

Scalable Manycore Computing for Sparse Computation SC 2013

<http://www.sandia.gov/~maherou/SC13ManycoreSparseTutorial.pdf>

Michael A. Heroux
Sandia National Laboratories

Collaborators:

Erik Boman, Irina Demesko, Karen Devine, H. Carter Edwards, Mark Hoemmen, Siva Rajamanickam, Christian Trott



Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.





Outline

- The Landscape of Sparse Computations:
 - ◆ Direct, Iterative.
 - ◆ Dense “containers”.
- Overview of Trilinos:
 - ◆ Capabilities & Organization.
- Parallel Computing Trends & Models:
 - ◆ How we reason about parallelism.
- Trilinos Manycore Efforts:
 - ◆ Algorithms & Data Classes
- Future Directions:
 - ◆ Resilience.

Matrices

- **Matrix (defn):** (not rigorous) An m -by- n , 2 dimensional array of numbers.
- Examples:

$$\mathbf{A} = \begin{pmatrix} 1.0 & 2.0 & 1.5 \\ 2.0 & 3.0 & 2.5 \\ 1.5 & 2.5 & 5.0 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

Sparse Matrices

- ***Sparse Matrix (defn):*** (not rigorous) An m -by- n matrix with enough zero entries that it makes sense to keep track of what is zero and nonzero.
- Example:

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} \end{pmatrix}$$



Dense vs Sparse Costs

- What is the cost of storing the tridiagonal matrix with all entries?
- What is the cost if we store each of the diagonals as vector?
- What is the cost of computing $y = Ax$ for vectors x (known) and y (to be computed):
 - ◆ If we ignore sparsity?
 - ◆ If we take sparsity into account?

Origins of Sparse Matrices

- In practice, *most* large matrices are sparse. Specific sources:
 - ◆ Differential equations.
 - Encompasses the vast majority of scientific and engineering simulation.
 - E.g., structural mechanics.
 - $F = ma$. Car crash simulation.
 - ◆ Stochastic processes.
 - Matrices describe probability distribution functions.
 - ◆ Networks.
 - Electrical and telecommunications networks.
 - Matrix element a_{ij} is nonzero if there is a wire connecting point i to point j .
 - ◆ 3D imagery for Google Earth
 - Relies on SuiteSparse (via the Ceres nonlinear least squares solver developed by Google).
 - ◆ And more...

Example: 1D Heat Equation (Laplace Equation)

- The one-dimensional, steady-state heat equation on the interval $[0,1]$ is as follows:

$$u''(x) = 0, \quad 0 < x < 1.$$

$$u(0) = a.$$

$$u(1) = b.$$

- The solution $u(x)$, to this equation describes the distribution of heat on a wire with temperature equal to a and b at the left and right endpoints, respectively, of the wire.

Finite Difference Approximation

- The following formula provides an approximation of $u''(x)$ in terms of u :

$$u''(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{2h^2} + O(h^2)$$

- For example if we want to approximate $u''(0.5)$ with $h = 0.25$:

$$u''(0.5) \approx \frac{u(0.75) - 2u(0.5) + u(0.25)}{2(1/4)^2}$$

1D Grid

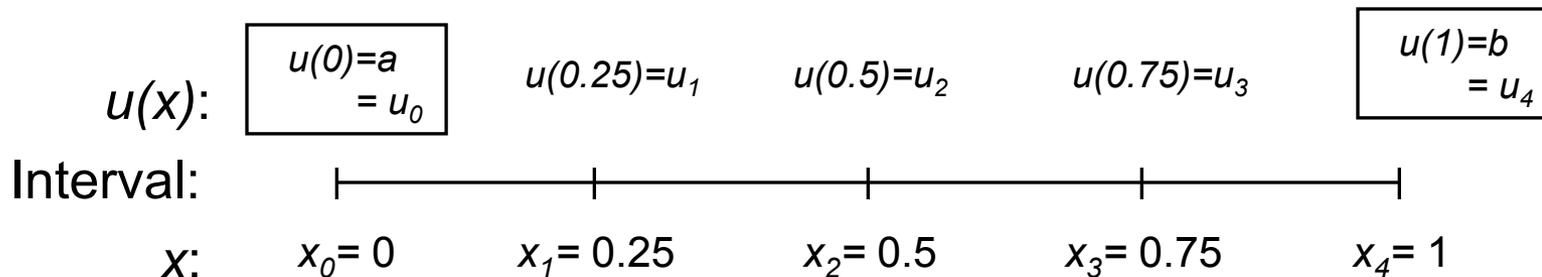
Note that it is impossible to find $u(x)$ for all values of x .

Instead we:

Create a “grid” with n points.

Then find an approximate to u at these grid points.

If we want a better approximation, we increase n .



Note:

We know u_0 and u_4 .

We know a relationship between the u_i via the finite difference equations.

We need to find u_i for $i=1, 2, 3$.

What We Know

Left endpoint:

$$\frac{1}{h^2} (u_0 - 2u_1 + u_2) = 0, \text{ or } 2u_1 - u_2 = a.$$

Right endpoint:

$$\frac{1}{h^2} (u_2 - 2u_3 + u_4) = 0, \text{ or } 2u_3 - u_2 = b.$$

Middle points:

$$\frac{1}{h^2} (u_{i-1} - 2u_i + u_{i+1}) = 0, \text{ or } -u_{i-1} + 2u_i - u_{i+1} = 0 \text{ for } i = 2.$$

Write in Matrix Form

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ b \end{bmatrix}$$

Notes:

1. This system was **assembled** from pieces of what we know.
2. This is a linear system with 3 equations and three unknowns.
3. We can easily solve.
4. Note that $n=5$ generates this 3 equation system.
5. In general, for n grid points on $[0, 1]$, we will have $n-2$ equations and unknowns.

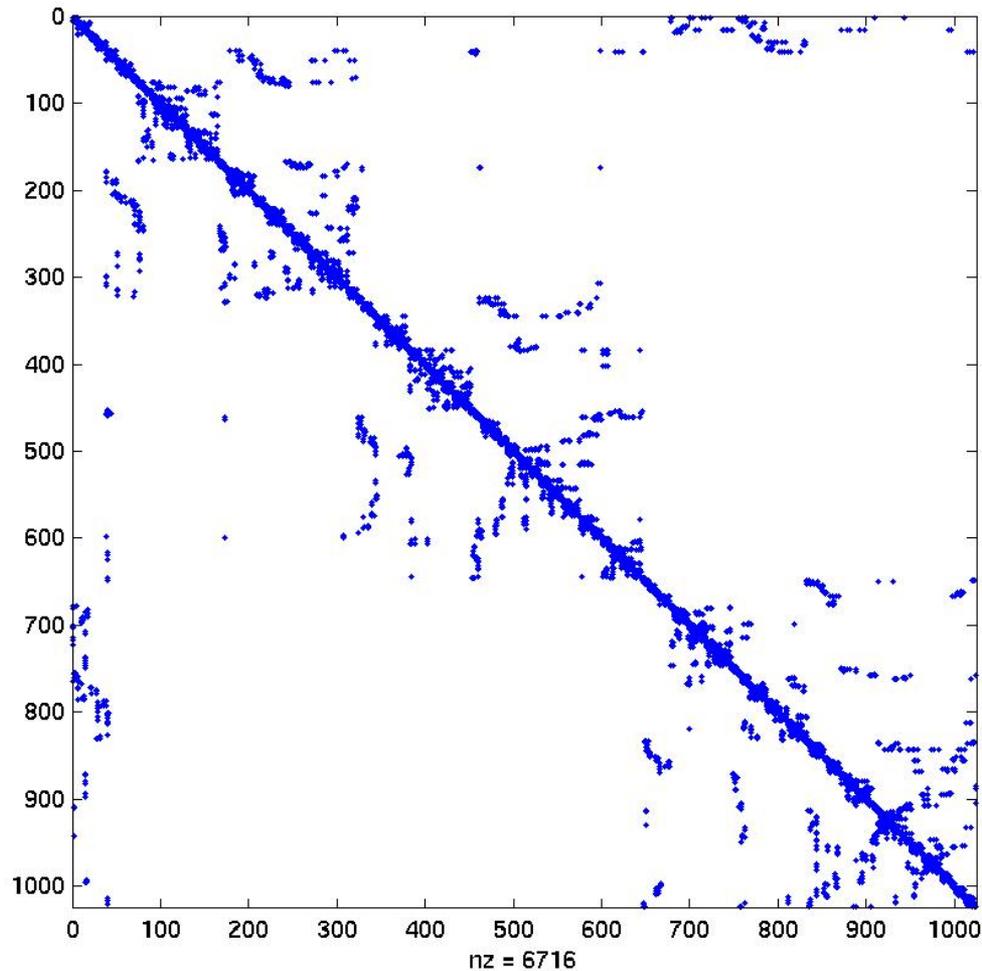
General Form of 1D Finite Difference Matrix

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} a \\ 0 \\ 0 \\ \vdots \\ b \end{bmatrix}$$

A View of More Realistic Problems

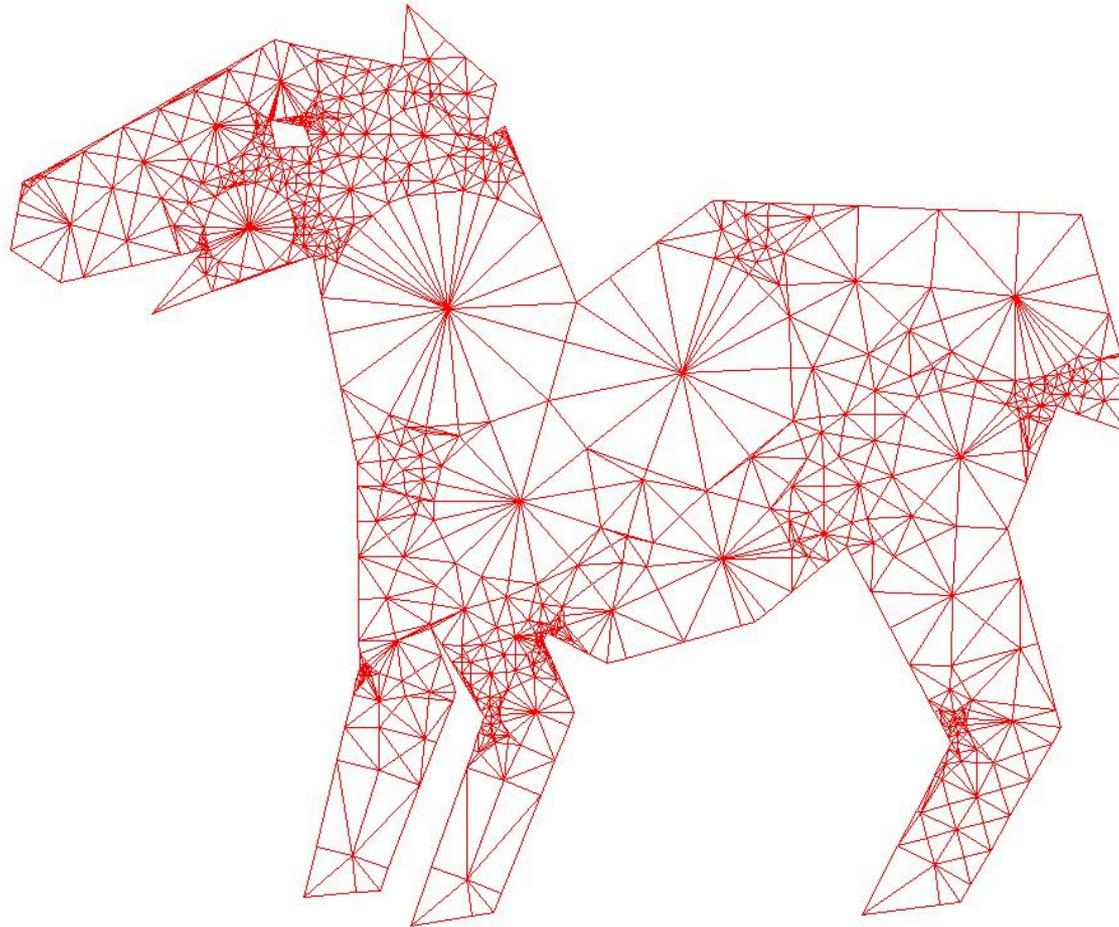
- The previous example is very simple.
- But basic principles apply to more complex problems.
- Finite difference approximations exist for *any* differential equation.
- Finite volume is widely used.
- Sandia primarily used **finite elements**.
- Leads to far more complex matrix patterns.
- Fun example...

“Tapir” Matrix (John Gilbert)



Corresponding Mesh

15



Sparse Linear Systems: Problem Definition

- A frequent requirement for scientific and engineering computing is to solve:

$$Ax = b$$

where A is a known large (sparse) matrix *a linear operator*,
 b is a known vector,
 x is an unknown vector.

NOTE: We are using x differently than before.

Previous x : Points in the interval $[0, 1]$.

New x : Vector of u values.

- Goal: Find x .
- Question: How do we solve this problem?

Sparse Direct Methods

- Construct L and U , lower and upper triangular, resp, s.t.

$$LU = A$$

- Solve $Ax = b$:

- $Ly = b$

- $Ux = y$

- Symmetric versions: $LL^T = A$, LDL^T

- When are direct methods effective?

- ◆ 1D: Always, even on many, many processors.
- ◆ 2D: Almost always, except on many, many processors.
- ◆ 2.5D: Most of the time.
- ◆ 3D: Only for “small/medium” problems on “small/medium” processor counts.

- Bottom line: Direct sparse solvers should always be in your toolbox.

Sparse Direct Solver Packages

- HSL: <http://www.hsl.rl.ac.uk>
- MUMPS: <http://mumps.enseeiht.fr>
- Pardiso: <http://www.pardiso-project.org>
- PaStiX: <http://pastix.gforge.inria.fr>
- SuiteSparse: <http://www.cise.ufl.edu/research/sparse/SuiteSparse>
- SuperLU: <http://crd-legacy.lbl.gov/~xiaoye/SuperLU/index.html>
- UMFPACK: <http://www.cise.ufl.edu/research/sparse/umfpack/>
- WSMP: http://researcher.watson.ibm.com/researcher/view_project.php?id=1426
- Trilinos/Amesos/Amesos2: <http://trilinos.org>
- Notes:
 - ♦ All have threaded parallelism.
 - ♦ All but SuiteSparse and UMFPACK have distributed memory (MPI) parallelism.
 - ♦ MUMPS, PaStiX, SuiteSparse, SuperLU, Trilinos, UMFPACK are freely available.
 - ♦ HSL, Pardiso, WSMP are available freely, with restrictions.
 - ♦ Some research efforts on GPUs, unaware of any products.
- Emerging hybrid packages:
 - ♦ PDSLIn – Sherry Li.
 - ♦ HIPS – Gaidamour, Henon.
 - ♦ Trilinos/ShyLU – Rajamanickam, Boman, Heroux.



Other Sparse Direct Solver Packages

- TAUCS : <http://www.tau.ac.il/~stoledo/taucs/>
- PSPASES : <http://www-users.cs.umn.edu/~mjoshi/pspases/>
- BCSLib : <http://www.boeing.com/phantom/bcslib/>

- “Legacy” packages that are open source but not under active development today.

Emerging Trend in Sparse Direct

- New work in low-rank approximations to off-diagonal blocks.
- Typically:
 - ◆ Off-diagonal blocks in the factorization stored as dense matrices.
- New:
 - ◆ These blocks have low rank (up to the accuracy needed for solution).
 - ◆ Can be represented by approximate SVD.
- Still uncertain how broad the impact will be.
 - ◆ Will rank- k SVD continue to have low rank for hard problems?
- Potential: Could be breakthrough for extending sparse direct method to much larger 3D problems.

Iterative Methods

- Given an initial guess for x , called $x^{(0)}$, ($x^{(0)} = 0$ is acceptable) compute a sequence $x^{(k)}$, $k = 1, 2, \dots$ such that each $x^{(k)}$ is “closer” to x .
- Definition of “close”:
 - ◆ Suppose $x^{(k)} = x$ exactly for some value of k .
 - ◆ Then $r^{(k)} = b - Ax^{(k)} = 0$ (the vector of all zeros).
 - ◆ And $norm(r^{(k)}) = sqrt(\langle r^{(k)}, r^{(k)} \rangle) = 0$ (a number).
 - ◆ For any $x^{(k)}$, let $r^{(k)} = b - Ax^{(k)}$
 - ◆ If $norm(r^{(k)}) = sqrt(\langle r^{(k)}, r^{(k)} \rangle)$ is small ($< 1.0E-6$ say) then we say that $x^{(k)}$ is close to x .
 - ◆ The vector r is called the residual vector.

Sparse Iterative Solver Packages

- PETSc: <http://www.mcs.anl.gov/petsc>
- hypre: https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html
- Trilinos: <http://trilinos.sandia.gov>
- HSL: <http://www.hsl.rl.ac.uk>
- Sparskit: <http://www-users.cs.umn.edu/~saad/software>
- Notes:
 - ◆ There are many other efforts, but I am unaware of any that have a broad user base like hypre, PETSc and Trilinos.
 - ◆ Sparskit, and other software by Yousef Saad, is not a product with a large official user base, but these codes appear as embedded (serial) source code in many applications.
 - ◆ PETSc and Trilinos support threading, distributed memory (MPI) and growing functionality for accelerators.
 - ◆ Many of the direct solver packages support some kind of iteration, if only iterative refinement.



Trilinos Overview

What is Trilinos?

- Object-oriented software framework for...
- Solving big complex science & engineering problems
- More like LEGO™ bricks than Matlab™



Trilinos Contributors

Travis Austin
Chris Baker
Ross Bartlett
Eric Bavier
Matt Betterncourt
Pavel Bochev
Erik Boman
Cedric Chevalier
Todd Coffey
Julien Cortial
Eric Cyr
David Day
Karen Devine
Clark Dohrmann
Carter Edwards
Jeremie Gaidamour
Deaglan Halligan
Glen Hansen
Heath Hanshaw
David Hensinger
Mike Heroux
Mark Hoemmen
Russell Hooper
Jonathan Hu
Chetan Jhurani
Patrick Knupp
Joe Kotulski
Jason Kraftcheck
Nicole Lemaster Slattengren

Jay Lofstead
Kevin Long
Karla Morris
Kurtis Nusbaum
Ron Oldfield
Mike Parks
Roger Pawlowski
Brent Perschbacher
Kara Peterson
Eric Phipps
Radu Popescu
Vicki Porter
Andrey Prokopenko
Siva Rajamanickam
Denis Ridzal
Lee Ann Riesen
Damian Rouson
Andrew Salinger
Nico Schlömer
Chris Siefert
Greg Sjaardema
Bill Spotz
Dan Sunderland
Heidi Thornquist
Christian Trott
Boyd Tidwell
Ray Tuminaro
Dena Vigil
Jim Willenbring
Alan Williams

Past Contributors
Paul Boggs
Lee Buermann
Cedric Chevalier
Jason Cross
Kelly Fermoyle
David Gay
Michael Gee
Esteban Guillen
Bob Heaphy
Ulrich Hetmaniuk
Robert Hoekstra
Vicki Howle
Kris Kampshoff
Ian Karlin
Sarah Knepper
Tammy Kolda
Rich Lehoucq
Joe Outzen
Mike Phenow
Marzio Sala
Paul Sexton
Bob Shuttleworth
Ken Stanley
Michael Wolf



Background/Motivation

Trilinos



- ◆ R&D 100 Winner
- ◆ 9400 Registered Users.
- ◆ 32,400 Downloads.
- ◆ Open Source.



Laptops to Leadership systems

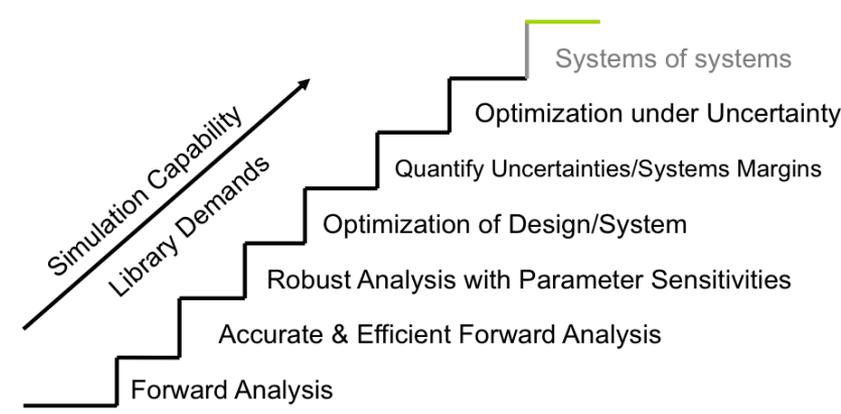
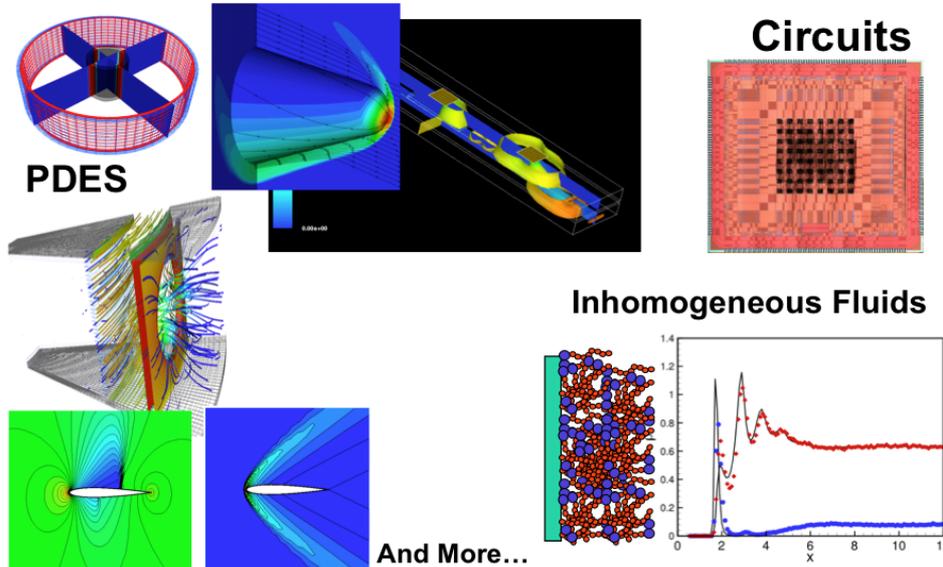
Optimal Kernels to Optimal Solutions:

- ◆ Geometry, Meshing
- ◆ Discretizations, Load Balancing.
- ◆ Scalable Linear, Nonlinear, Eigen, Transient, Optimization, UQ solvers.
- ◆ Scalable I/O, GPU, Manycore

- ◆ 60 Packages.
- ◆ Binary distributions:
 - ◆ Cray LIBSCI
 - ◆ Debian, Ubuntu
 - ◆ Intel (in process)



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 Strategic Goals

- Scalable Computations: As problem size and processor counts increase, the cost of the computation will remain nearly fixed.
- Hardened Computations: Never fail unless problem essentially intractable, in which case we diagnose and inform the user why the problem fails and provide a reliable measure of error.
- Full Vertical Coverage: Provide leading edge enabling technologies through the entire technical application software stack: from problem construction, solution, analysis and optimization.
- *Grand* Universal Interoperability: All Trilinos **packages**, and important external packages, will be interoperable, so that any combination of packages and external software (e.g., PETSc, Hypra) that makes sense algorithmically will be **possible** within Trilinos.
- Universal Accessibility: All Trilinos capabilities will be available to users of major computing environments: C++, Fortran, Python and the Web, and from the desktop to the latest scalable systems.
- Universal Solver RAS: Trilinos will be:
 - ♦ **Reliable**: Leading edge hardened, scalable solutions for each of these applications
 - ♦ **Available**: Integrated into every major application at Sandia
 - ♦ **Serviceable**: “Self-sustaining”.

Algorithmic
Goals

Software
Goals



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)
 - ◆ **User Experience: (W. Spotz)**
- Each leader provides strategic direction across all Trilinos packages within area.

Unique features of Trilinos

- Huge library of algorithms
 - ◆ Linear and nonlinear solvers, preconditioners, ...
 - ◆ Optimization, transients, sensitivities, uncertainty, ...
- Growing support for multicore & hybrid CPU/GPU
 - ◆ Built into the new Tpetra linear algebra objects
 - Therefore into iterative solvers with zero effort!
 - ◆ Unified intranode programming model
 - ◆ Spreading into the whole stack:
 - Multigrid, sparse factorizations, element assembly...
- Support for mixed and arbitrary precisions
 - ◆ Don't have to rebuild Trilinos to use it
- Support for huge (> 2B unknowns) problems

Trilinos Current Release

- Trilinos 11.4 current, 11.6 RSN.
 - ◆ 54 packages.
 - ◆ Multicore/GPU enhancements to 5 packages.
- Downloads:
 - ◆ Average: 500/month.
- Website: trilinos.org.



Trilinos software organization

Trilinos Package Summary

	Objective	Package(s)
Discretizations	Meshing & Discretizations	STK, Intrepid, Pamgen, Sundance, ITAPS, Mesquite
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Services	Linear algebra objects	Epetra, Tpetra, Kokkos , Xpetra
	Interfaces	Thyra, Stratimikos, RTOp, FEI, Shards
	Load Balancing	Zoltan, Isorropia, Zoltan2
	“Skins”	PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika
	C++ utilities, I/O, thread API	Teuchos, EpetraExt, Kokkos, Triutils, ThreadPool, Phalanx, Trios
Solvers	Iterative linear solvers	AztecOO, Belos, Komplex
	Direct sparse linear solvers	Amesos, Amesos2, ShyLU
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi, Rbgen
	ILU-type preconditioners	AztecOO, IFPACK, Ifpack2, ShyLU
	Multilevel preconditioners	ML, CLAPS, Muelu
	Block preconditioners	Meros, Teko
	Nonlinear system solvers	NOX, LOCA, Piro
	Optimization (SAND)	MOOCHO, Aristos, TriKota, Globipack, Optipack
	Stochastic PDEs	Stokhos



Interoperability vs. Dependence

(“Can Use”)

(“Depends On”)

- Although most Trilinos packages have no explicit dependence, often packages must interact with *some* other packages:
 - ◆ NOX needs operator, vector and linear solver objects.
 - ◆ AztecOO needs preconditioner, matrix, operator and vector objects.
 - ◆ Interoperability is enabled at configure time.
 - ◆ Trilinos **cmake** system is vehicle for:
 - Establishing interoperability of Trilinos components...
 - Without compromising individual package autonomy.
 - Trilinos_ENABLE_ALL_OPTIONAL_PACKAGES option
- Architecture supports simultaneous development on many fronts.



Trilinos is made of packages

- Not a monolithic piece of software
 - ◆ Like LEGO™ bricks, not Matlab™
- Each package:
 - ◆ Has its own development team and management
 - ◆ Makes its own decisions about algorithms, coding style, etc.
 - ◆ May or may not depend on other Trilinos packages
- Trilinos is not “indivisible”
 - ◆ You don’t need all of Trilinos to get things done
 - ◆ Any subset of packages can be combined and distributed
 - ◆ Current public release contains ~50 of the 55+ Trilinos packages
- Trilinos top layer framework
 - ◆ Not a large amount of source code: ~1.5%
 - ◆ Manages package dependencies
 - Like a GNU/Linux package manager
 - ◆ Runs packages’ tests nightly, and on every check-in
- Package model supports multifrontal development
- New effort to create apps by gluing Trilinos together: Albany



Software Development and Delivery

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, for now.

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

<p>Optimization Unconstrained: Constrained:</p>	<p>Find $u \in \mathbb{R}^n$ that minimizes $g(u)$ Find $x \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$ that minimizes $g(x, u)$ s.t. $f(x, u) = 0$</p>	<p style="writing-mode: vertical-rl; transform: rotate(180deg);">Sensitivities (Automatic Differentiation: Sacado)</p>	MOOCHO
<p>Bifurcation Analysis</p>	<p>Given nonlinear operator $F(x, u) \in \mathbb{R}^{n+m}$ For $F(x, u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$</p>		LOCA
<p>Transient Problems DAEs/ODEs:</p>	<p>Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0, T], x(0) = x_0, \dot{x}(0) = x'_0$ for $x(t) \in \mathbb{R}^n, t \in [0, T]$</p>		Rythmos
<p>Nonlinear Problems</p>	<p>Given nonlinear operator $F(x) \in \mathbb{R}^m \rightarrow \mathbb{R}^m$ Solve $F(x) = 0 \quad x \in \mathbb{R}^n$</p>		NOX
<p>Linear Problems Linear Equations: Eigen Problems:</p>	<p>Given Linear Ops (Matrices) $A, B \in \mathbb{R}^{m \times n}$ Solve $Ax = b$ for $x \in \mathbb{R}^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \mathbb{R}^n, \lambda \in \mathbb{C}$</p>		Anasazi
<p>Distributed Linear Algebra Matrix/Graph Equations:</p>	<p>Compute $y = Ax; A = A(G); A \in \mathbb{R}^{m \times n}, G \in \mathcal{S}^{m \times n}$</p>		Ifpack, ML, etc... AztecOO
<p>Vector Problems:</p>	<p>Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \mathbb{R}^n$</p>		Epetra
			Teuchos

Solver Software Stack

Phase I packages

Phase II packages

Phase III packages: Manycore*, templated

Optimization	Find $u \in \mathbb{R}^n$ that minimizes $g(u)$ Unconstrained: Find $x \in \mathbb{R}^m$ and $u \in \mathbb{R}^n$ that Constrained: minimizes $g(x, u)$ s.t. $f(x, u) = 0$	Sensitivities (Automatic Differentiation: Sacado)	MOOCHO	
Bifurcation Analysis	Given nonlinear operator $F(x, u) \in \mathbb{R}^{n+m}$ For $F(x, u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$		LOCA	T-LOCA
Transient Problems	Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0, T], x(0) = x_0, \dot{x}(0) = x'_0$ for $x(t) \in \mathbb{R}^n, t \in [0, T]$		Rythmos	
Nonlinear Problems	Given nonlinear operator $F(x) \in \mathbb{R}^m \rightarrow \mathbb{R}^m$ Solve $F(x) = 0 \quad x \in \mathbb{R}^n$		NOX	T-NOX
Linear Problems	Given Linear Ops (Matrices) $A, B \in \mathbb{R}^{m \times n}$ Solve $Ax = b$ for $x \in \mathbb{R}^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \mathbb{R}^n, \lambda \in \mathbb{C}$		Anasazi	
Linear Equations:			AztecOO	Belos*
Eigen Problems:			Ifpack, ML, etc...	Ifpack2*, Muelu*, etc.
Distributed Linear Algebra	Matrix/Graph Equations: Compute $y = Ax; A = A(G); A \in \mathbb{R}^{m \times n}, G \in \mathcal{S}^{m \times n}$ Vector Problems: Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \mathbb{R}^n$		Epetra	Tpetra* Kokkos*
		Teuchos		

Getting Performance From Iterative Solvers

- Classical preconditioned iterative solver, performance is driven by these underlying kernels:
 - Sparse matrix times dense vector $y = Ax$ (SpMV).
 - Sparse triangular solve $Ly = b$ (SpSV).
 - Vector updates ($w = ax + by$), a, b scalars.
 - Dot products ($a = x^T y$) a scalar.
- Block Krylov:
 - SpMM, SpSM: Multiple RHS.
 - GEMM: But different shapes than LAPACK (block vector update, dot product).
- s-step (CA) methods:
 - Matrix powers kernel $y_i = A^i x$, $i = 1, s$.
 - Sparse triangular solve: ??
 - Multivector updates and dot products
- Multi-level solvers:
 - Hierarchy setup, coarse grid solve.

Linear Conjugate Gradient Methods

Linear Conjugate Gradient Algorithm

Types of operations

Types of objects

Compute $r^{(0)} = b - Ax^{(0)}$ for the initial guess $x^{(0)}$.

for $i = 1, 2, \dots$

$$\rho_{i-1} = \langle r^{(i-1)}, r^{(i-1)} \rangle$$

$$\beta_{i-1} = \rho_{i-1} / \rho_{i-2} \quad (\beta_0 = 0)$$

$$p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)} \quad (p^{(1)} = r^{(1)})$$

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

$$\gamma_i = \langle p^{(i)}, q^{(i)} \rangle$$

$$\alpha_i = \rho_{i-1} / \gamma_i$$

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

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

check convergence; continue if necessary

end

linear operator applications

vector-vector operations

Scalar operations

scalar product $\langle x, y \rangle$ defined by vector space

Linear Operators

- A

Vectors

- r, x, p, q

Scalars

- $\rho, \beta, \gamma, \alpha$

Vector spaces?

- \mathcal{X}

$$\langle v, w \rangle = v_1 * w_1 + v_2 * w_2 + \dots + v_n * w_n$$

General Sparse Matrix

- Example:

$$A = \begin{pmatrix} a_{11} & 0 & 0 & 0 & 0 & a_{16} \\ 0 & a_{22} & a_{23} & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & 0 & a_{35} & 0 \\ 0 & 0 & 0 & a_{44} & 0 & 0 \\ 0 & 0 & a_{53} & 0 & a_{55} & a_{56} \\ a_{61} & 0 & 0 & 0 & a_{65} & a_{66} \end{pmatrix}$$

Compressed Row Storage (CRS) Format

Idea:

- Create
 - 1 length nnz array of non-zero value.
 - 1 length nnz array of column indices.
 - 1 length $m+1$ array of ints:
 - double * values = new double [nnz];
 - int * colIndices = new int[nnz];
 - int * rowPointers = new int[m+1];

nnz – Number of nonzero terms in the matrix.

m – Matrix dimension.

Compressed Row Storage (CRS) Format

Fill arrays as follows:

```
rowPointer[0] = 0;
```

```
double * curValuePtr = values;
```

```
int      * curIndicesPtr = colIndices;
```

```
for (i=0; i<m; ++i) { // for each row
```

```
    rowPointer[i+1] = numRows, number of nonzero entries in row i.
```

```
    for (j=0; j<numRowEntries; j++) { // for each entry in row i
```

```
        *curValuePtr++ = value of jth nonzero entry in row i.
```

```
        *curIndicesPtr ++ = column index of jth nonzero entry in row i.
```

```
    }
```

```
}
```

CRS Example

$$A = \begin{pmatrix} 4 & 0 & 0 & 1 \\ 0 & 3 & 0 & 2 \\ 0 & 0 & 6 & 0 \\ 5 & 0 & 9 & 8 \end{pmatrix}$$

values = {4, 1, 3, 2, 6, 5, 9, 8},

colIndices = {0, 3, 1, 3, 2, 0, 2, 3}

nrowPointers = {0, 2, 4, 5, 8}

Serial Sparse MV

```
int sparsemv( int m, double * values,
              int * colIndices, int * rowPointers,
              double * x, double * y){

for (int i=0; i< m; ++i){
    double sum = 0.0;
    curNumEntries = rowPointers[i+1] - rowPointers[i];
    double * curVals = values[rowPointers[i]];
    int * curInds = colIndices[rowPointers[i]];

    for (int j=0; j< curNumEntries; j++)
        sum += curVals[j]*x[curInds[j]];
    y[i] = sum;
}
return(0);
}
```

Getting Good Performance From Iterative Methods

- Optimize the above kernels:
 - Across many node types (using DRY).
 - Requires new algorithms, new implementations.
 - Biggest challenges:
 - Maintain performance and portability.
 - Manycore smoothers (for multi-level preconditioners).
 - Kernels for latency, throughput and hybrid parallel nodes.
 - Permutations, partitionings.
 - More ...
- Don't forget about linear system assembly.
 - Will get to it later...



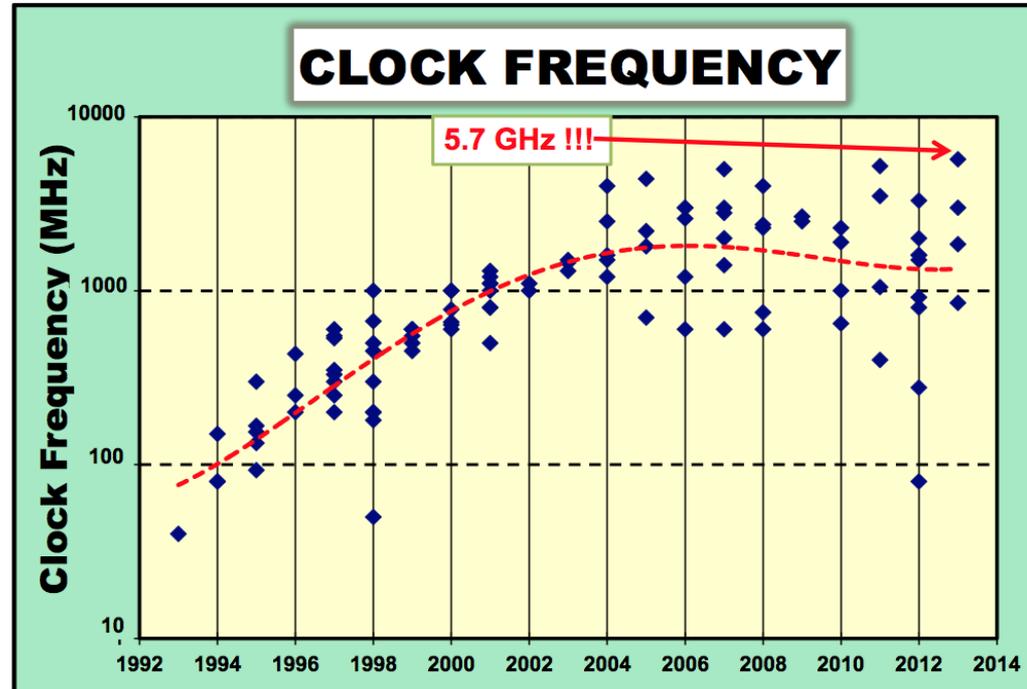
Parallel Computing Trends & Models

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

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

What is Different: Old Commodity Trends Failing

- Clock Speed.
 - Well-known.
 - Related: Instruction-level Parallelism (ILP).
- Number of nodes.
 - Connecting 100K nodes is complicated.
 - Electric bill is large.
- Memory per core.
 - Going down (but some hope in sight).
- Consistent performance.
 - Equal work \nrightarrow Equal execution time.
 - Across peers or from one run to the next.



International Solid-State Circuits Conference (ISSCC 2013) Report
http://lisscc.org/doc/2013/2013_Trends.pdf

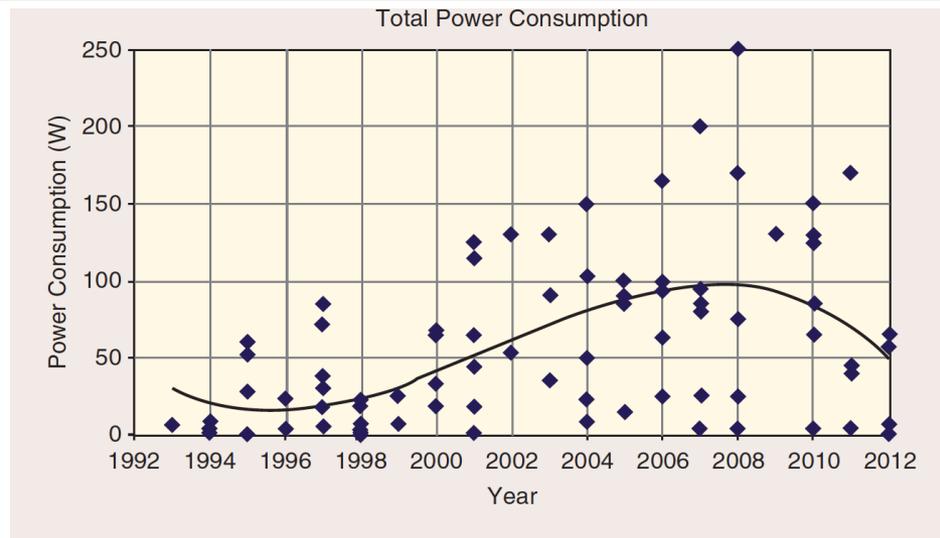
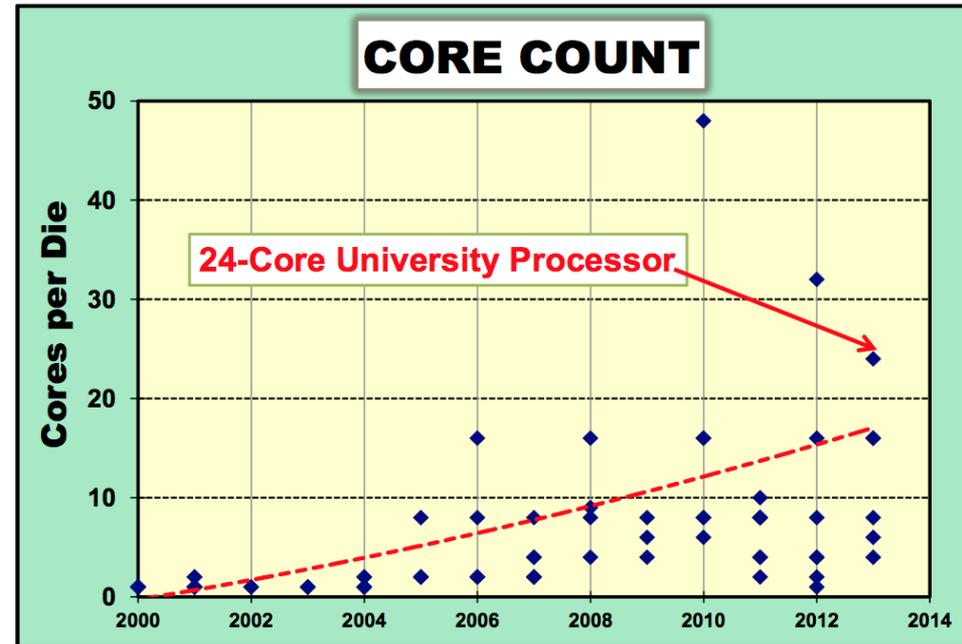
New Commodity Trends and Concerns Emerge

Big Concern: Energy Efficiency.

- Thread count.
 - Occupancy rate.
 - State-per-thread.
- SIMT/SIMD (Vectorization).
- Heterogeneity:
 - Performance variability.
 - Core specialization.
- Memory trends:
 - Exciting, hard to predict.
 - Could be better, faster, *and* cheaper!

Take-away:

Parallelism is essential.





Factoring 1K to 1B-Way Parallelism

- Why 1K to 1B?
 - Clock rate: $O(1\text{GHz}) \rightarrow O(10^9)$ ops/sec sequential
 - Terascale: 10^{12} ops/sec $\rightarrow O(10^3)$ simultaneous ops
 - 1K parallel intra-node.
 - Petascale: 10^{15} ops/sec $\rightarrow O(10^6)$ simultaneous ops
 - 1K-10K parallel intra-node.
 - 100-1K parallel inter-node.
 - Exascale: 10^{18} ops/sec $\rightarrow O(10^9)$ simultaneous ops
 - 1K-10K parallel intra-node.
 - 100K-1M parallel inter-node.



The HPC Ecosystem

Three Parallel Computing Design Points

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

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

Common Element

Most applications will not adopt an exascale programming strategy that is incompatible with tera and peta scale.

MPI+X Parallel Programming Model: Multi-level/Multi-device

HPC Value-Added

Inter-node/*inter-device* (distributed) parallelism and resource management

Message Passing

network of computational nodes

Node-local control flow (serial)

Broad Community Efforts

computational node with manycore CPUs and / or GPGPU

Intra-node (manycore) parallelism and resource management

Threading

Stateless, vectorizable, efficient computational kernels run on each core

stateless kernels



Incentives for MPI+X

- Almost all DOE scalable applications use MPI.
 - MPI provides portability layer.
 - Typically app developer accesses via conceptual layer.
 - Could swap in another SPMD approach (UPC, CAF).
 - Even dynamic SPMD is possible. Adoption expensive.
- Entire computing community is focused on X.
 - It takes a community...
 - Many promising technologies emerging.
 - Industry very interested in programmer productivity.
- MPI and X interactions well understood.
 - Straight-forward extension of existing MPI+Serial.
 - New MPI features will address specific threading needs.



Effective node-level parallelism: First priority

- Future performance is mainly from node improvements.
 - Number of nodes is not increasing dramatically.
- Application refactoring efforts on node are disruptive:
 - Almost every line of code will be displaced.
 - All current serial computations must be threaded.
 - Successful strategy similar to SPMD migration of 90s.
 - Define parallel pattern framework.
 - Make framework scalable for minimal physics.
 - Migrate large sequential fragments into new framework.
- If no node parallelism, we fail at all computing levels.



Algorithms Challenges from HW Trends

- 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.
- For more challenging problems.
- Certainly internode parallelism continues to be important, but:
 - Improvements can mostly be incremental.
 - However MPI interface changes are crucial.



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.
 - Express 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?



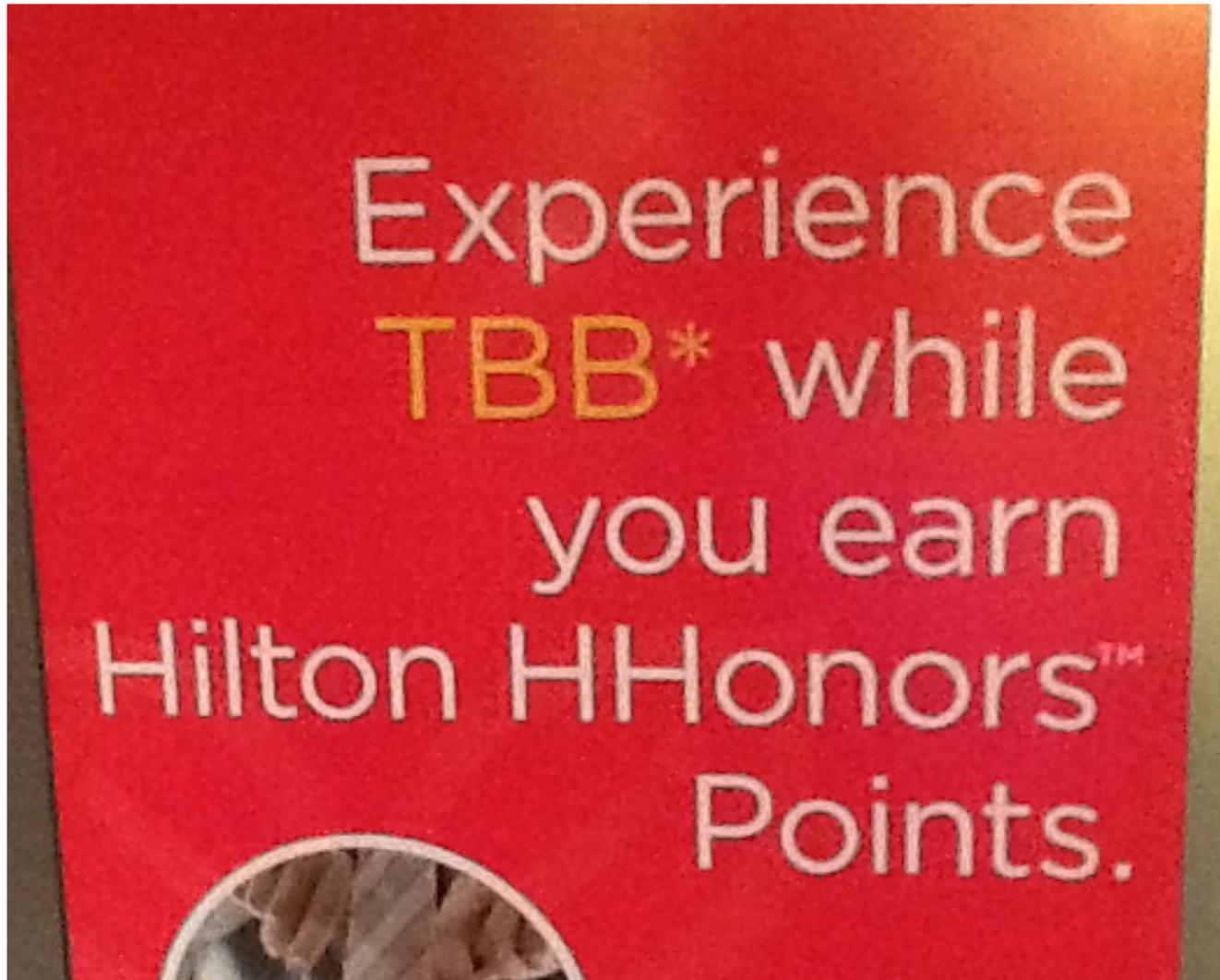
Challenge: Achieve Scalable 1B-way Concurrency

- 10^{18} Ops/sec with 10^9 clock rates: 10^9 Concurrency.
- Question: What role (if any) will MPI play?
- Answer: Major role as MPI+X.
 - MPI: Today's MPI with several key enhancements.
 - X: Industry-provided; represents numerous options.
- Why: MPI+X is leveraged, synergistic, doable.
 - Resilience: Algorithms + MPI/Runtime enhancements.
 - Programmability: There is a path.
- Urgent: Migration to manycore must begin in earnest.
 - We can't wait around for some magic exascale programming model.
 - We have to begin in earnest to learn about X options and deploy as quickly as possible.



Manycore Concerns for Libraries

Node parallelism will impact everything





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



Multi-dimensional Dense Arrays

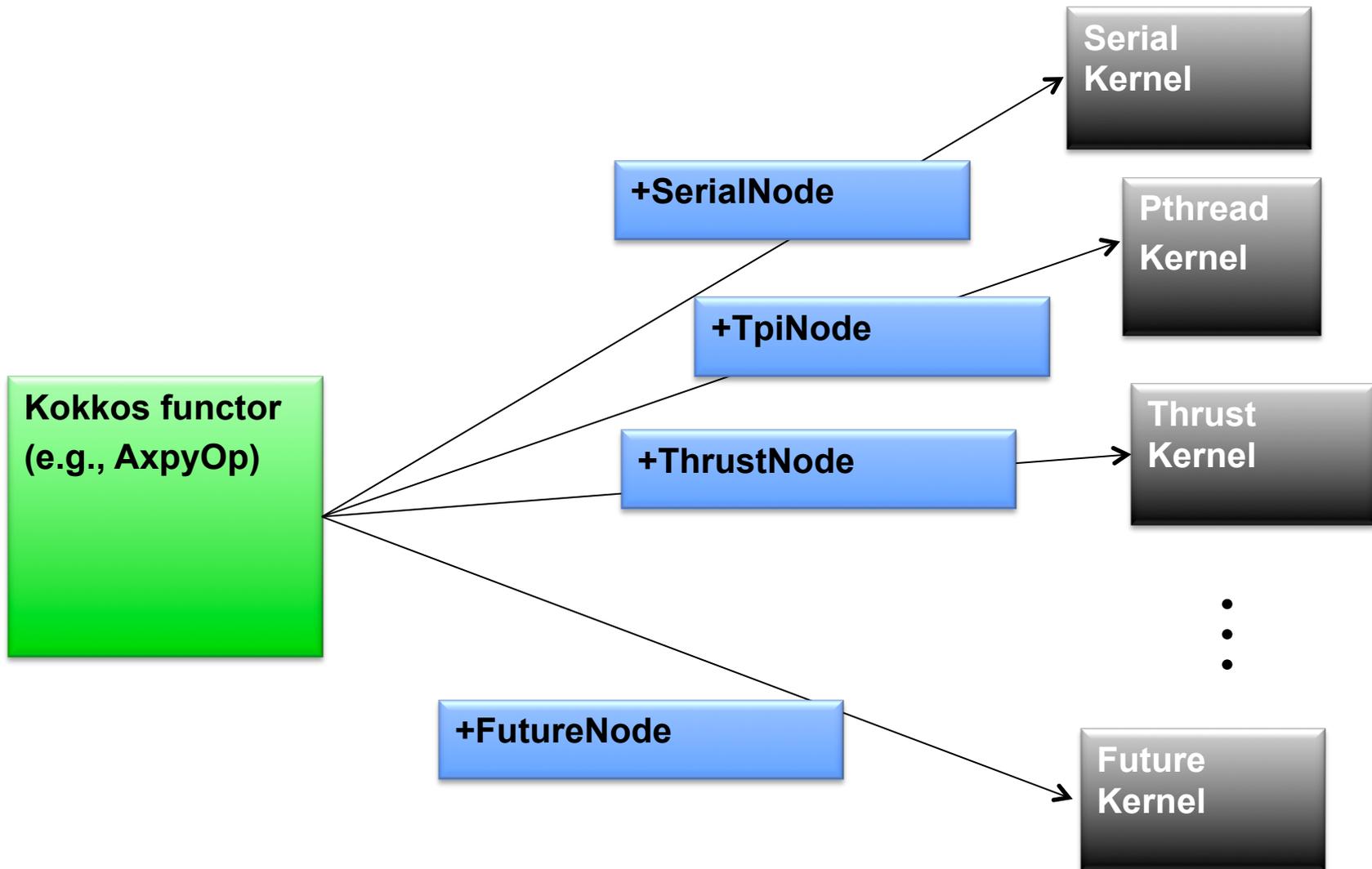
- Many computations work on data stored in multi-dimensional arrays:
 - Finite differences, volumes, elements.
 - Sparse iterative solvers.
- Dimension are (k,l,m,\dots) where one dimension is long:
 - $A(3,1000000)$
 - 3 degrees of freedom (DOFs) on 1 million mesh nodes.
- A classic data structure issue is:
 - Order by DOF: $A(1,1), A(2,1), A(3,1); A(1,2) \dots$ or
 - By node: $A(1,1), A(1,2), \dots$
- Adherence to raw language arrays force a choice.

Struct-of-Arrays vs. Array-of-Structs



A False Dilemma

Compile-time Polymorphism





A Bit about Functors

Classic function “ComputeWAXPBY_ref.cpp”

```
/*!
```

```
Routine to compute the update of a vector with the sum of two  
scaled vectors where:  $w = \alpha * x + \beta * y$ 
```

```
@param[in] n the number of vector elements (on this processor)
```

```
@param[in] alpha, beta the scalars applied to x and y respectively.
```

```
@param[in] x, y the input vectors
```

```
@param[out] w the output vector.
```

```
@return returns 0 upon success and non-zero otherwise
```

```
*/
```

```
int ComputeWAXPBY_ref(const local_int_t n, const double alpha, const double *  
const x, const double beta, const double * const y, double * const w) {
```

```
for (local_int_t i=0; i<n; i++) w[i] = alpha * x[i] + beta * y[i];
```

```
return(0);
```

```
}
```

A Bit about Functors

Functor-calling function “ComputeWAXPBY.cpp”

```
/*!
```

```
Routine to compute the update of a vector with the sum of two  
scaled vectors where:  $w = \alpha * x + \beta * y$ 
```

```
@param[in] n the number of vector elements (on this processor)
```

```
@param[in] alpha, beta the scalars applied to x and y respectively.
```

```
@param[in] x, y the input vectors
```

```
@param[out] w the output vector.
```

```
@return returns 0 upon success and non-zero otherwise
```

```
*/
```

```
int ComputeWAXPBY(const local_int_t n, const double alpha, const double * const x, const  
double beta, const double * const y, double * const w) {
```

```
// for (local_int_t i=0; i<n; i++) w[i] = alpha * x[i] + beta * y[i];
```

```
tbb::parallel_for(tbb::blocked_range<size_t>(0,n), waxpby_body(n, alpha, x, beta, y, w) );
```

```
return(0);
```

```
}
```

A Bit about Functors

Functor “waxpby body”

```
#include "tbb/parallel_for.h"
#include "tbb/blocked_range.h"
class waxpby_body{
    size_t n_;
    double alpha_;
    double beta_;
    const double * const x_;
    const double * const y_;
    double * const w_; public:
waxpby_body(size_t n, const double alpha, const double * const x, const double beta,
const double * const y, double * const w)
    : n_(n), alpha_(alpha), x_(x), beta_(beta), y_(y), w_(w) { }
void operator() (const tbb::blocked_range<size_t> &r) const {
    const double * const x = x_;
    const double * const y = y_;
    double * const w = w_;
    double alpha = alpha_;
    double beta = beta_;
    for(size_t i=r.begin(); i!=r.end(); i++) w[i] = alpha * x[i] + beta * y[i];
}
};
```



Kokkos Library

- Kokkos IS:
 - An implementation of the programming model
 - Consolidation of proxy-applications' common functionality
 - “Low level” enabling data structures and algorithms
 - Extremely attentive to:
 1. Portability & performance (as per project charter)
 2. Usability: ease of use, error detection, extensibility, maintainability, ...
- Kokkos IS NOT:
 - A linear algebra library
 - A discretization library
 - A mesh library
 - Intent: Build such libraries on top of KokkosArray



The Problem / Challenge

Future of HPC: Manycore Accelerators

- Multicore CPU
 - Increasing core counts with decreasing global memory / core
 - Cores share caches and memory controllers
 - Non-uniform memory access (NUMA), performance issues
 - Increasing vector unit lengths
 - Memory access patterns critical for *best* performance
- Manycore GPU (e.g., NVIDIA, AMD)
 - Physically separate memory with data-transfer overhead
 - Work-dispatch interaction between host and device
 - Memory controller optimized for thread-gang (warp) based access
 - Memory access patterns critical for *acceptable* performance
- Is all about Memory Access Patterns



The Problem / Challenge

Future of HPC: Manycore Accelerators

- Shared Memory Threading within MPI is *required*
 - Cannot run MPI-everywhere on GPU
 - Cannot afford MPI process memory for every core
 - Cannot scale MPI collectives to millions of CPU cores
 - Unless you have heroic hardware: Blue Gene Q
- Memory Access Patterns are Critical
 - Correctness – no race conditions among threads
 - Performance – proper blocking or striding
- Access Pattern Requirements are Device-dependent
 - CPU-core : blocking for cache and cache-lines
 - GPU : striding for coalesced access
 - “array of structures” vs. “structure of arrays”

Programming Model Concept

- Manycore Device
 - Has a separate memory space (physically or logically)
 - Dispatch work to cores/threads of the device
 - Work : computations + data residing on the device
 - Currently supported devices CPU+pthreads, CUDA
- Classic Multidimensional Arrays, *with a twist*
 - Map multi-index (i,j,k,...) onto memory location *on the device*
 - Should be efficient for both memory used and time to compute
 - Map is derived from a Layout
 - Choose Layout for device-specific access pattern requirements
 - Layout must change when porting among devices
 - Layout changes are transparent to the user code;
 - IF the user code honors the simple array API: $a(i,j,k,\dots)$

Programming Model Implementation

- Standard C++ Library, not a Language extension
 - In *spirit* of Intel's TBB, NVIDIA's Thrust & CUSP, MS C++AMP, ...
 - Not a language extension like OpenMP, OpenACC, OpenCL, CUDA
- Template Meta-Programming
 - For device-specializations and array layout polymorphism
 - C++1998 standard (would really be nice to have C++2011)
- Extremely Attentive to:
 1. Portability – the project charter R&D constraint
 2. Performance – the project charter R&D objective
 3. Usability – the SQE objective

Current Capabilities

- Multidimensional Arrays
 - Declare dimensions and access data members
 - Allocate and deallocate in Device memory space
 - Deep-copy data between host and device memory space
 - Optionally choose or define your own Layout
- Parallel-For and Parallel-Reduce
 - Define thread-parallel work functors (function + data)
 - Dispatch work to device
 - Optionally wait for dispatched work to complete
 - Reduction is guaranteed deterministic, given same # of threads
- Research: Task-Parallelism, Pipeline-Parallelism



Status: Kokkos

- parallel_for, parallel_reduce covered.
- Covers 90+% of SLOC, especially in apps.
 - FEM, FDM, FVM construction, except final assemble.
- Important issues:
 - Data placement, ownership.
 - Construction and view APIs.
 - Extensibility (RTI).

FELIX_ViscosityFO_Def.hpp

```
for (std::size_t cell=0; cell < workset.numCells; ++cell) {
  for (std::size_t qp=0; qp < numQPs; ++qp) {
    //evaluate non-linear viscosity, given by Glen's law, at quadrature points
    epsilonEqpSq = Ugrad(cell,qp,0,0)*Ugrad(cell,qp,0,0); //epsilon_xx^2
    epsilonEqpSq += Ugrad(cell,qp,1,1)*Ugrad(cell,qp,1,1); //epsilon_yy^2
    epsilonEqpSq += Ugrad(cell,qp,0,0)*Ugrad(cell,qp,1,1); //epsilon_xx*epsilon_yy
    epsilonEqpSq += 1.0/4.0*(Ugrad(cell,qp,0,1) + Ugrad(cell,qp,1,0))*(Ugrad(cell,qp,0,1) + Ugrad(cell,qp,1,0)); //epsilon_xy^2
    epsilonEqpSq += 1.0/4.0*Ugrad(cell,qp,0,2)*Ugrad(cell,qp,0,2); //epsilon_xz^2
    epsilonEqpSq += 1.0/4.0*Ugrad(cell,qp,1,2)*Ugrad(cell,qp,1,2); //epsilon_yz^2
    epsilonEqpSq += ff; //add regularization "fudge factor"
    mu(cell,qp) = factor*pow(epsilonEqpSq, power); //non-linear viscosity, given by Glen's law
  }
}
```

Viscosity Kokkos kernel

```
template < typename ScalarType, class DeviceType >
class Viscosity {
  Array2 mu_;
  Array4 U_;
  int numQPs_;
  ScalarType ff_;
  ScalarType factor_;
  ScalarType power_;

public:
  typedef DeviceType device_type;

  Viscosity (Array2 &mu,
            Array4 &u,
            int numQPs,
            ScalarType ff,
            ScalarType factor,
            ScalarType power)
    : mu_(mu)
    , U_(u)
    , numQPs_(numQPs)
    , ff_(ff)
```

```
    , factor_(factor)
    , power_(power){}

  KOKKOS_INLINE_FUNCTION
  void operator () (std::size_t i) const
  {
    ScalarType ep=0.0;
    for (std::size_t j=0; j<numQPs_; j++)
    {
      ep=U_(i, j,0,0)*U_(i,j,0,0);
      ep +=U_(i, j,1,1)*U_(i,j,1,1);
      ep +=U_(i, j,0,0)*U_(i,j,1,1);
      ep +=1.0/4.0*(U_(i, j,0,1)+ U_(i,j,1,0))*(U_(i,j,0,1)+U_(i,j,1,0));
      ep +=1.0/4.0*U_(i,j,0,2)*U_(i,j,0,2);
      ep +=1.0/4.0*U_(i,j,1,2)*U_(i,j,1,2);
      ep +=ff_;
      mu_(i,j) = factor_*pow(ep, power_);
    }
  }
};
```



Kokkos::Cuda on Shannon

Viscosity Host_time =	0.654771	Viscosity Device_time =	0.000481
Body Force Host_time =	0.014789	Body Force Device_time =	0.000451
Residual Host_time =	0.636981	Residual Device_time =	0.000536

Kokkos::Threads on Shannon

Viscosity Host_time =	0.69962	Viscosity Device_time =	0.045445
Body Force Host_time =	0.017365	Body Force Device_time =	0.002276
Residual Host_time =	0.565082	Residual Device_time =	0.040913

numThreads =2, numCores =8

Kokkos::OpenMP on Compton (MIC)

Viscosity Host_time =	7.41132	Viscosity Device_time =	0.019931
Body Force Host_time =	1.18717	Body Force Device_time =	0.010295
Residual Host_time =	35.458	Residual Device_time =	0.130741

numThreads =4, numCores =56
numCells=10000, numWorkSet=100



The Application-Solver Interface

- Next-generation solvers are not just about scalable algorithms.
- Problem construction: Setting up $Ax=b$.
 - 1% – 50% of total execution time for MPI-only
 - Amdahl's Law (without some change): 50% – 99%.
- Performance requirements for $Ax=b$ setup:
 - Vectorized.
 - Threaded.
 - Data properly placed.
 - Repeatable: Inspector-Executor pattern.



Future Requirements (for all of us)

- Your code should either:
 - Spawn threads or
 - Be thread-safe or
 - Both.
- Your code should use KokkosArray, or something like.
 - Array of structs vs. struct of arrays: false choice.
 - Need to abstract.
 - Alternative(?) Array-of-structs-of-arrays.

Sparse Kernels



Sparse Kernels Requirements

- Efficient sparse kernels are essential for manycore computations.
- Kokkos Experience: Generic kernels insufficient.
- Inspector-executor model is essential:
 - We often solve a *family* of sparse systems whose:
 - Patterns are identical.
 - Values change.
 - Inspection phase builds a “plan”:
 - Performs work related to pattern-only issues:
 - Communication patterns.
 - Permutations.
 - Executor: Executes plan for given set of values.
 - Update values: Replace old values with new.



Sparse Kernel Design

- Sparse Matrix Graph:
 - 1st class object for inspector-executor.
 - Sharable between multiple matrices.
- Two sources for sparse kernels:
 - Pretty good kernel library (PGKlib).
 - Pretty-good threaded (OpenMP, CUDA) kernels.
 - Similar to Traditional approaches for serial.
 - Goal (achieved): Reach achievable bandwidth.
 - How achieved?: KokkosArray.
 - *Very* easy plug-in mechanism for vendor libraries.
 - Especially for sparse kernels, dense non-trivial.



Leveraging Vendor Libraries

- Lots of discussion about autotuning sparse computations.
- Less-frequently mentioned: Vendor-tuned kernels.
- Computer system vendors (Intel, Nvidia, Cray) are writing node-level optimal kernels: SpMV, BLAS, SpSV, ...
- Virtuous circle:
 - Vendor kernel is optimal.
 - (Demmel) student beats vendor implementation.
 - Vendor improves their kernel.
- Reaction:
 - Make integration of third-party kernels simple.
 - Again, inspector-executor pattern support is essential.



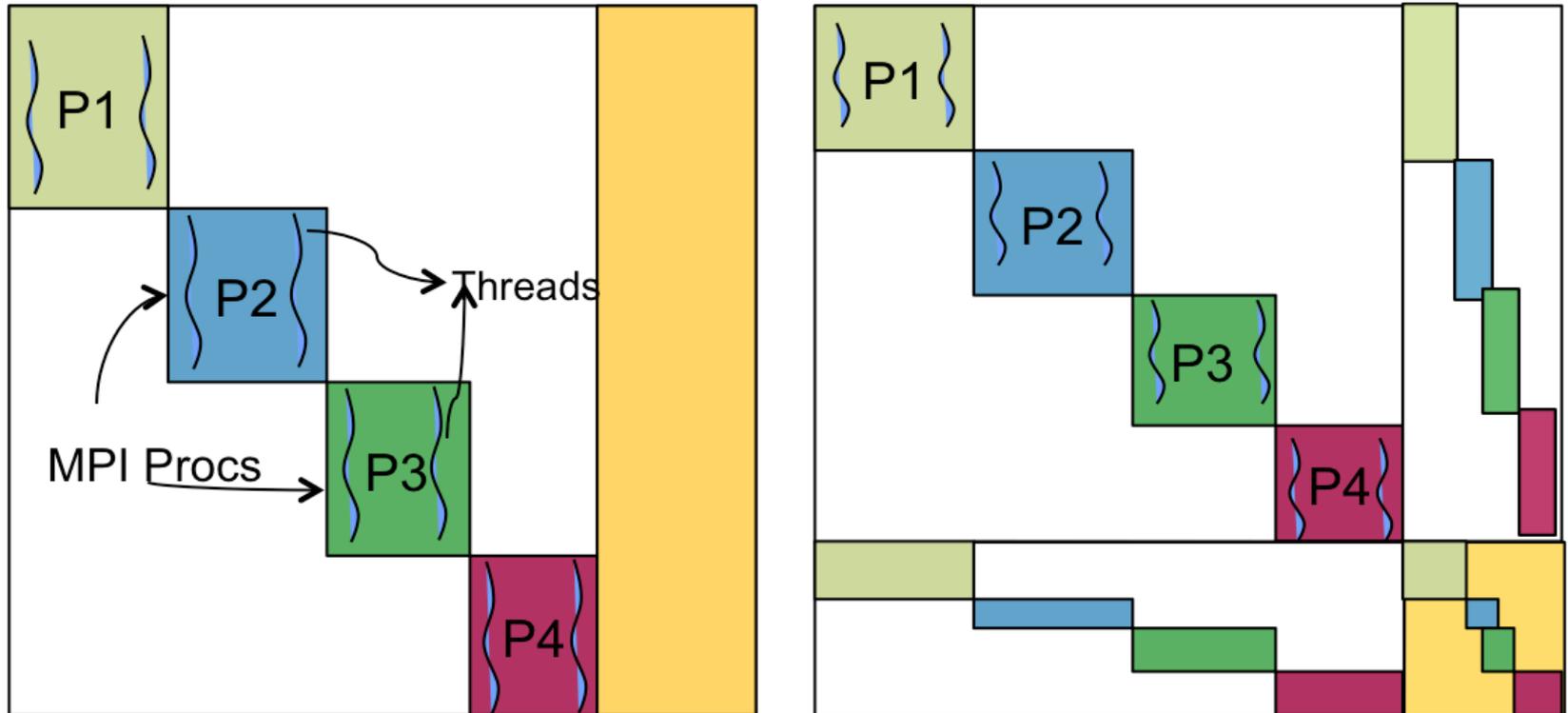
Manycore Smoothers



Manycore Smoothers: Essential for Scalable Solvers

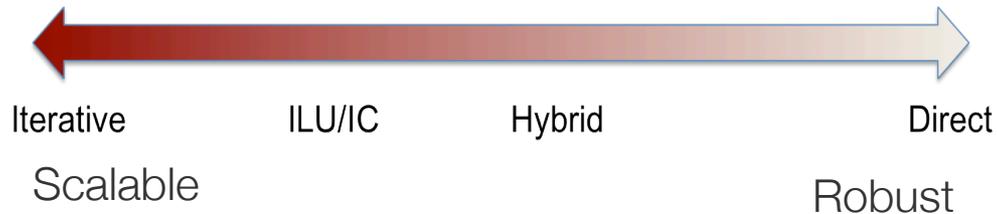
- Migration to MPI+X (or something like it) is assumed.
- Multi-level preconditioners essential for constraining iteration growth.
- Natural decomposition: 1 subdomain per manycore node.
- Requirement: Need manycore smoothers.

Algebraic Schur Complement Framework



$$Sx_2 = (A_{22} - A_{21}A_{11}^{-1}A_{12})x_2 = b_2 - A_{21}A_{11}^{-1}b_1$$

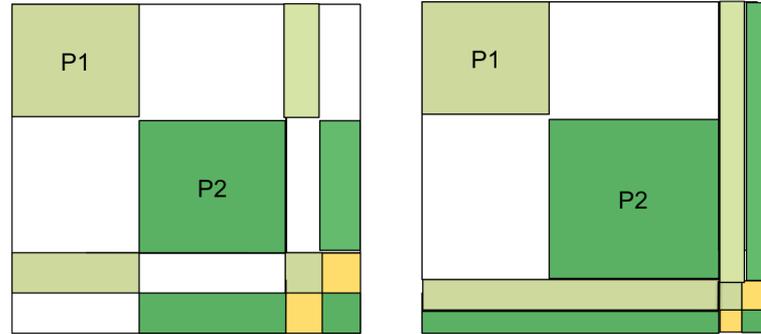
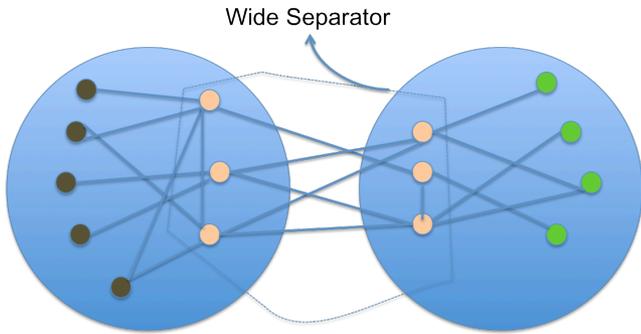
ShyLU Overview



- ShyLU (Scalable Hybrid LU) is hybrid:
 - In the mathematical sense (direct + iterative) for robustness
 - In the parallel programming sense (MPI + Threads) for scalability
- More robust than simple preconditioners and scalable than direct solvers
- ShyLU is a subdomain solver where a subdomain is not limited to
- one MPI process
- Multithreaded direct factorization for the block diagonals.
- Compute block diagonals with wide or narrow separators
- Approximate Schur complement with dropping or probing
- Use it as an Ifpack preconditioner

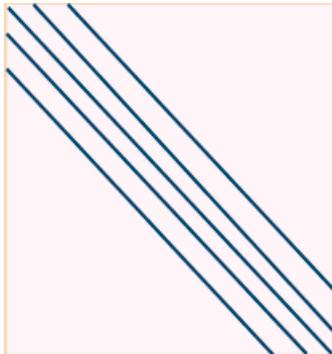
ShyLU Internals

Wide Separators: Scalable, but larger Schur complement

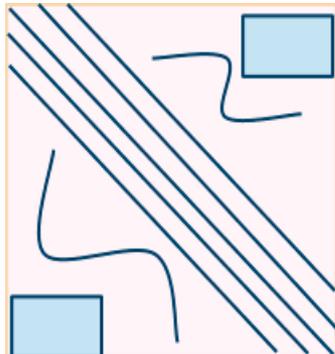


Structure of approximate Schur complement

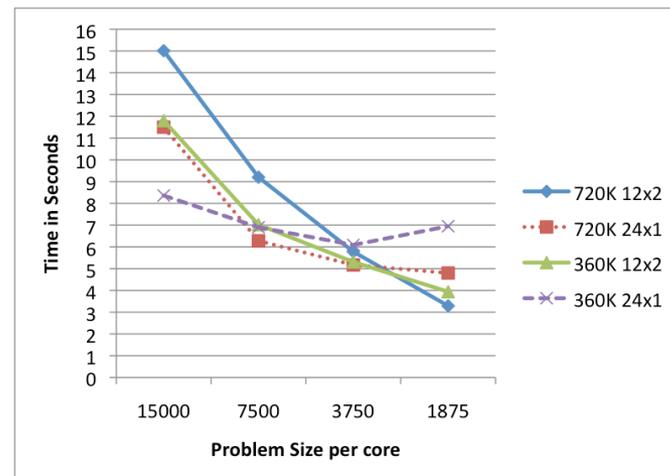
Original Probing
Chen, Mathew
1991



ShyLU's Probing,
Band + nnz(A)

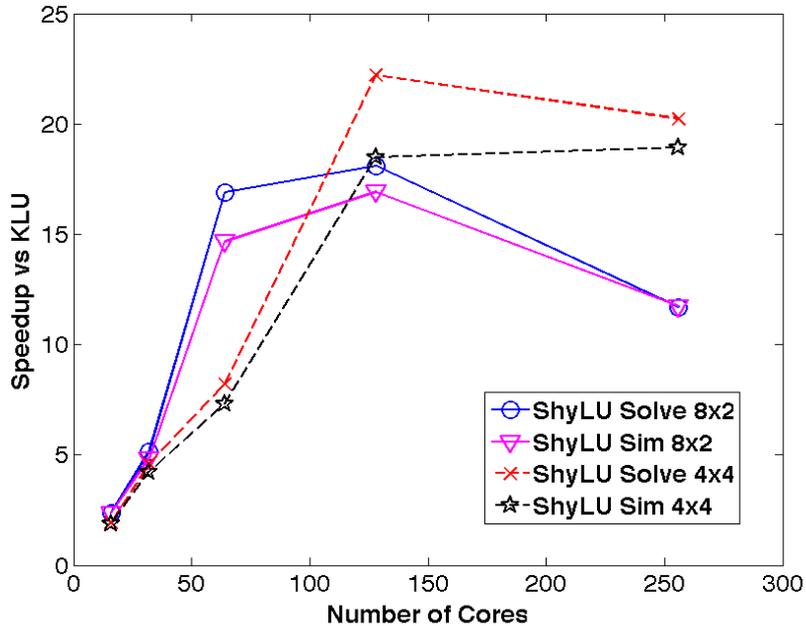


Smaller problems are better for
the threads



Simulation of Circuit problems

Speedup of solver & simulation times



- Parallel direct solvers fail with this linear system
- Simulation with *KLU* takes over a week (20 hs or more).

Need a robust, parallel solver

- ShyLU provides 19x speedup in the total simulation time (w/ 16 nodes)
- Simulation with ShyLU completes in few hours.
- More than 64 cores was infeasible with MPI. Hybrid parallelism helped in utilizing more cores.

Solver	ShyLU	KLU	SuperLU	SuperLU_Dist	PARDISO
Simulation time for 1ns run	10.67 mins	3.95 hrs	20.04 hrs	-	-

Co-Design for Resilience



Our Luxury in Life (wrt FT/Resilience)

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



Paradigm Shift is Coming

Fault rate is growing exponentially therefore faults will eventually become continuous.

Faults will be continuous and across all levels from HW to Apps (no one level can solve the problem -- solution must be holistic)

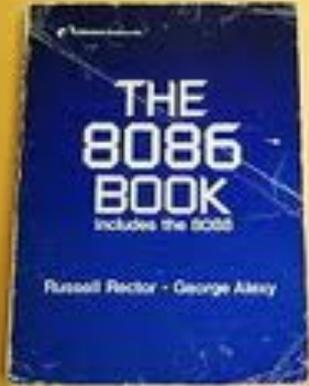
Expectations should be set accordingly with users and developers

Self-healing system software and application codes needed

Development of such codes requires a fault model and a framework to test resilience at scale

Validation in the presence of faults is critical for scientists to have faith in the results generated by exascale systems

Resilience Trends Today: An X86 Analogy



- Published June 1980
- Sequential ISA.
- Preserved today.
- Illusion:
 - Out of order exec.
 - Branch prediction.
 - Shadow registers.
 - ...
- Cost: Complexity, energy.



Global checkpoint restart

- Preserve the illusion:
 - reliable digital machine.
 - CP/R model: Exploit latent properties.
- SCR: Improve performance 50-100%.
- NVRAM, etc.
- More tricks are still possible.
- End game predicted many times.

Resilient applications

- Expose the reality:
 - Fault-prone analog machine.
 - New fault-aware approaches.
- New models:
 - Programming, machine, execution.
- New algorithms:
 - Relaxed BSP.
 - LFLR.
 - Selective reliability.



Resilience Problems: Already Here, Already Being Addressed, Algorithms & Co-design Are Key

- Already impacting performance: Performance variability.
 - HW fault prevention and recovery introduces variability.
 - Latency-sensitive collectives impacted.
 - MPI non-blocking collectives + new algorithms address this.
- Localized failure:
 - Now: local failure, global recovery.
 - Needed: local recovery (via persistent local storage).
 - MPI FT features + new algorithms: Leverage algorithm reasoning.
- Soft errors:
 - Now: Undetected, or converted to hard errors.
 - Needed: Apps handle as performance optimization.
 - MPI reliable messaging + PM enhancement + new algorithms.
- *Key to addressing resilience: algorithms & co-design.*

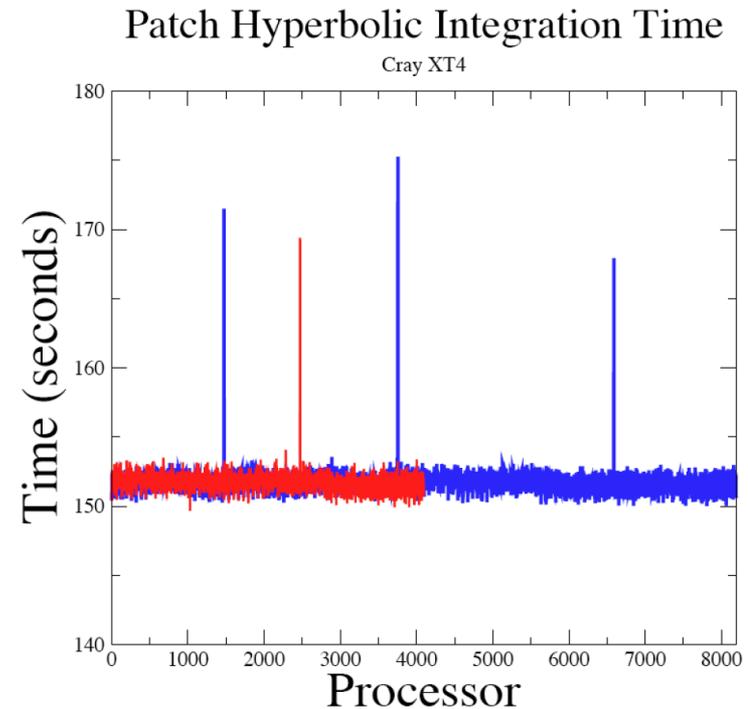


Four Resilient Programming Models

- Relaxed Bulk Synchronous (rBSP)
- Local-Failure, Local-Recovery (LFLR)
- Skeptical Programmer (SP)
- Selective (Un)reliability (SU/R)

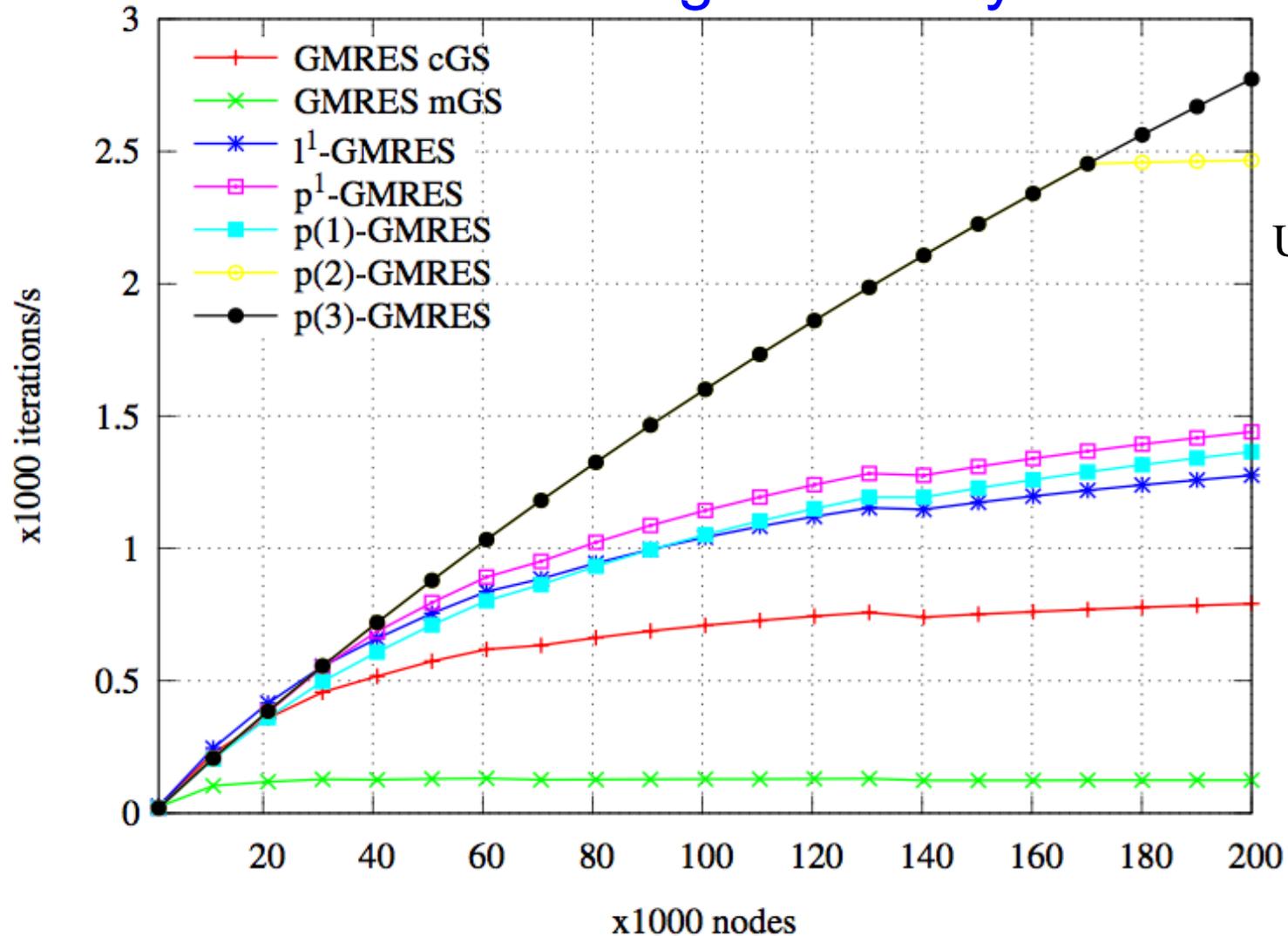
Resilience Issues Already Here

- First impact of unreliable HW?
 - Vendor efforts to hide it.
 - Slow & correct vs. fast & wrong.
- Result:
 - Unpredictable timing.
 - Non-uniform execution across cores.
- Blocking collectives:
 - $t_c = \max_i \{t_i\}$
- Also called “Limpware”:
 - Haryadi Gunawi, University of Chicago
 - <http://www.anl.gov/events/lights-case-limping-hardware-tolerant-systems>



Brian van Straalen, DOE Exascale Research Conference, April 16-18, 2012. *Impact of persistent ECC memory faults.*

Latency-tolerant Algorithms + MPI 3: Recovering scalability



Hiding global communication latency in the GMRES algorithm on massively parallel machines,

P. Ghysels T.J. Ashby K. Meerbergen W. Vanroose, Report 04.2012.1, April 2012,



What is Needed to Support Latency Tolerance?

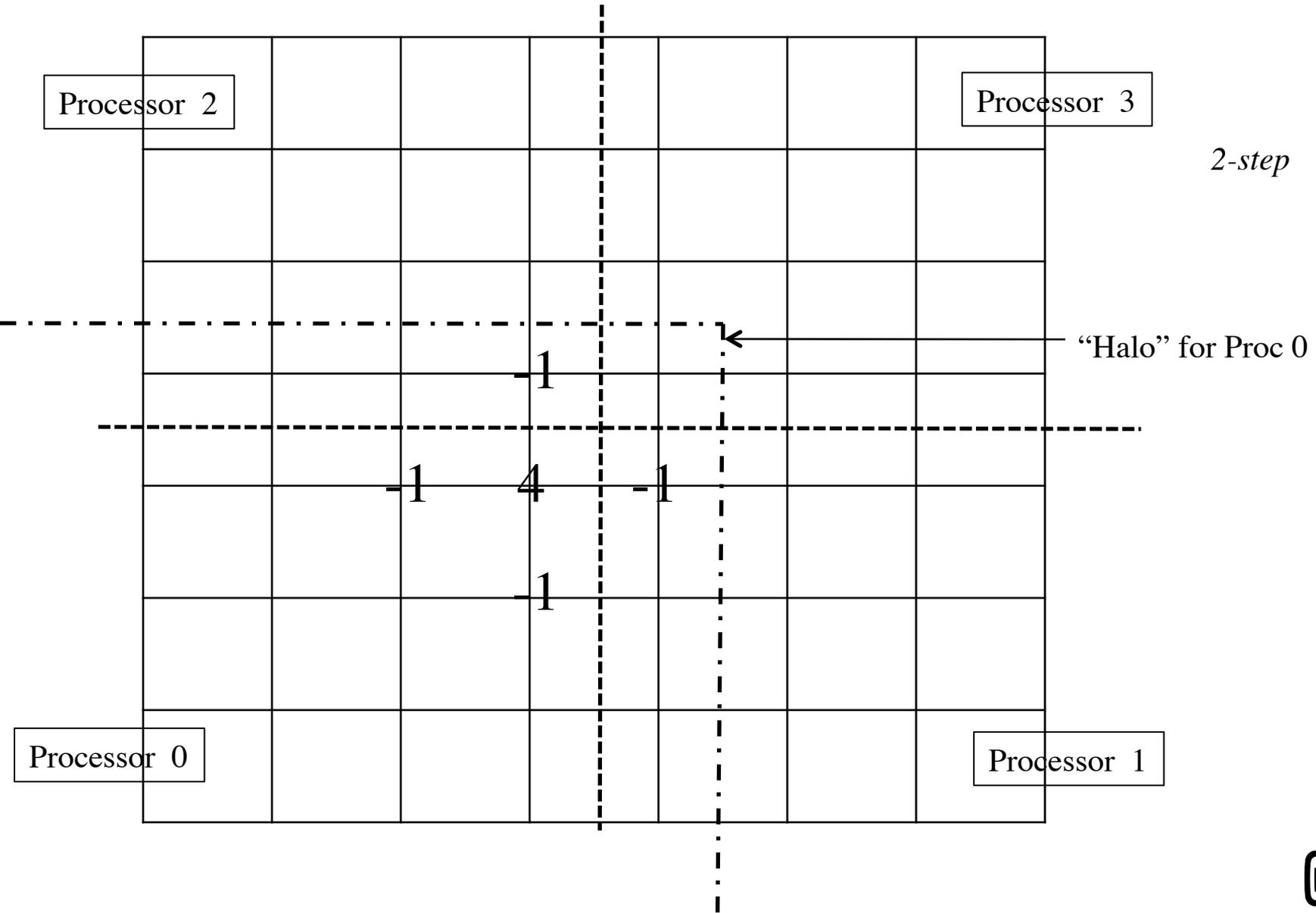
- MPI 3 (SPMD):
 - Asynchronous global and neighborhood collectives.
- A “relaxed” BSP programming model:
 - Start a collective operation (global or neighborhood).
 - Do “something useful”.
 - Complete the collective.
- The pieces are coming online.
- With new algorithms we can recover some scalability.



“Communication-Avoiding” Methods Challenges

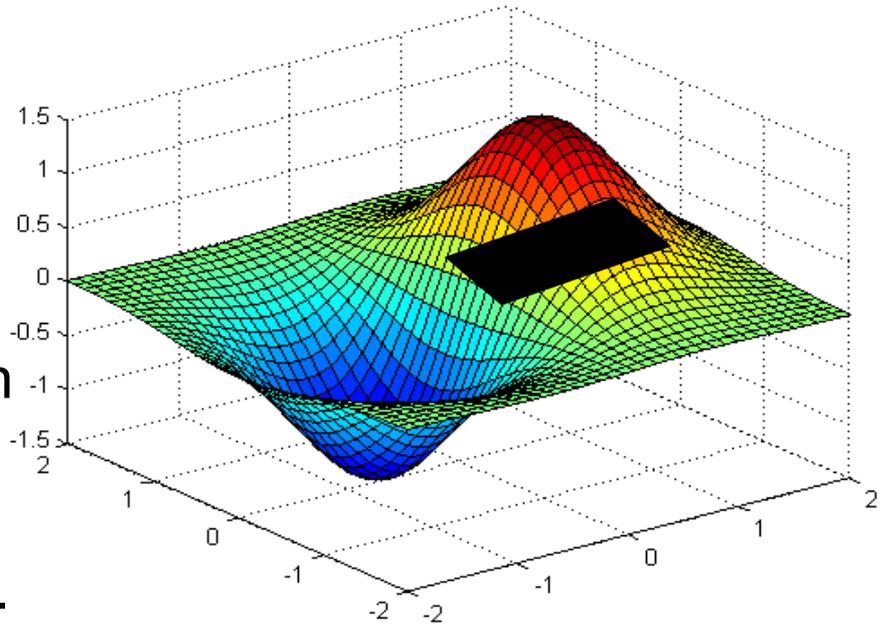
- Popular, been around a long time:
 - s -step iterative methods.
 - Simultaneous execution of s Krylov iterations.
- Biggest issue: Compute (prec) matrix-powers kernel.
 - $(Ax, A(Ax), A(A(Ax)), \dots)$
 - Fewer, larger data transfers, redundant computing.
- Old numerical issue: Behaves like the power method.
 - Fix: Use a different basis.
 - CA iterative methods now work well for “easy” linear systems.
- Current issues: Hard to use with non-trivial preconditioners.
 - $(AMx, AM(AMx), AM(AM(AMx)), \dots)$
 - Current “black box” preconditioner interface not compatible.
 - $z = Mr$, Preconditioner takes a vector r and produces z .

2D PDE on Regular Grid (Standard Laplace)



Enabling Local Recovery from Local Faults

- Current recovery model:
Local node failure,
global kill/restart.
- Different approach:
 - App stores key recovery data in persistent local (per MPI rank) storage (e.g., buddy, NVRAM), and registers recovery function.
 - Upon rank failure:
 - MPI brings in reserve HW, assigns to failed rank, calls recovery fn.
 - App restores failed process state via its persistent data (& neighbors’?).
 - All processes continue.





LFLR Algorithm Opportunities & Challenges

- Enables fundamental algorithms work to aid fault recovery:
 - Straightforward app redesign for explicit apps.
 - Enables reasoning at approximation theory level for implicit apps:
 - What state is required?
 - What local discrete approximation is sufficiently accurate?
 - What mathematical identities can be used to restore lost state?
 - Enables practical use of many exist algorithms-based fault tolerant (ABFT) approaches in the literature.



What is Needed for Local Failure Local Recovery (LFLR)?

- LFLR realization is non-trivial.
- Programming API (but not complicated).
- Lots of runtime/OS infrastructure.
 - Persistent storage API (frequent brainstorming outcome).
- Research into messaging state and recovery.
- New algorithms, apps re-work.
- But:
 - Can leverage global CP/R logic in apps.
- This approach is often considered next step in beyond CP/R.

First LFLR Example

- Prototype LFLR Transient PDE solver.
- Simulated process lost.
- Simulated persistent store.
- Over-provisioned MPI ranks.
- Manual process kill.

Data/work recovery time

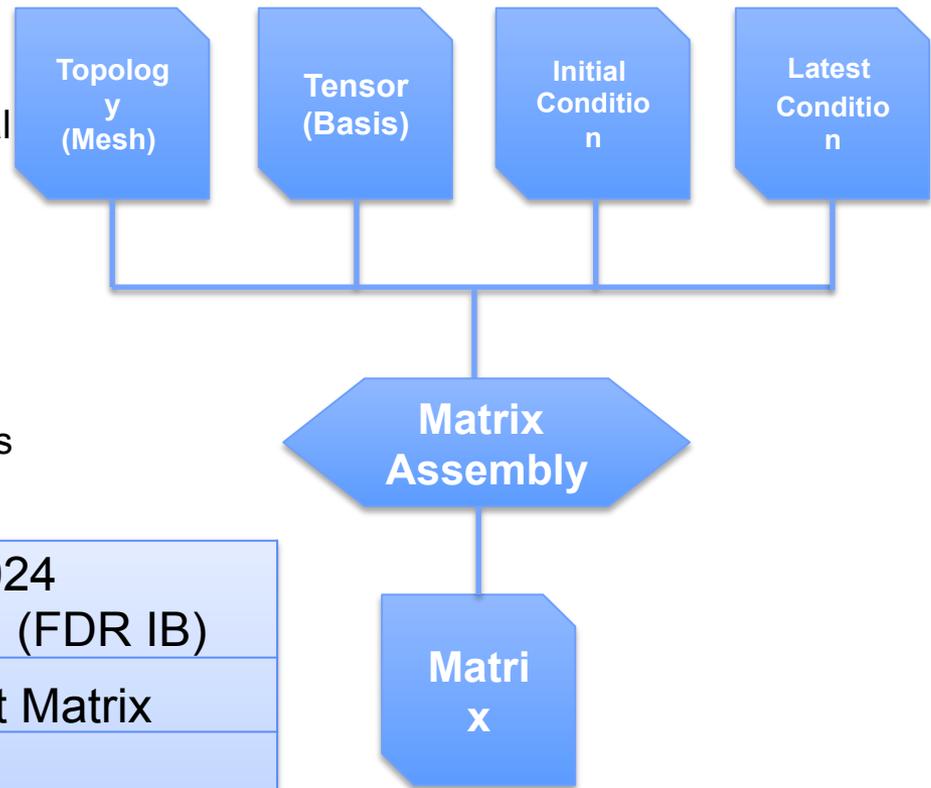
Persistent store time

# of Processes	CG	READ	WRITE	ALL
4	2.64	0.008	0.01	2.77
8	5.39	0.09	0.012	5.83
16	7.84	0.008	0.013	7.99
32	9.9	0.008	0.014	10.04
64	12.56	0.009	0.0145	12.76
128	16.99	0.0085	0.015	17.14
256	21.6	0.009	0.016	21.76
512	28.75	0.009	0.015	28.91

Results from explicit variant of Mantevo/MiniFE, Keita Teranishi

Data Recovery from Computation

- Lots of scientific objects are dependent on more compact data objects
 - Higher abstraction of mathematical model
- Can be recovered through inexpensive computation
 - 90%+ storage reduction in miniFE
 - Some refactoring in scientific objects
 - Put them “recoverable” subclass
 - Increase roll-back overhead



miniFE: 512x512x512: 1024 SandyBridge CPU Cores (FDR IB)		
	With Matrix	Without Matrix
Storage per core	53.94 MB	2.1 MB
Regenerate overhead	(in memory) 0.1 sec (in global file system) 5 sec+	(in memory + compute) 0.6 sec

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.

Skeptical Programming

I might not have a reliable digital machine

- Expect rare faulty computations
- Use analysis to derive cheap “detectors” to filter large errors
- Use numerical methods that can absorb *bounded error*

Algorithm 1: GMRES algorithm

```
for  $l = 1$  to do
   $\mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x}^{(j-1)}$ 
   $\mathbf{q}_1 := \mathbf{r} / \|\mathbf{r}\|_2$ 
  for  $j = 1$  to restart do
     $\mathbf{w}_0 := \mathbf{A}\mathbf{q}_j$ 
    for  $i = 1$  to  $j$  do
       $h_{i,j} := \mathbf{q}_i \cdot \mathbf{w}_{i-1}$ 
       $\mathbf{w}_i := \mathbf{w}_{i-1} - h_{i,j}\mathbf{q}_i$ 
    end
     $h_{j+1,j} := \|\mathbf{w}\|_2$ 
     $\mathbf{q}_{j+1} := \mathbf{w} / h_{j+1,j}$ 
    Find  $\mathbf{y} = \min \|\mathbf{H}_j\mathbf{y} - \|\mathbf{b}\| \mathbf{e}_1\|_2$ 
    Evaluate convergence criteria
    Optionally, compute  $\mathbf{x}_j = \mathbf{Q}_j\mathbf{y}$ 
  end
end
```

GMRES

Theoretical Bounds on the Arnoldi Process

$$\|\mathbf{w}_0\| = \|\mathbf{A}\mathbf{q}_j\| \leq \|\mathbf{A}\|_2 \|\mathbf{q}_j\|_2$$

$$\|\mathbf{w}_0\| \leq \|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F$$

From isometry of orthogonal projections,

$$|h_{i,j}| \leq \|\mathbf{A}\|_F$$

- h_{ij} form Hessenberg Matrix
- Bound only computed once, valid for entire solve

FT-GMRES Algorithm

Input: Linear system $Ax = b$ and initial guess x_0

$r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, $q_1 := r_0/\beta$

for $j = 1, 2, \dots$ until convergence **do**

Inner solve: Solve for z_j in $q_j = Az_j$

$v_{j+1} := Az_j$

for $i = 1, 2, \dots, k$ **do**

$H(i, j) := q_i^* v_{j+1}$, $v_{j+1} := v_{j+1} - q_i H(i, j)$

end for

$H(j+1, j) := \|v_{j+1}\|_2$

Update rank-revealing decomposition of $H(1:j, 1:j)$

if $H(j+1, j)$ is less than some tolerance **then**

if $H(1:j, 1:j)$ not full rank **then**

Try recovery strategies

else

Converged; return after end of this iteration

end if

else

$q_{j+1} := v_{j+1}/H(j+1, j)$

end if

$y_j := \operatorname{argmin}_y \|H(1:j+1, 1:j)y - \beta e_1\|_2$ \triangleright GMRES projected problem

$x_j := x_0 + [z_1, z_2, \dots, z_j]y_j$ \triangleright Solve for approximate solution

end for

“Unreliably” computed.

Standard solver library call.

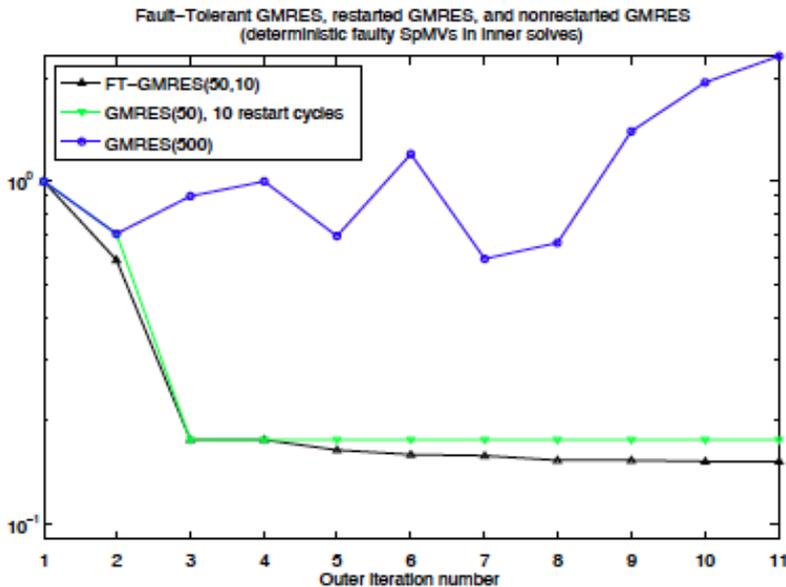
Majority of computational cost.

\triangleright Orthogonalize v_{j+1}

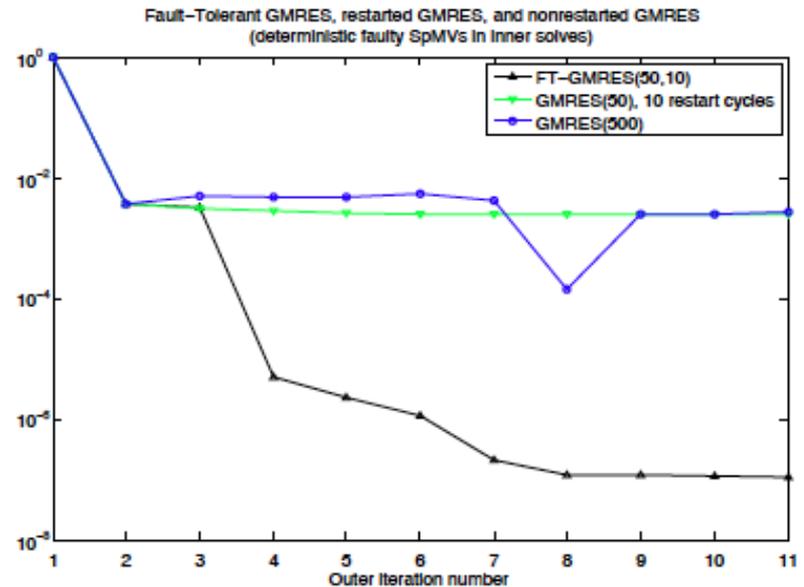
Captures true linear operator issues, AND
Can use some “garbage” soft error results.

Selective reliability enables “running through” faults

- ▶ FT-GMRES can run through faults and still converge.
- ▶ Standard GMRES, with or without restarting, cannot.



FT-GMRES vs. GMRES on Ill_Stokes (an ill-conditioned discretization of a Stokes PDE).



FT-GMRES vs. GMRES on mult_dcop_03 (a Xyce circuit simulation problem).



Desired properties of FT methods

- Converge eventually
 - No matter the fault rate
 - Or it detects and indicates failure
 - Not true of iterative refinement!
- Convergence degrades gradually as fault rate increases
 - Easy to trade between reliability and extra work
- Requires as little reliable computation as possible
- Can exploit fault detection if available
 - e.g., if no faults detected, can advance aggressively



Selective Reliability Programming

- Standard approach:

- System over-constrains reliability
- “Fail-stop” model
- Checkpoint / restart
- Application is ignorant of faults

- New approach:

- System lets app control reliability
- Tiered reliability
- “Run through” faults
- App listens and responds to faults



What is Needed for Selective Reliability?

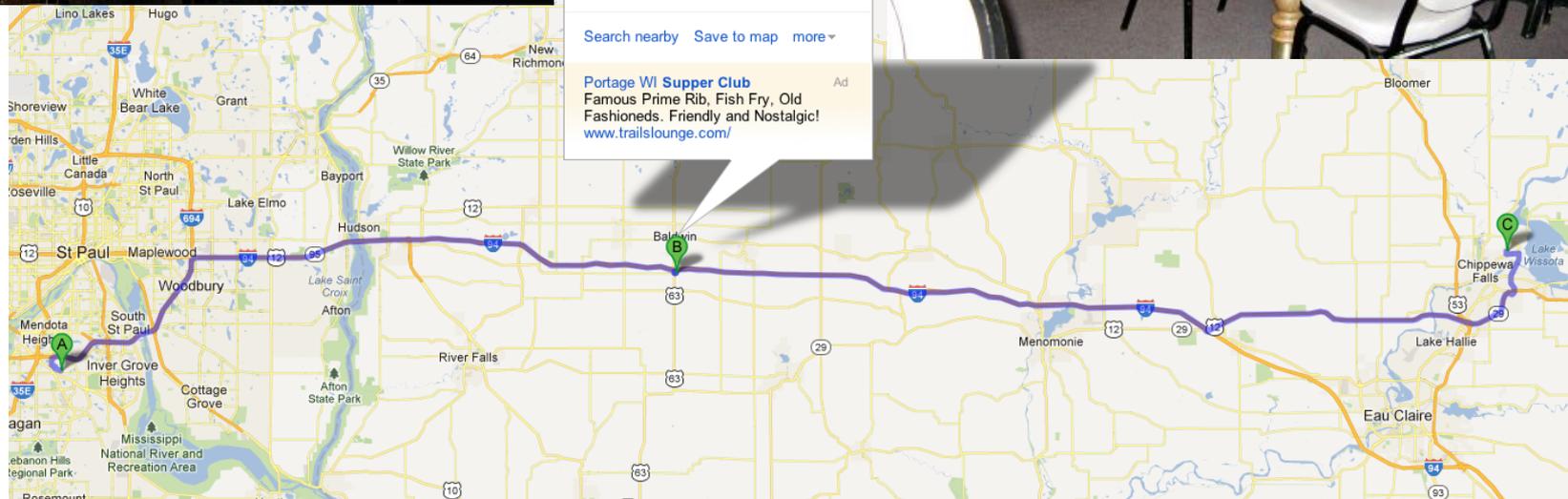
- A lot, lot.
- A programming model.
- Algorithms.
- Lots of runtime/OS infrastructure.
- Hardware support?



Comments on New Algorithm R&D

Co-Design Cray-style (circa 1996)

(This is not a brand new idea)





Algorithm Success & System Design

- Better latency tolerant algorithms permit:
 - More error detection/correction w/o harming scalability.
 - Cheaper networks, especially for collectives.
- LFLR:
 - Best ways to achieve persistent storage?
 - Reduce or eliminate high speed global file system.
- Skeptical programming:
 - Simple approach to greatly reduce SDC and bad results.
 - Permit lower reliability HW.
- Selective unreliability:
 - How to program?
 - How to implement high (or low?) reliability.
- Co-design of algorithms and system is very important.

Software Engineering and HPC

Efficiency vs. Other Quality Metrics

Validation

Verification

How focusing on the factor below affects the factor to the right	Correctness	Usability	Efficiency	Reliability	Integrity	Adaptability	Accuracy	Robustness
Correctness	↑		↑	↑			↑	↓
Usability		↑				↑	↑	
Efficiency	↓		↑	↓	↓	↓	↓	
Reliability	↑			↑	↑		↑	↓
Integrity			↓	↑	↑			
Adaptability					↓	↑		↑
Accuracy	↑		↓	↑		↓	↑	↓
Robustness	↓	↑	↓	↓	↓	↑	↓	↑

Source:
Code Complete
Steve McConnell

Helps it ↑

Hurts it ↓

Summary of Current Efforts

- Node-level parallelism is the new commodity curve:
 - Tasks, threads, vectors.
- MPI+X is and will be dominant platform for tera, peta, exascale:
 - Natural fit for many science & engineering apps.
 - Hierarchical composition matches tera, peta and exascale.
 - Naturally leverages industry efforts.
- C++ compile-time polymorphism seems essential.
- Everything must be threaded or thread-safe or both.
- Multifaceted library strategy:
 - Trilinos: Good API + PGKlib.
 - Easy integration of vendor or specialize kernels.
- Manycore smoothers are essential.
- Resilient computing models emerging. Algorithms R&D essential.