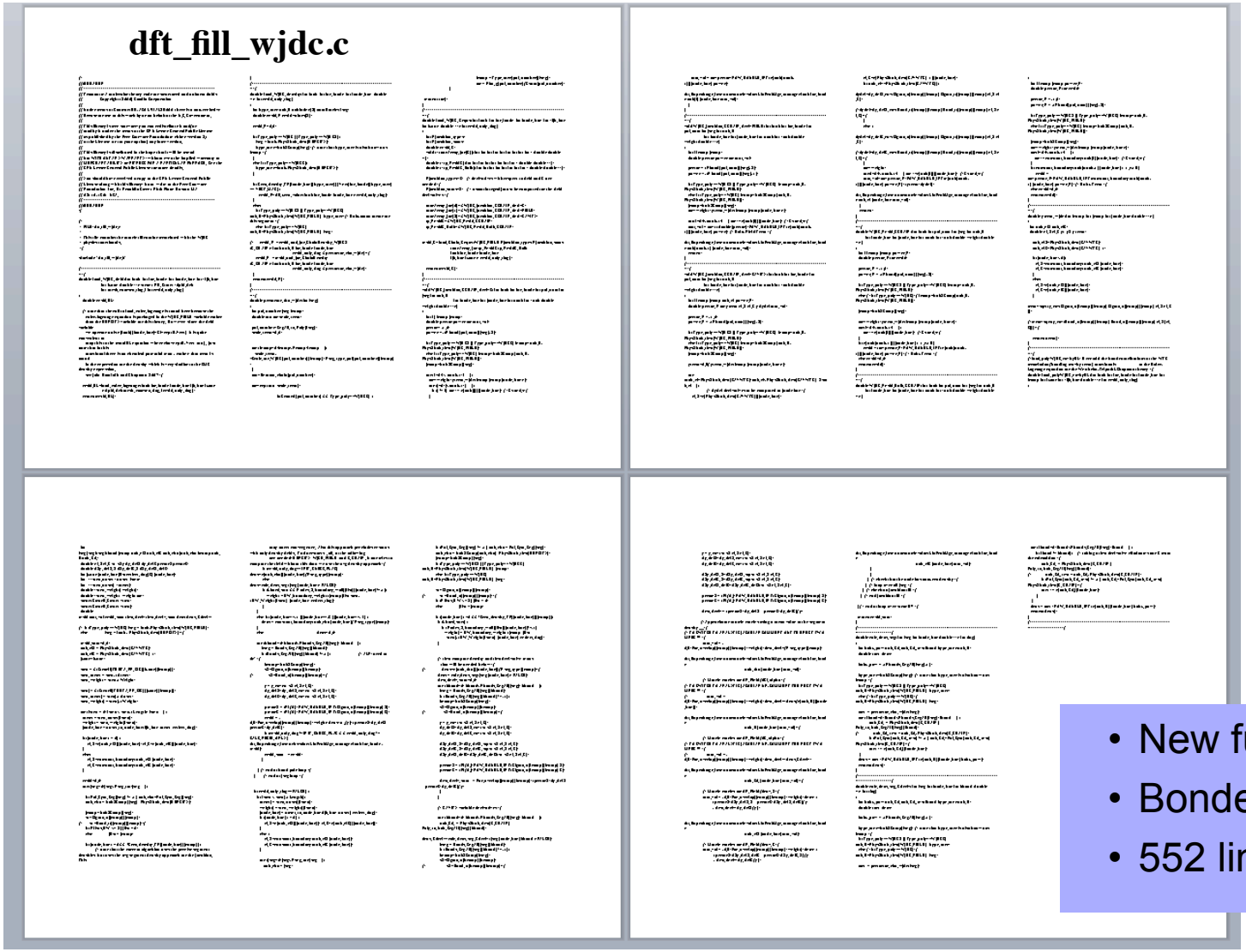
Designing for Trends

- Long-term success must include design for change.
- Algorithms we develop today must adapt to future changes.
- Lesson from Distributed Memory (SPMD):
  - What was the trend? Increasing processor count.
  - Domain decomposition algs matched trend.
    - Design algorithm for *p* domains.
    - Design software for expanded modeling within a domain.

New Trends and Responses

- Increasing data parallelism:
  - Design for vectorization and increasing vector lengths.
  - SIMT a bit more general, but fits under here.
- Increasing core count:
  - Expose task level parallelism.
  - Expresss task using DAG or similar constructs.
- Reduced memory size:
  - Express algorithms as multi-precision.
  - Compute data vs. store
- Memory architecture complexity:
  - Localize allocation/initialization.
  - Favor algorithms with higher compute/communication ratio.
- Resilience:
  - Distinguish what must be reliably computed.
  - Incorporate bit-state uncertainty into broader UQ contexts?

Observations and Strategies for Parallel Algorithms Design

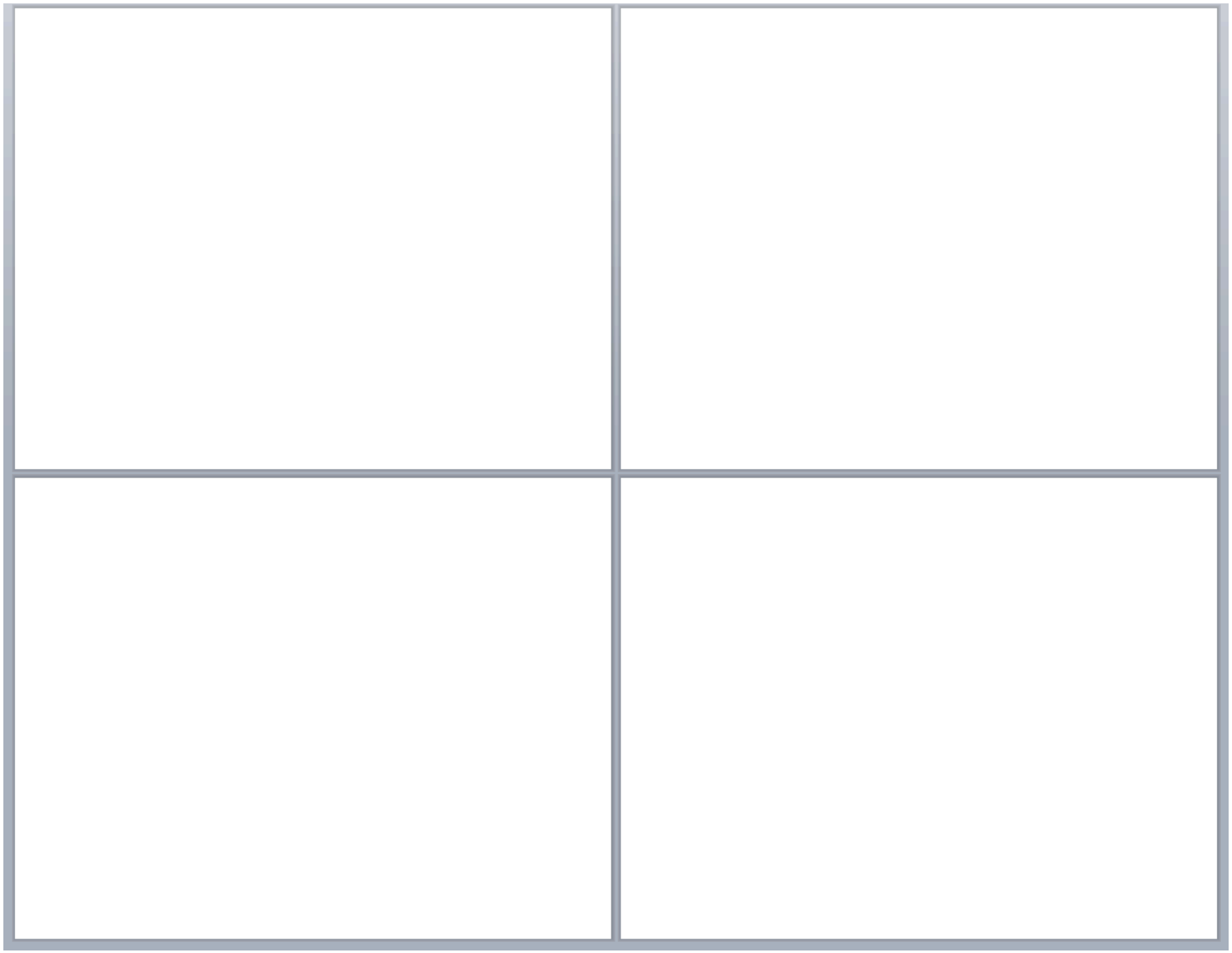


Tramonto WJDC Functional

- 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?

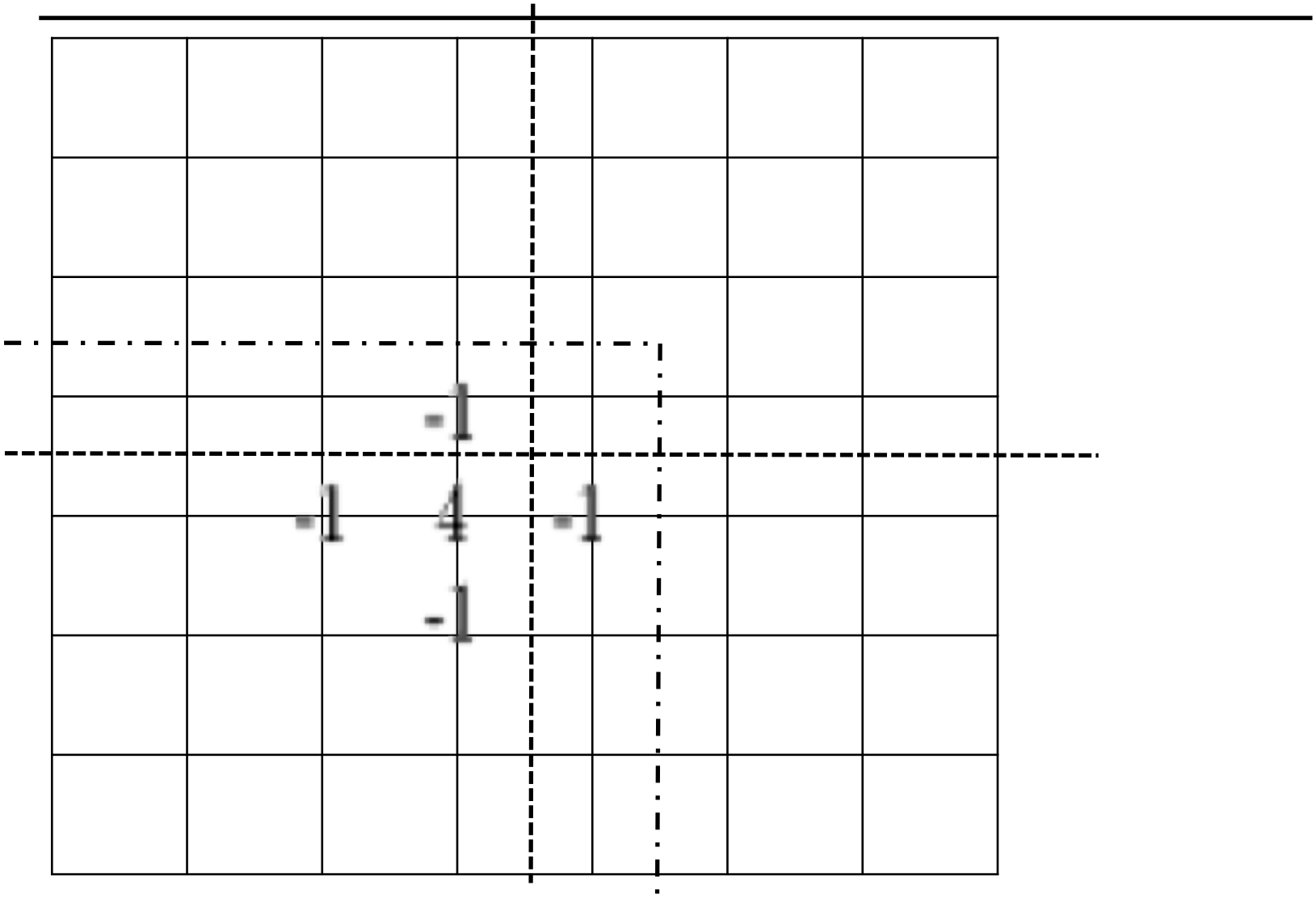


dft_fill_wjdc.c
MPI-specific
code

Single Program Multiple Data (SPMD) 101
Separation of Concerns: Parallelism vs. Modeling

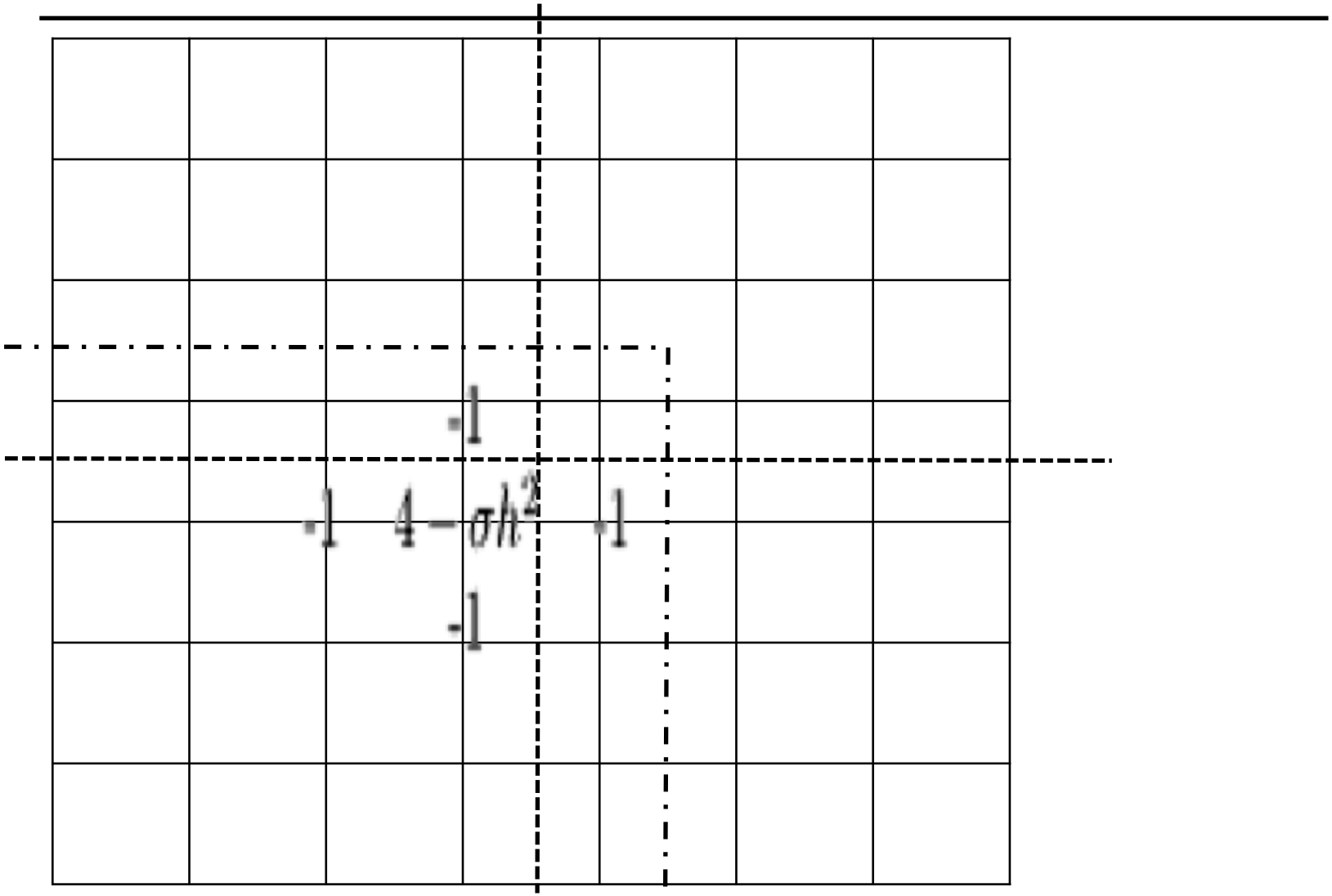
Sandia National Laboratories

2D PDE on Regular Grid (Standard Laplace)



Sandia National Laboratories

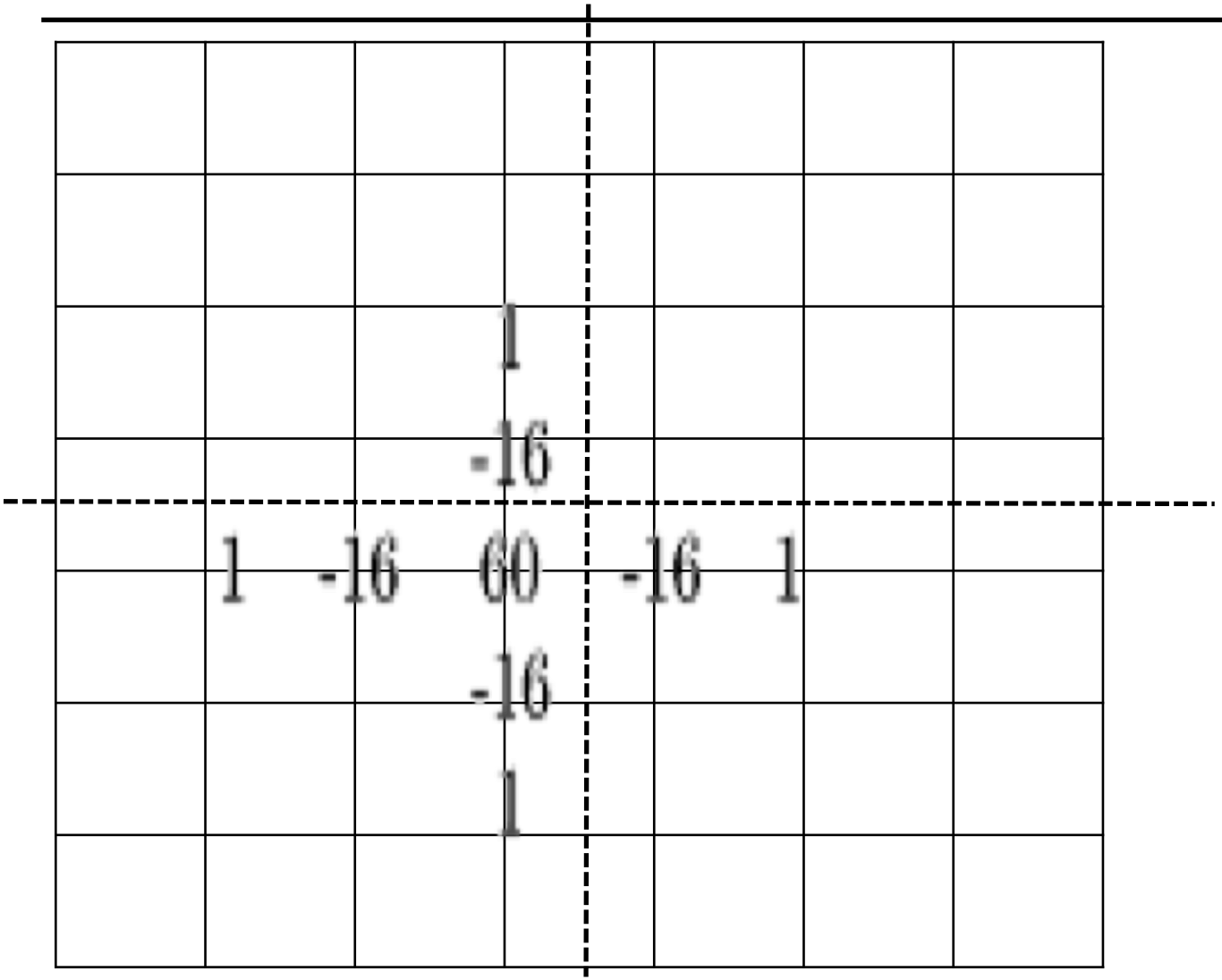
2D PDE on Regular Grid (Helmholtz)



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

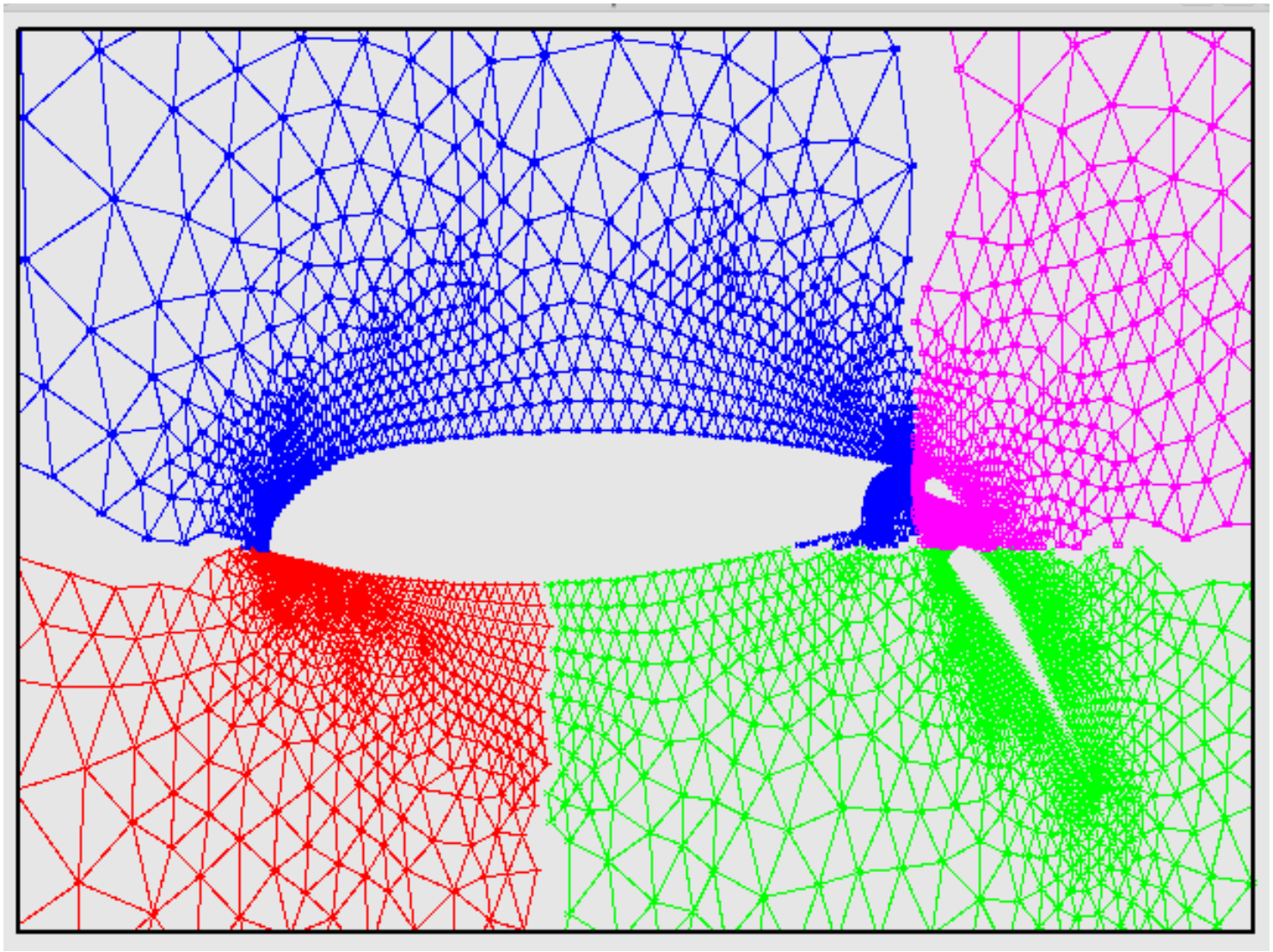
Sandia National Laboratories

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



Sandia National Laboratories

More General Mesh and Partitioning

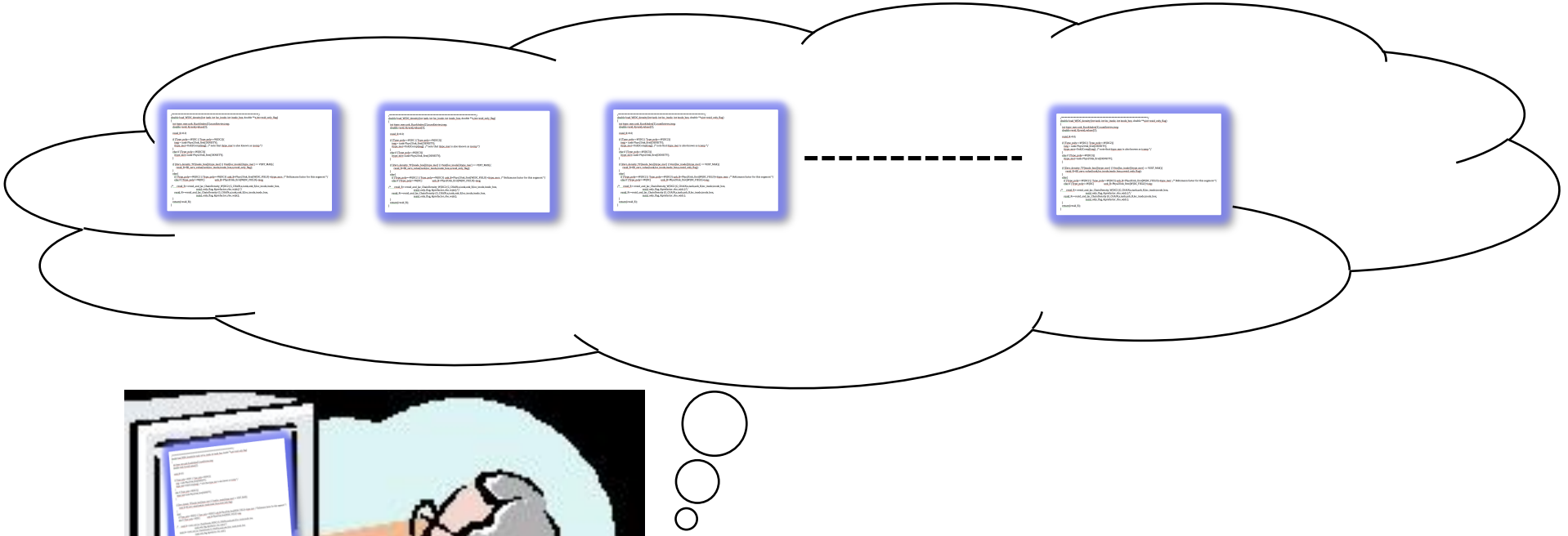


Sandia National Laboratories

SPMD Patterns for Domain Decomposition

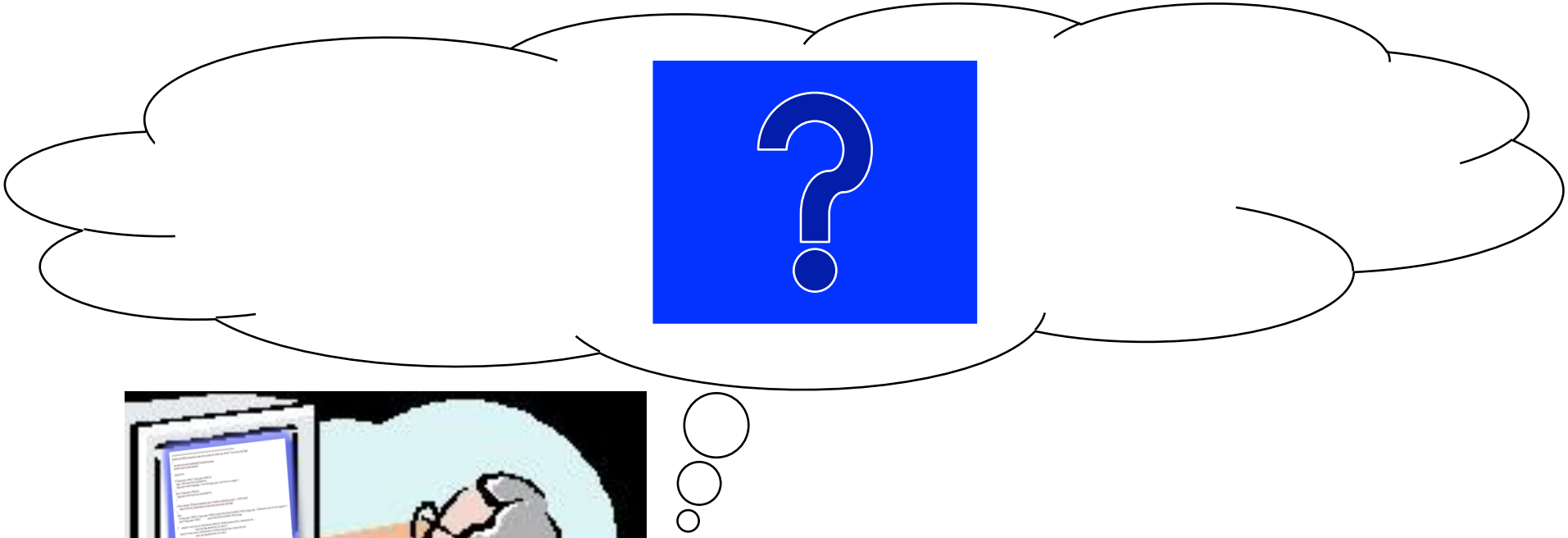
- Halo Exchange:
  - Conceptual.
  - Needed for any partitioning, halo layers.
  - MPI is simply portability layer.
  - Could be replace by PGAS, one-sided, …
- Collectives:
  - Dot products, norms.
- All other programming:
  - Sequential!!!

Computational Domain Expert Writing MPI Code



Sandia National Laboratories

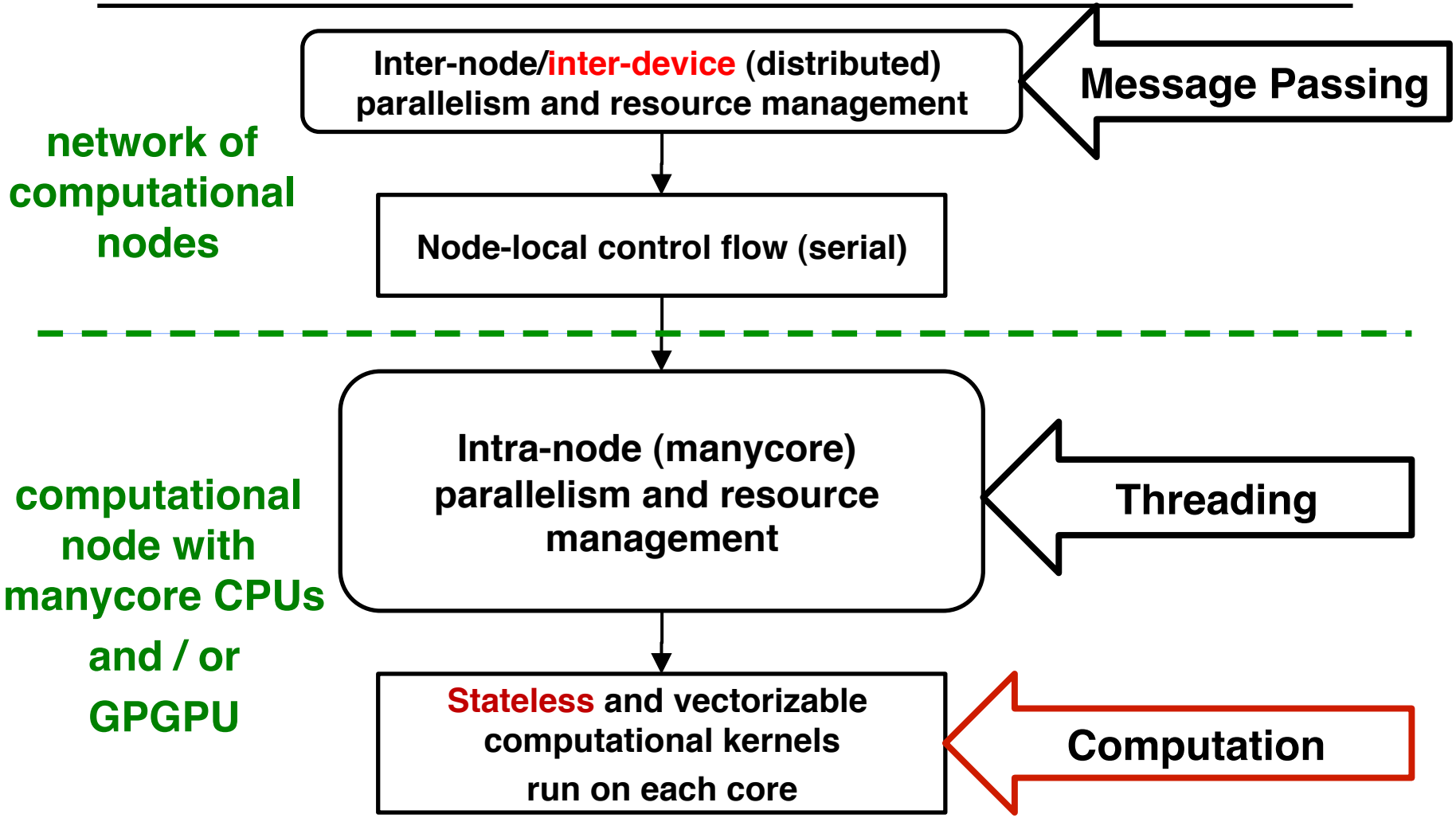
Computational Domain Expert Writing Future Parallel Code



Sandia National Laboratories

Evolving Parallel Programming Model

Parallel Programming Model: Multi-level/Multi-device



Domain Scientist's Parallel Palette

- MPI-only (SPMD) apps:
  - Single parallel construct.
  - Simultaneous execution.
  - Parallelism of even the messiest serial code.
- 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?

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.

Other construct: Thread team

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

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?

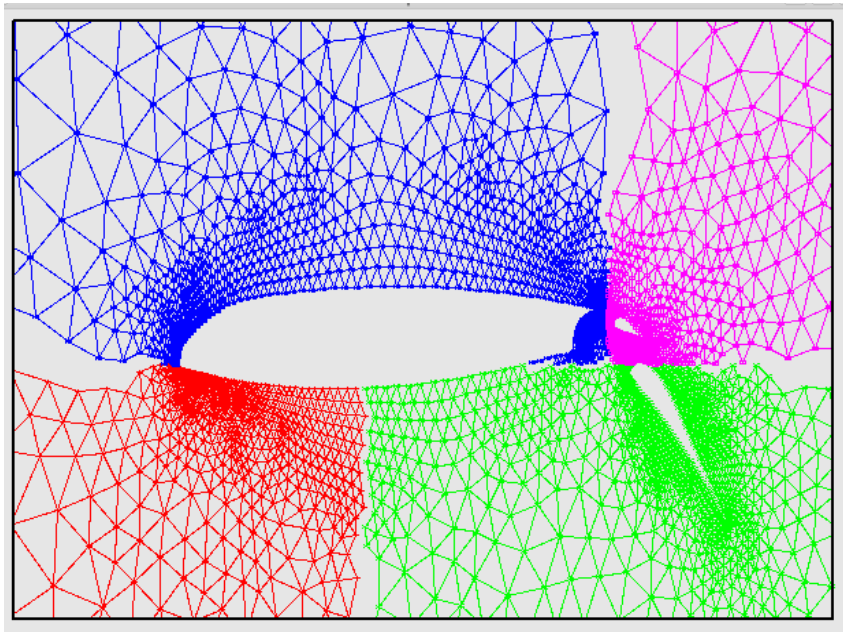
Sandia National Laboratories

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.

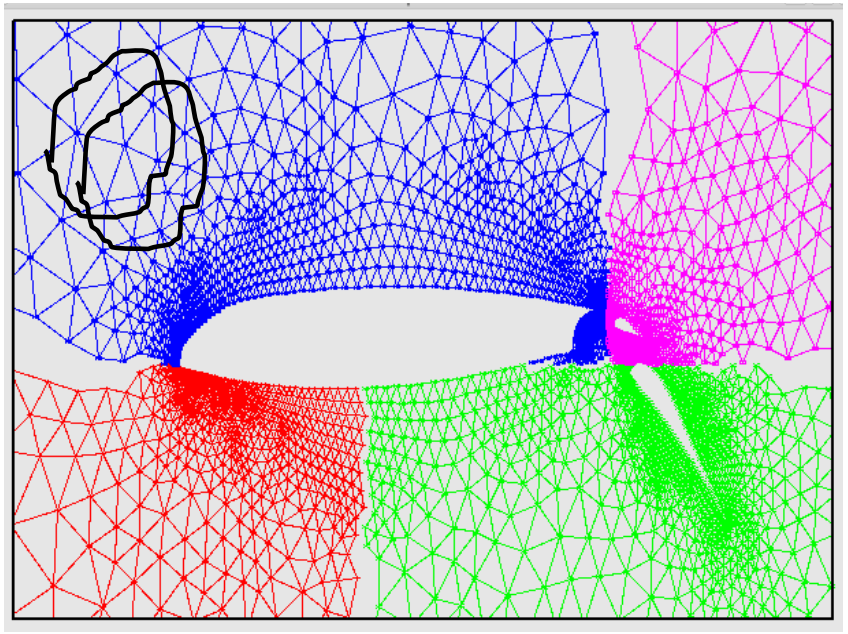


FE/FV/FD Parallel Programming Tomorrow

```
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).



Portable Multi/Manycore Programming

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

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

Work-data pair (WDP) struct provides: • loop body via `WDP::execute(i)`	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    );
```

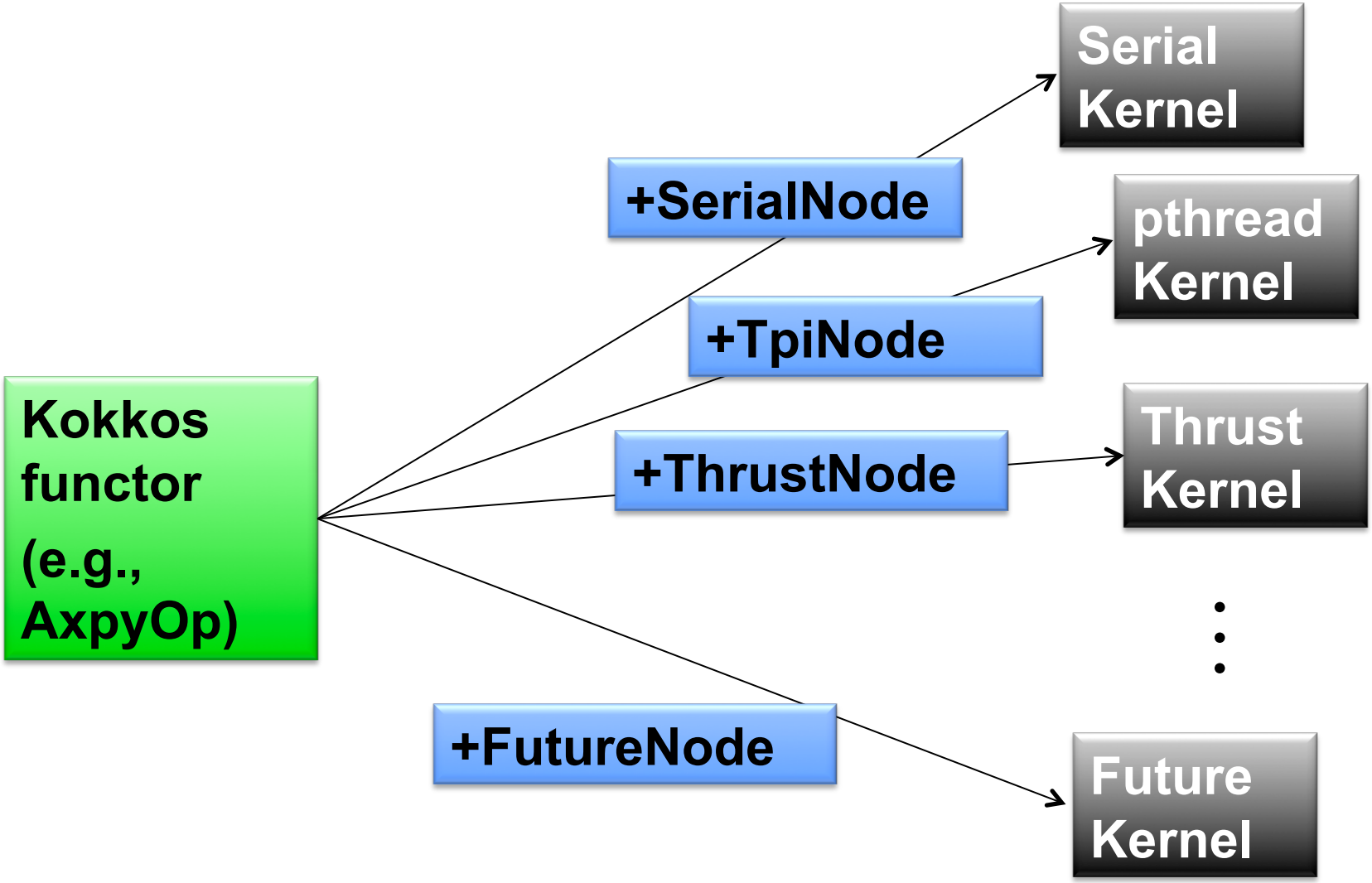
```
template <class T>
struct AxpyOp {
  const T * x;
  T * y;
  T alpha, beta;
  void execute(int i)
  { y[i] = alpha*x[i] + beta*y[i]; }
};
```

```
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



Tool: Tpetra Reduce/Transform

- 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)`
- This levels provides maximal expressiveness, but convenience wrappers are available as well.

```
// single dot() with double accumulator using custom kernels
result = Tpetra::RTI::reduce( *x, *y,  myDotProductKernel<float,double>() );
// ... or an composite adaptor and well known functors
result = Tpetra::RTI::reduce( *x, *y,
                              reductionGlob<ZeroOp<double>>(
                                    std::multiplies<float>(),
                                    std::plus<double>()) );
// ... or using inline functors via C++ lambdas
result = Tpetra::RTI::reduce( *x, *y,
                               reductionGlob<ZeroOp<double>>(
                               [](float x, float y)  {return x*y;} ,
                               [](double a, double b){return a+b;} );
// ... or using a convenience macro
result = TPETRA_REDUCE2( x, y,   x*y, ZeroOp<float>, std::plus<double>() );
```

Sandia National Laboratories

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
```

Example: Recursive Multi-Prec CG

Courtesy Chris Baker, ORNL

```
for (k=0; k<numIters; ++k) {
  A->apply(*p,*Ap);                                        // Ap = A*p
  T pAp = TPETRA_REDUCE2( p, Ap,
                          p*Ap, ZeroOp<T>, plus<T>() );// p'*Ap
  const T alpha = zr / pAp;
  TPETRA_BINARY_TRANSFORM( x,    p,
                           x + alpha*p  );                 // x = x + alpha*p
  TPETRA_BINARY_TRANSFORM( rold, r,
                           r            );                 // rold = r
  T rr = TPETRA_BINARY_PRETRANSFORM_REDUCE(
                    r, Ap,                                 // fused:
                    r - alpha*Ap,                          //      : r - alpha*Ap
                    r*r, ZeroOp<T>, plus<T>() );           //      : sum r'*r
  // recursive call to precondition this iteration
  recursiveFPCG<TS::next,LO,GO,Node>(out,db_T2);           // x_T2 = A_T2 \ b_T2
  auto plusTT = make_pair_op<T,T>(plus<T>());
  pair<T,T> both = TPETRA_REDUCE3( z, r, rold,             // fused: z'*r and z'*r_old
                                   make_pair(z*r, z*rold),
                                   ZeroPTT, plusTT );
  const T beta = (both.first - both.second) / zr;
  zr = both.first;
  TPETRA_BINARY_TRANSFORM( p, z,   z + beta*p );           // p = z + beta*p
}
```

Example: Recursive Multi-Prec CG

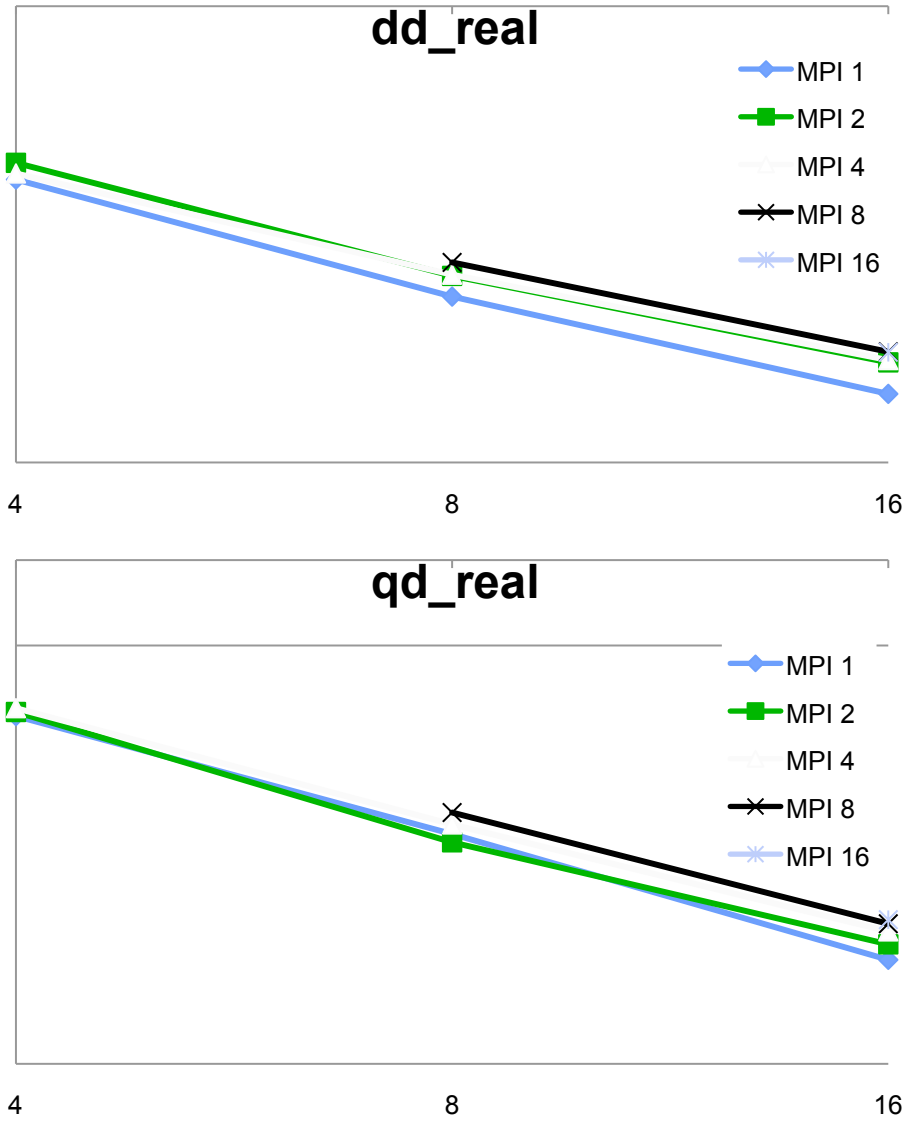
```
TBBNode initializing with numThreads == 2
TBBNode initializing with numThreads == 2

Running test with Node==Kokkos::TBBNode on rank 0/2

 Beginning recursiveFPCG<qd_real>
  Beginning recursiveFPCG<dd_real>
  |res|/|res_0|: 1.269903e-14
  |res|/|res_0|: 3.196573e-24
  |res|/|res_0|: 6.208795e-35
  Convergence detected!
  Leaving recursiveFPCG<dd_real> after 2 iterations.
 |res|/|res_0|: 2.704682e-32
  Beginning recursiveFPCG<dd_real>
  |res|/|res_0|: 4.531185e-09
  |res|/|res_0|: 6.341084e-20
  |res|/|res_0|: 8.326745e-31
  Convergence detected!
  Leaving recursiveFPCG<dd_real> after 2 iterations.
 |res|/|res_0|: 3.661388e-58
 Leaving recursiveFPCG<qd_real> after 2 iterations.
```

Example: Recursive Multi-Prec CG

- Problem: Oberwolfach/gyro
- N=17K, nnz=1M
- qd_real/dd_real/double
- MPI + TBB parallel node
- #threads = #mpi x #tbb
- Solved to over 60 digits
- Around 99.9% of time spent in double precision computation.
- Single codebase.



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

Every calculation matters

Soft Error Resilience

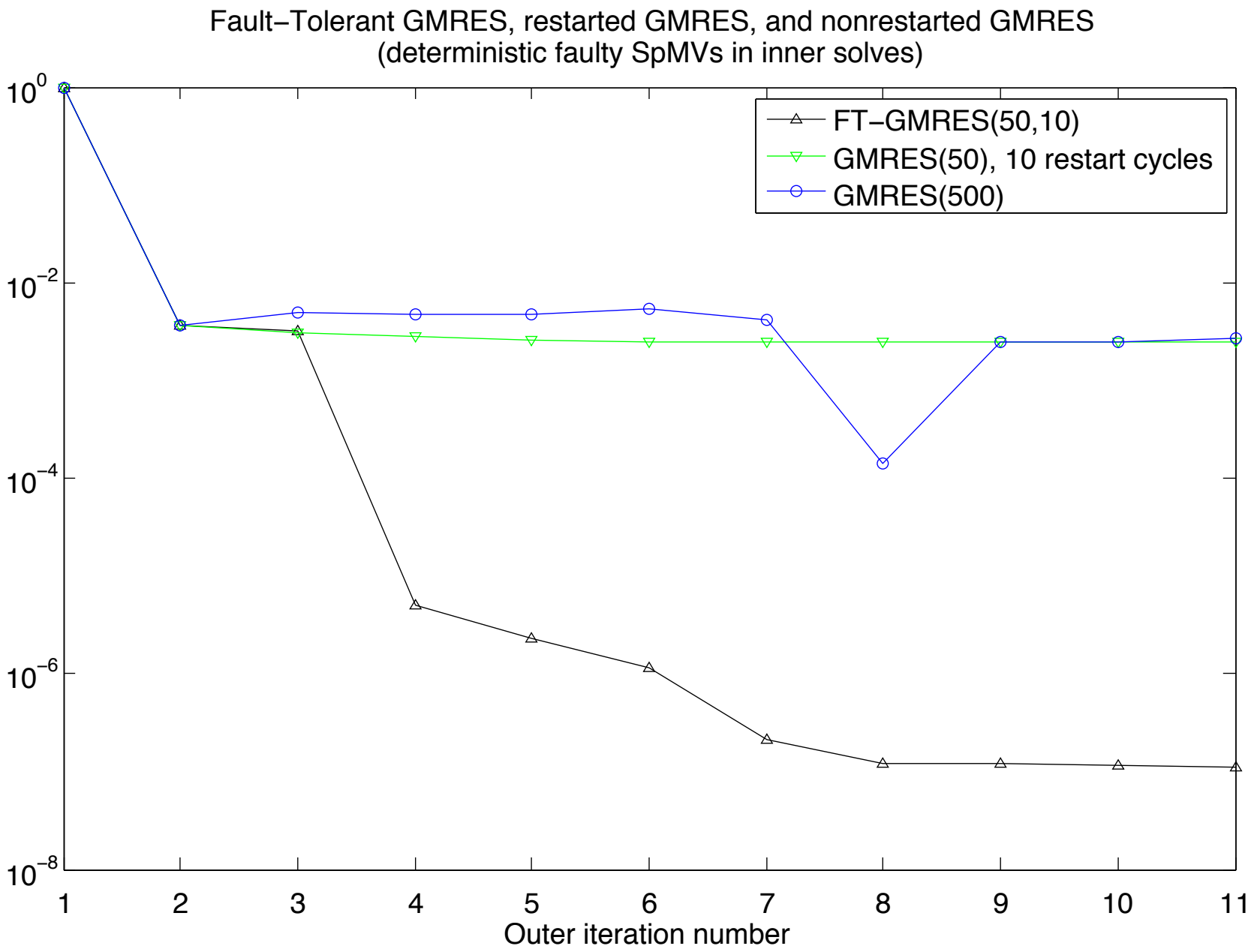
Description	Iters	FLOPS	Recursive Residual Error	Solution Error
All Correct Calcs	35	343M	4.6e-15	1.0e-6
Iter=2, y[1] += 1.0 SpMV incorrect Ortho subspace	35	343M	6.7e-15	3.7e+3
Q[1][1] += 1.0 Non-ortho subspace	N/C	N/A	7.7e-02	5.9e+5

- 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



With C++ as your hammer,
everything looks like your thumb.

"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

Trilinos

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

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

Trilinos

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. $f(x,u)=0$	Sensitivities (Automatic Differentiation: Sacado)	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 $F(x)=0$ $x \in \Re^n$		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) $\nu \in \Re^n$, $\lambda \in$		Anasazi AztecOO Ifpack, ML, etc... | Belos* T-Ifpack*, T-ML*, etc...
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 | Tpetra* Kokkos* Teuchos

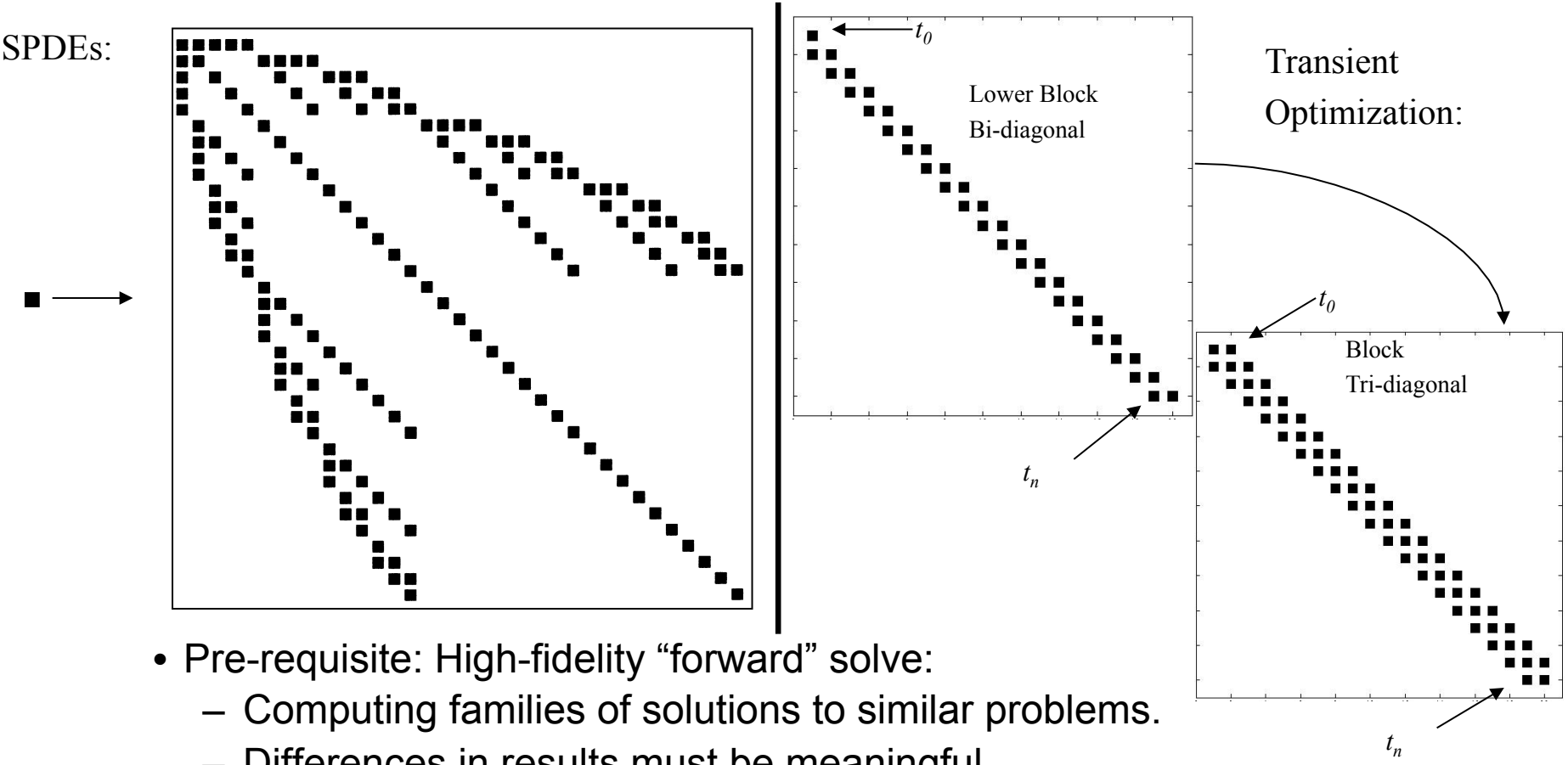
Meta-Algorithms

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.
  - Forward solve becomes "petascale kernel".

■ - 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

Summary

- Parallelism is the new commodity curve:
  - Tasks/threads, vectors.
  - Ensemble approaches OK. ICN cost not so big. Power is.
- Most future programmers won't need to write parallel code.
  - Pattern-based framework separates concerns.
  - Domain expert writes sequential fragment.
- Extended precision is not too expensive to be useful.
  - Multi-precision, intervals also possible.
- Resilience will be built into algorithms.
  - High-reliability declarations.
  - Bit-flips and UQ?
- A solution with error bars complements architecture trends:
  - Problem size growth.
  - Computation/communication ratios improve.
  - Memory usage an issue.

Sandia National Laboratories