Building the Next Generation of Parallel and Resilient Applications and Libraries

Michael A. Heroux
Scalable Algorithms Department
Sandia National Laboratories

Collaborators: Erik Boman, Carter Edwards, Mark Hoemmen, Siva Rajamanickam, Keita Teranishi, Christian Trott, Alan Williams

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

- Parallel Computing Trends and MPI+X.
- Reasoning about Parallelism.
- Programming Languages.
- Resilience.
- Co-Design.

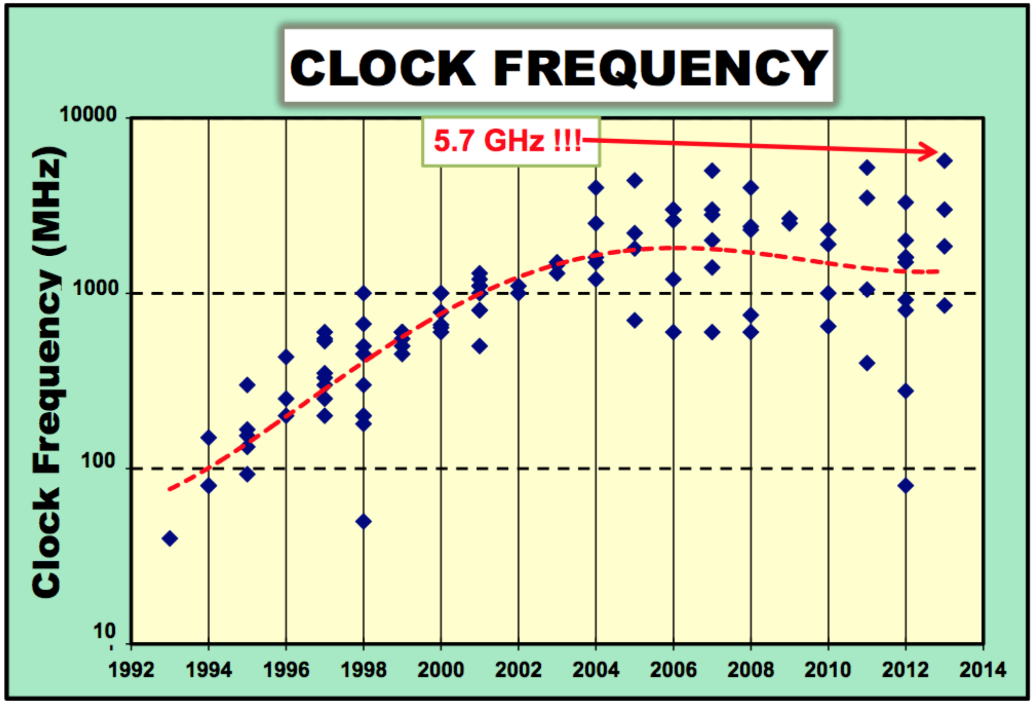
Why It’s not Business as Usual
(Experts, check your email at this time)

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 ⇏ Equal execution time.
    - Across peers or from one run to the next.



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

Sandia National Laboratories

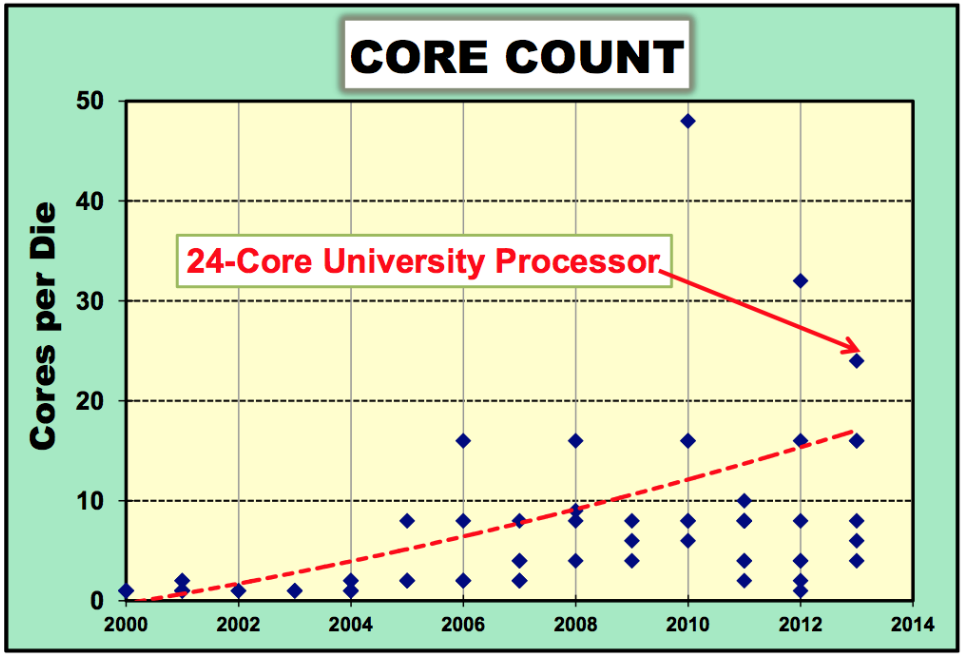
New Commodity Trends and Concerns Emerge

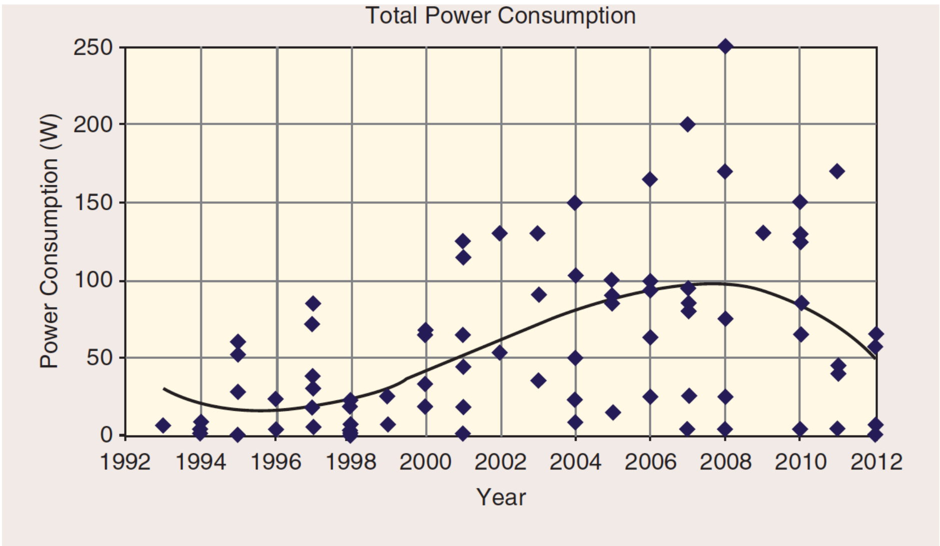
Big Concern: Energy Efficiency.

- 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.





International Solid-State Circuits Conference (ISSCC 2012) Report http://isscc.org/doc/2012/2012_Trends.pdf (top is 2013 report).

The HPC Ecosystem

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

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

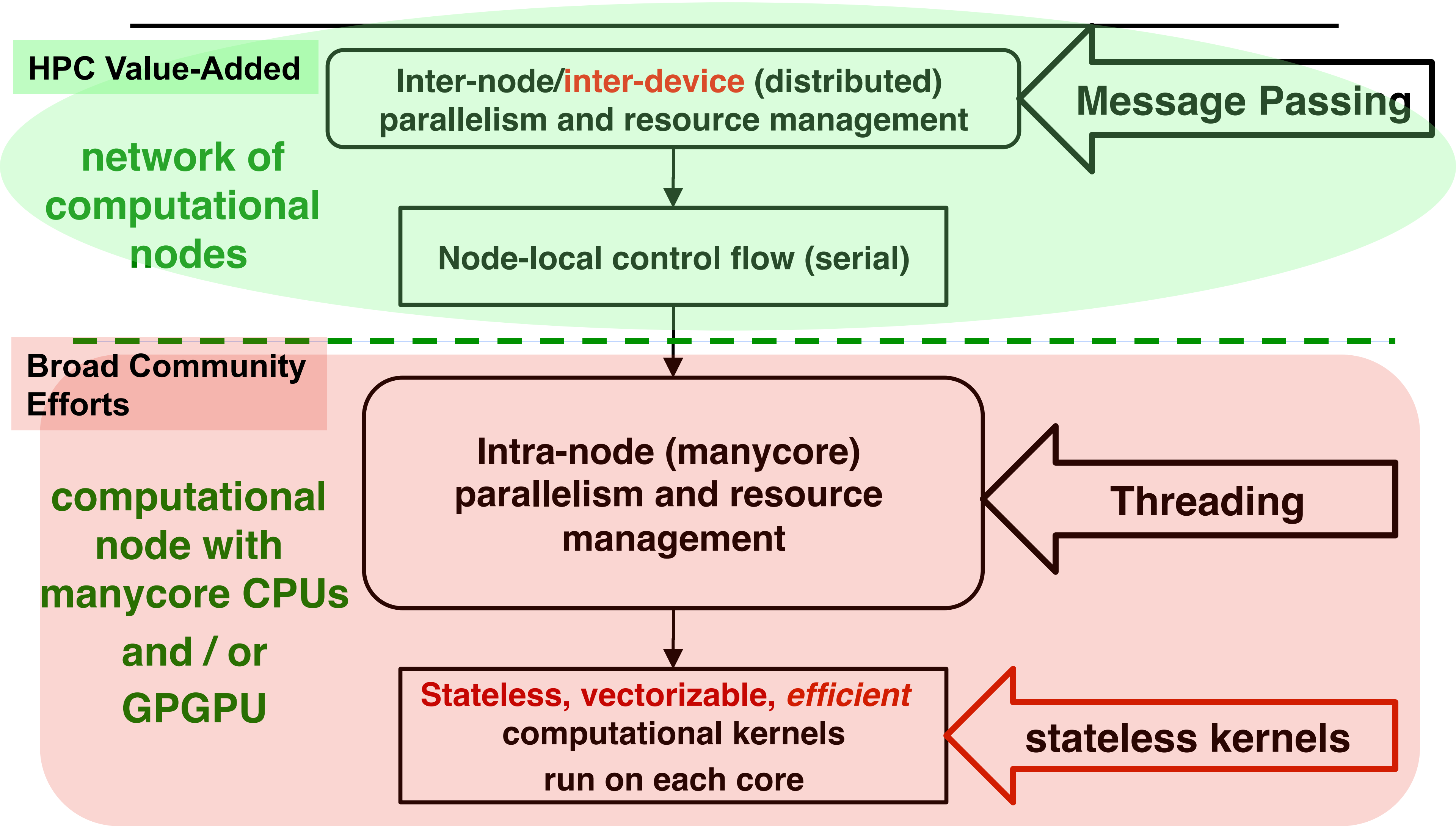
Reasons for SPMD/MPI Success?

- Portability? Standardization? Momentum? Yes.
- Separation of Parallel & Algorithms concerns? Big Yes.
- Preserving & Extending Sequential Code Investment? Big, Big Yes.

- MPI was disruptive, but not revolutionary.
  - A meta layer encapsulating sequential code.
    - Enabled mining of vast quantities of existing code and logic.
  - Sophisticated physics added as sequential code.
    - Ratio of science experts vs. parallel experts: 10:1.
- Key goal for new parallel apps: Preserve these dynamics.

Sandia National Laboratories

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

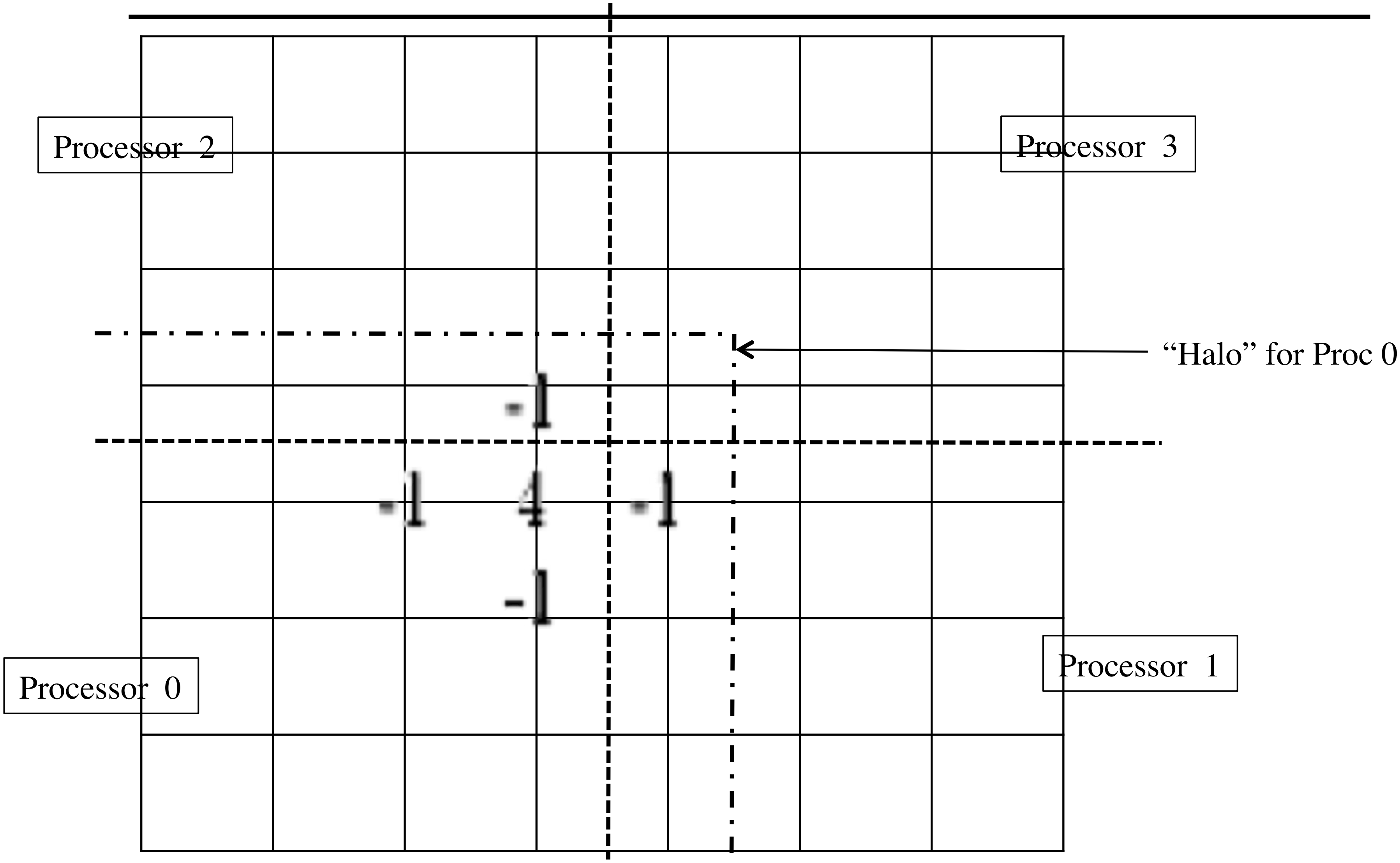


Effective node-level parallelism: First priority

- 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.

Parallel Patterns

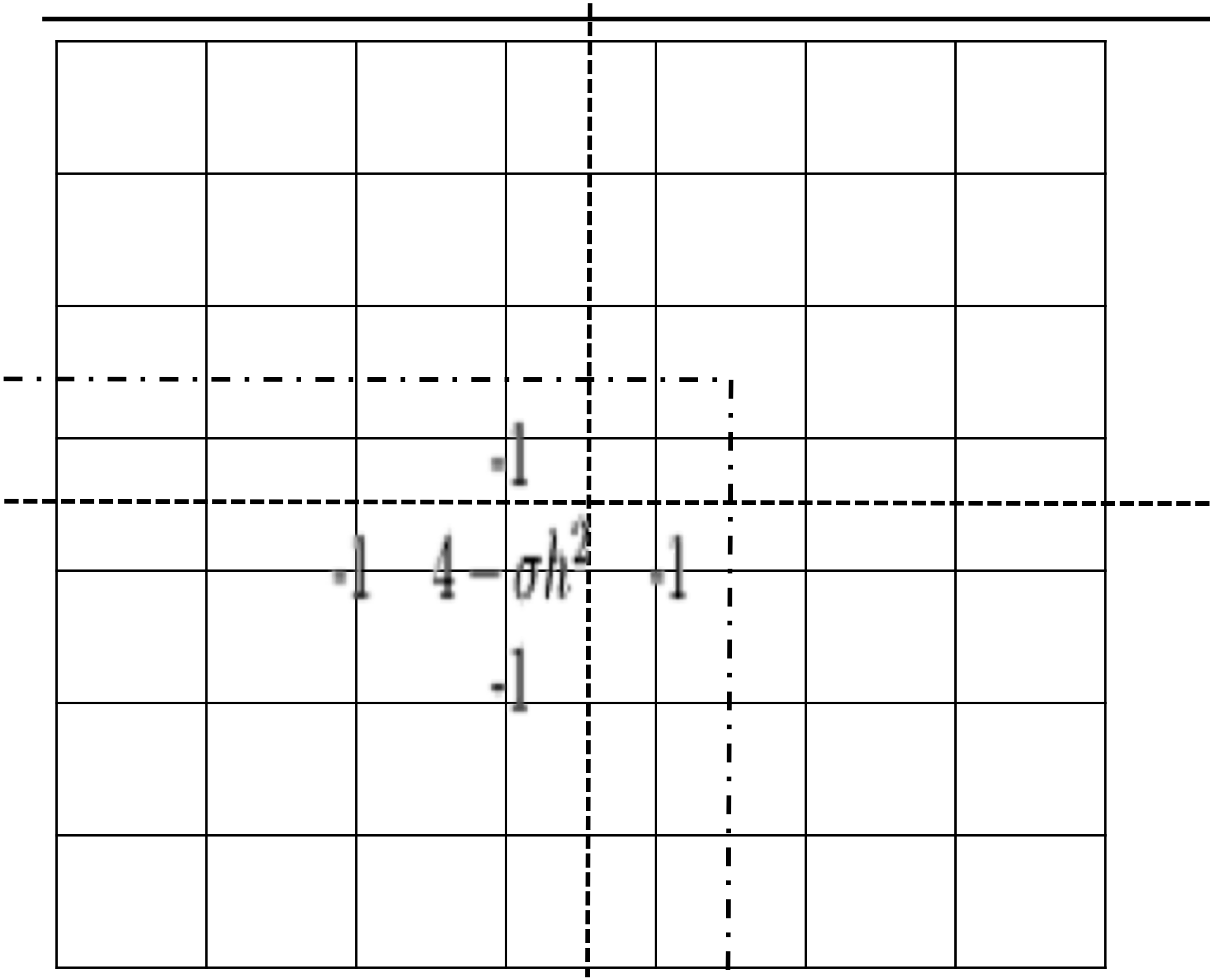
2D PDE on Regular Grid (Standard Laplace)



SPMD Patterns for Domain Decomposition

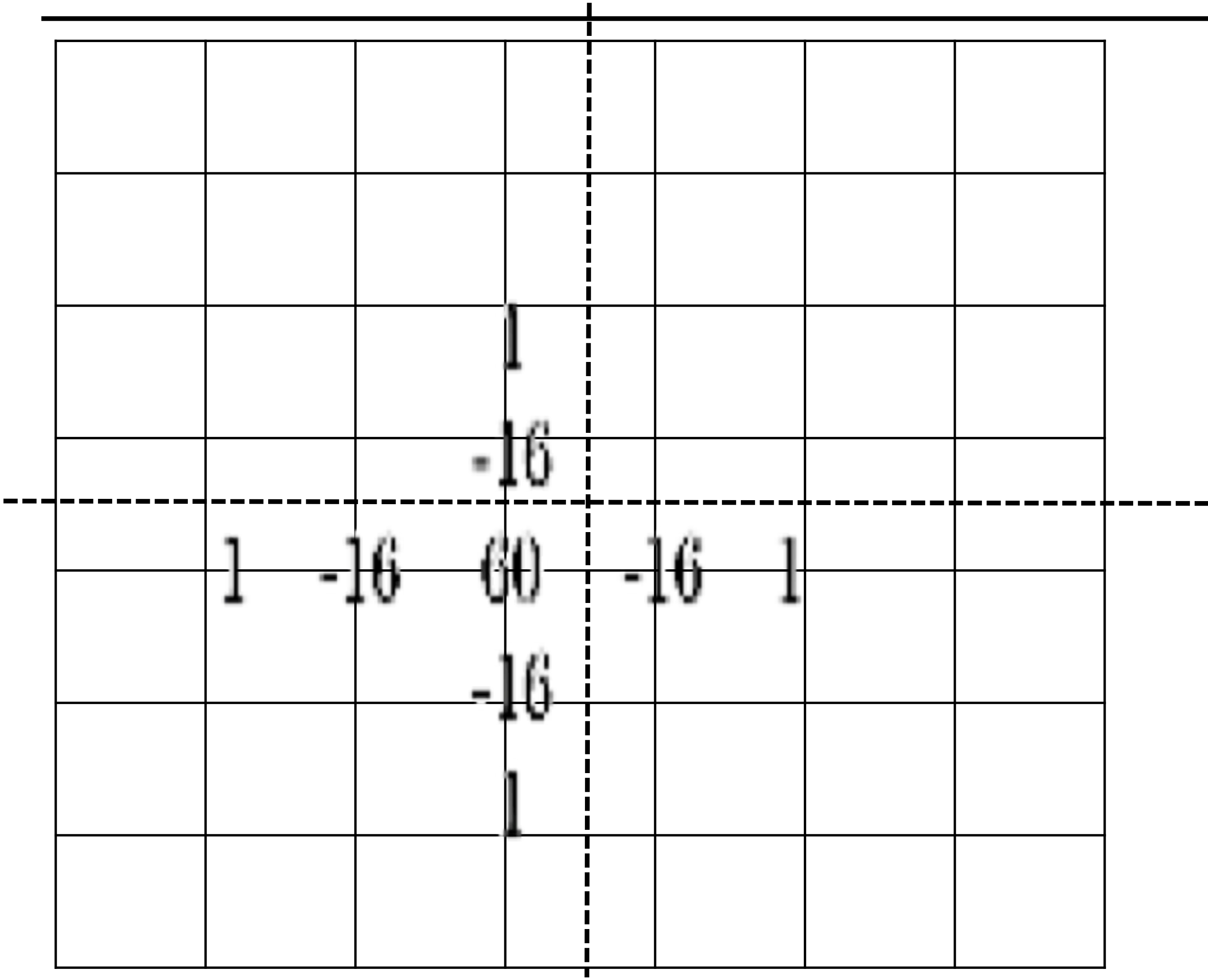
- Single Program Multiple Data (SPMD):
  - Natural fit for many differential equations.
  - All processors execute same code, different subdomains.
  - Message Passing Interface (MPI) is portability layer.
- Parallel Patterns:
  - Halo Exchange:
    - Written by parallel computing expert: Complicated code.
    - Used by domain expert: DoHaloExchange() - Conceptual.
    - Use MPI. Could be replace by PGAS, one-sided, …
  - Collectives:
    - Dot products, norms.
- All other programming:
  - Sequential.
  - Example: 5-point stencil computation is sequential.

2D PDE on Regular Grid (Helmholtz)



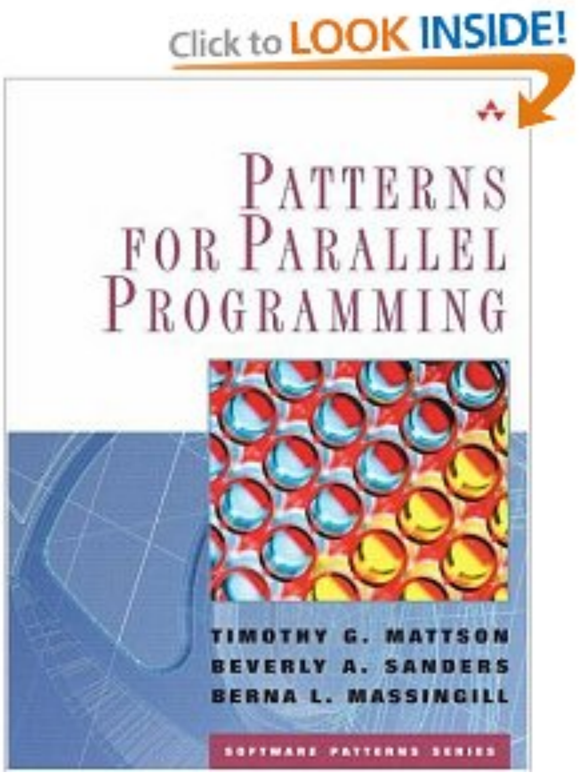
$$-\nabla u - \sigma u = f \qquad (\sigma \geq 0)$$

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



Thinking in Patterns

- First step of parallel application design:
  - Identify parallel patterns.
- Example: 2D Poisson (& Helmholtz)
  - SPMD:
    - Halo Exchange.
    - AllReduce (Dot product, norms).
  - SPMD+X:
    - Much richer palette of patterns.
    - Choose your taxonomy.
    - Some: Parallel-For, Parallel-Reduce, Task-Graph, Pipeline.



Sandia National Laboratories

Thinking in Parallel Patterns

- 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 >>> (…);
- Thrust, …
- Cray Autotasking (April 1989)

```
c.....do parallel SAXPY
CMIC$ DO ALL SHARED(N, ALPHA, X, Y)
CMIC$1   PRIVATE(i)
       do 10 i = 1, n
          y(i) = y(i) + alpha*x(i)
 10    continue
```

Sandia National Laboratories

Why Patterns

- Essential expressions of concurrency.
- Describe constraints.
- Map to many execution models.
- Example: Parallell-for.
  - Can be mapped to SIMD, SIMT, Threads, SPMD.
  - Future: Processor-in-Memory (PIM).
- Lots of ways to classify them.

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.

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

- Characteristics:
  - Multiple threads.
  - Fast barrier.
  - Shared, fast access memory pool.
  - Example: Nvidia SM, Intel MIC
  - X86 more vague, emerging more clearly in future.
- Qualitatively better algorithm:
  - Threaded triangular solve scales.
  - Fewer MPI ranks means fewer iterations, better robustness.
  - Data-driven parallelism.

Sandia National Laboratories

Programming Today for Tomorrow's Machines

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

What we need from Programming Models: Support for patterns

- SPMD:
  - MPI does this well. (TBB/pthreads/OpenMP… support the rest.)
  - Think of all that mpiexec does.
- Parallel_for, Parallel_reduce:
  - Should be automatic from vanilla source (OpenACC a start).
  - Make CUDA obsolete. OpenMP sufficient?
- Task graphs, pipelines
  - Lightweight.
  - Smart about data placement/movement, dependencies.
- Thread team:
  - Needed for fine-grain producer/consumer algorithms.
- Others too.

Goals:

1) **Allow domain scientist think parallel, write sequential.**
2) **Support rational migration strategy.**

Sandia National Laboratories

Needs: Data management

- Break storage association:
  - Physics i,j,k should not be storage i,j,k.
- Layout as a first-class concept:
  - Construct layout, then data objects.
  - Chapel has this right.
- Better NUMA awareness/resilience:
  - Ability to "see" work/data placement.
  - Ability to migrate data: MONT
- 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.
  - Note: Dynamic work-stealing is not as easy as it seems.
- Maybe better thread local allocation will mitigate impact.

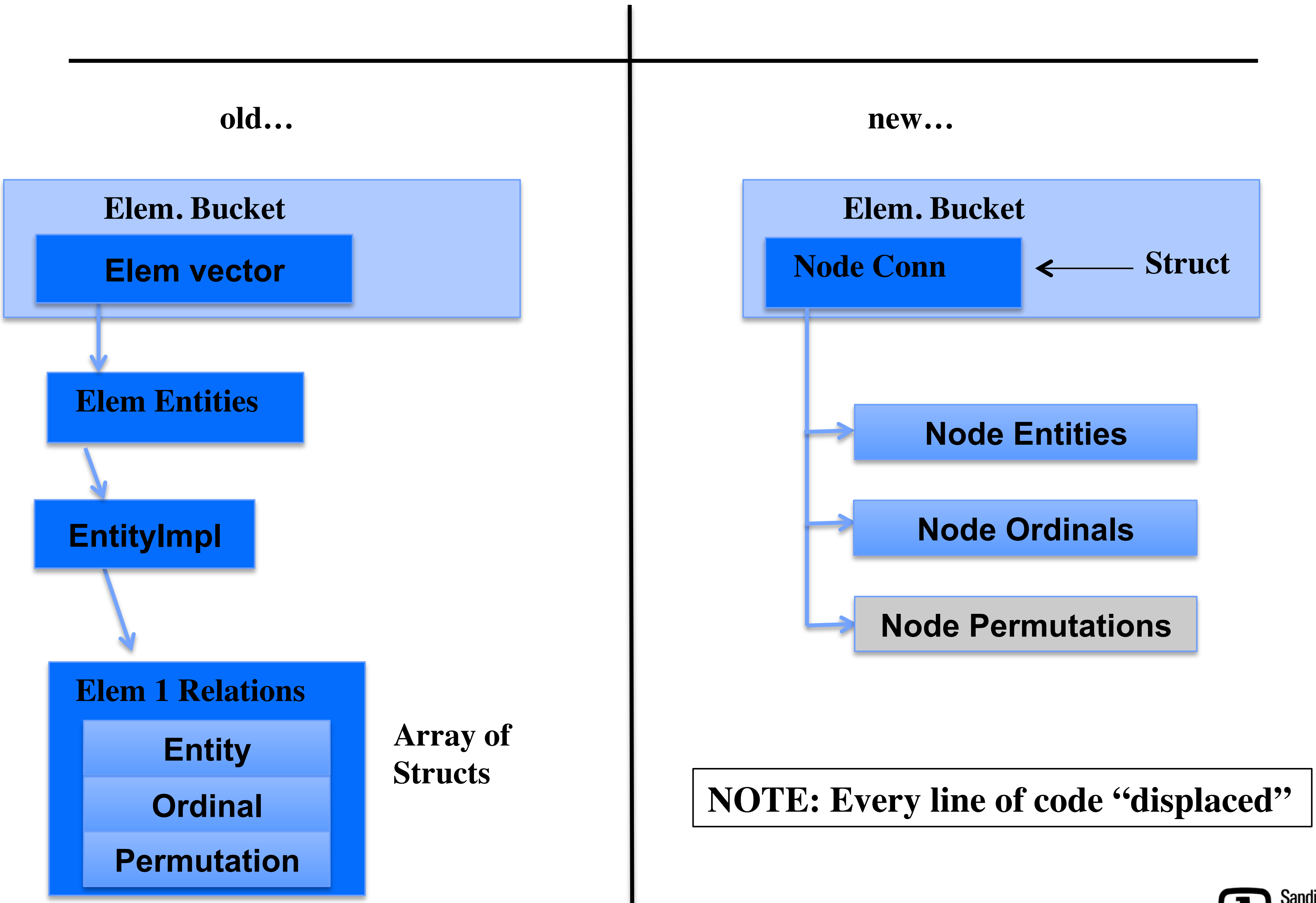
Multi-dimensional Dense Arrays

- Many computations work on data stored in multi-dimensional arrays:
  - Finite differences, volumes, elements.
  - Sparse iterative solvers.
- Dimension are (k,l,m,…) 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) … or
  - By node: A(1,1), A(1,2), …
- **Adherence to raw language arrays forces a choice.**

Array of Structs of Arrays

- Two typical Fortran choices:
  - Dim A(3,1000000) : “Array of structs”
    - Order: A(1,1), A(2,1), A(3,1); A(1,2)
    - Good nodal data locality: Good for caches, threading.
  - Dim A(1000000,3) : “Struct of arrays”
    - A(1,1), A(2,1), …, A(1000000,1), A(1,2), A(2,2), …, A(1000000,2),…
    - Long vectors: Good for GPU.
- But: CPU cores are starting to look a bit like a GPU SM.
- Third option: “Array of structs of arrays”
  - Dim A(*j*,3,*k*) where *j**k* = 1000000
  - Vectorizing CPU: *j*=10, *k*=100000
  - GPU: *j*=100000, *k*=10

Stk-mesh connectivity data layout

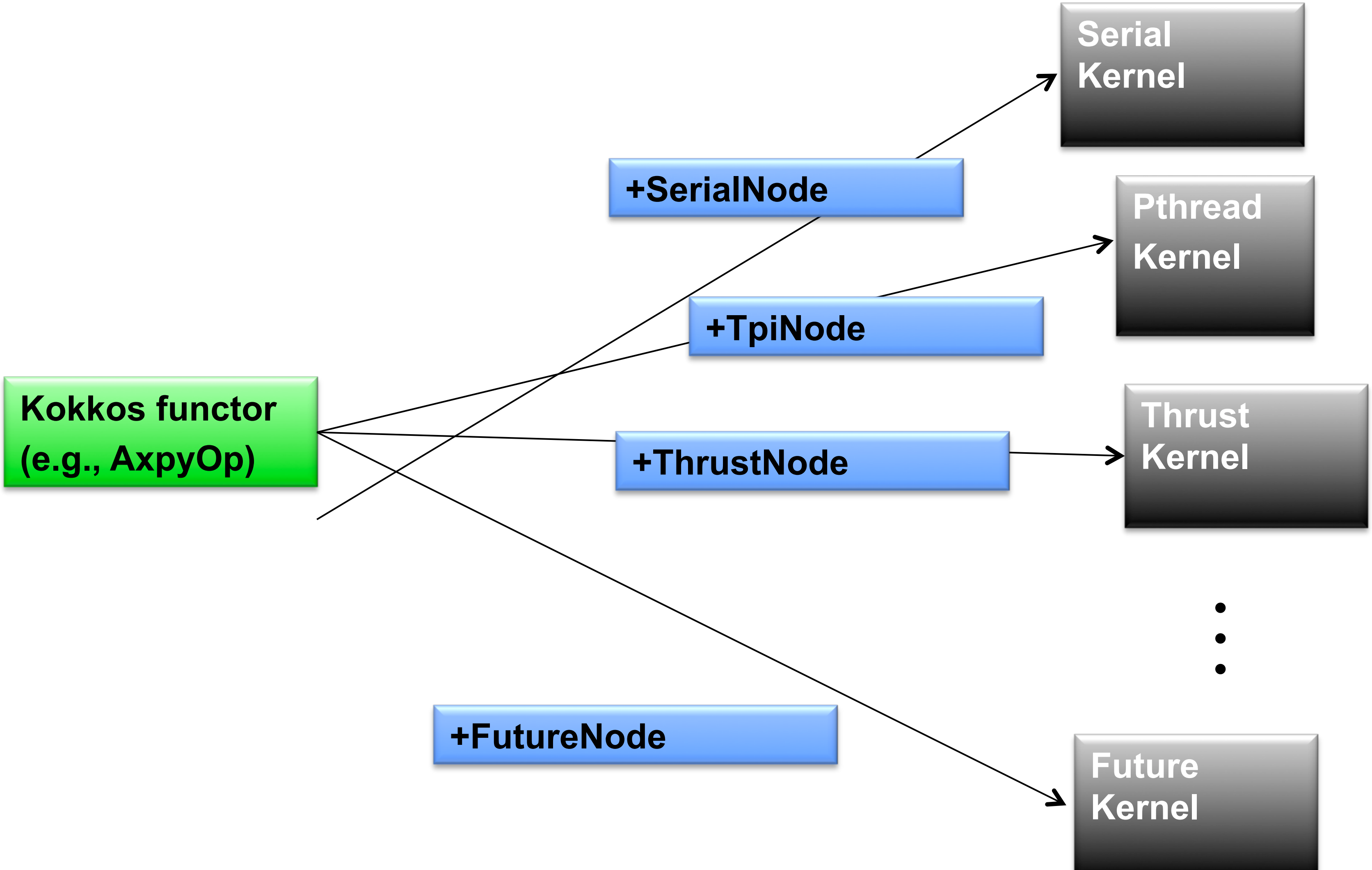


Struct-of-Arrays vs. Array-of-Structs

A False Dilemma

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

Compile-time Polymorphism



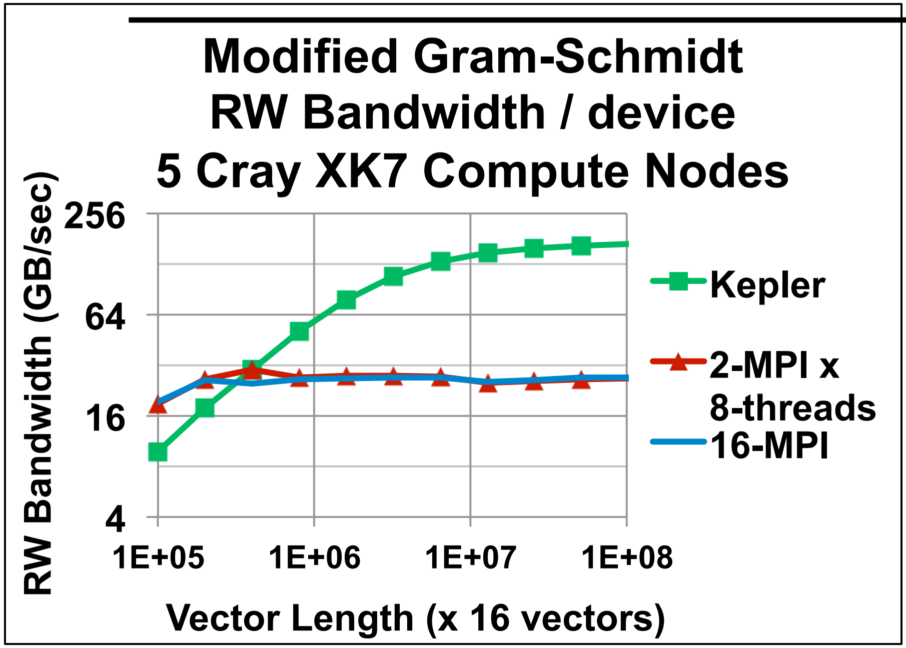
Kokkos Array Introduction

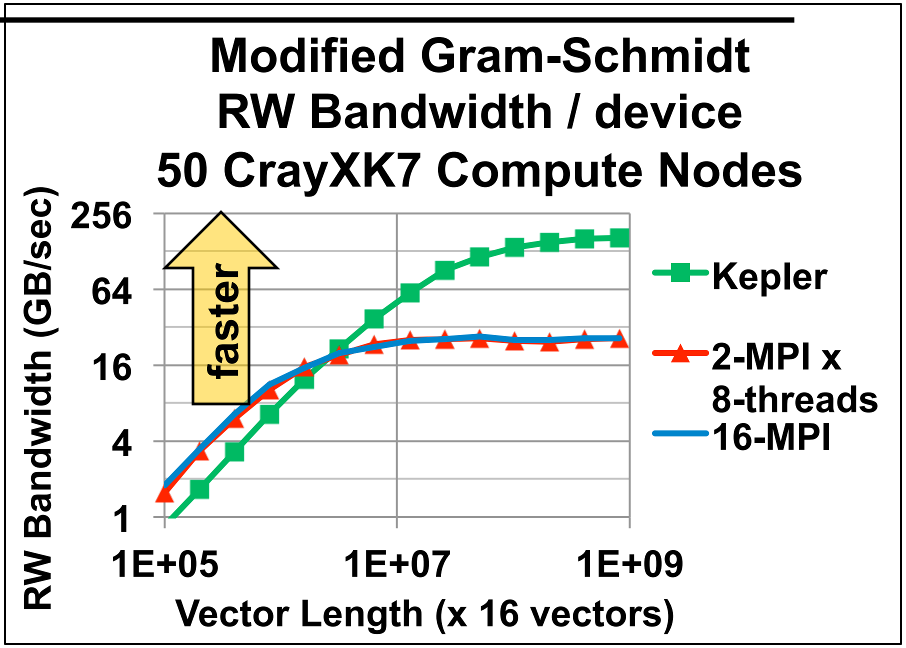
- Challenge: Manycore Portability with Performance
  - Multicore-CPU and manycore-accelerator (e.g., NVIDIA)
  - Diverse memory access patterns, shared memory utilization, …
- Via a Library, not a language
  - C++ with template meta-programming
  - In the *spirit* of Thrust or Threading Building Blocks (TBB)
  - Concise and simple API: functions and multidimensional arrays
- Data Parallel Functions
  - Deferred task parallelism, pipeline parallelism, ...
  - Simple parallel_for and parallel_reduce semantics
- Multidimensional Arrays
  - versus "arrays of structs" or "structs of arrays"

Kokkos Array Abstractions

- Manycore Device
  - Has many threads of execution sharing a memory space
  - Manages a memory space separate from the host process
    - Physically separate (GPU) or logically separate (CPU)
    - or with non-uniform memory access (NUMA)
- Data Parallel Function
  - Created in the host process, executed on the manycore device
  - Performance can be dominated by memory access pattern
    - E.g., NVIDIA coalesced memory access pattern
- Multidimensional Array
  - ➢ Map array data into a manycore device's memory
  - Partition array data for data parallel work
  - Function + parallel partition + map -> memory access pattern

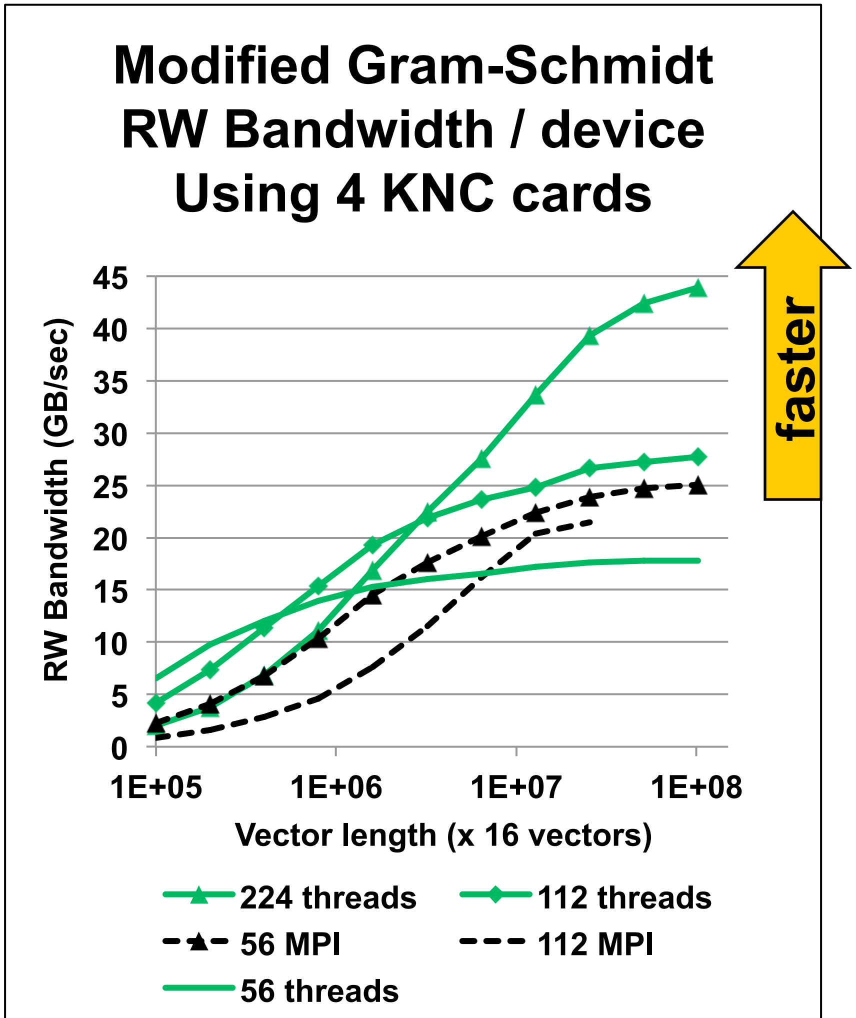
Modified Gram-Schmidt Performance
Limited by bandwidth and reductions





- **Performance normalized by # devices**
- **CrayXK7 compute nodes**
  - **AMD Opteron 6200 (2x8 cores), ~51 GB/sec theoretical peak**
  - **NVIDIA K20X, ~250 GB/sec theoretical peak**
- **RW performance at "large enough" problem size**
  - **Opteron: achieved ~51% of peak**
  - **K20X: achieved ~65% of peak**

Modified Gram-Schmidt Performance
On Knights Corner (pre-production)



- **Hyperthreading**
  - **Threads-on-hyperthreads improves performance**
  - **MPI-on-hyperthreads degrades performance**
- **RW performance at "large enough" problem size**
  - **Performance normalized by device**
  - **~200 GB/sec "achievable" peak (pre-production hardware)**
  - **Full threading utilization achieved ~23% of "achievable" peak**
  - **MPI-per-core achieved ~13% of "achievable" peak**

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

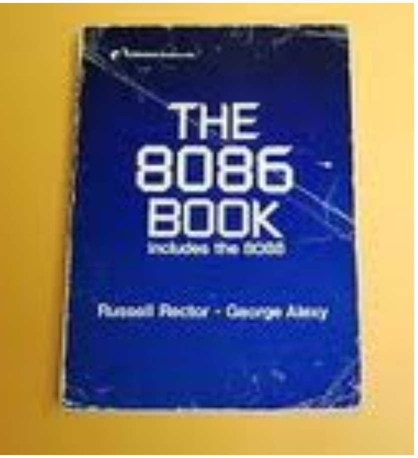
Self-healing system software and application codes needed

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

Sandia National Laboratories

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.

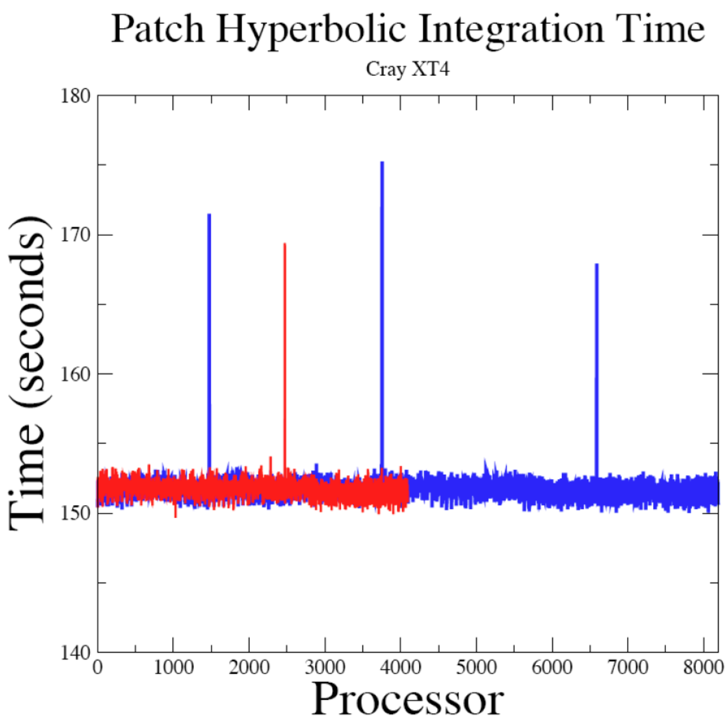
Sandia National Laboratories

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.*

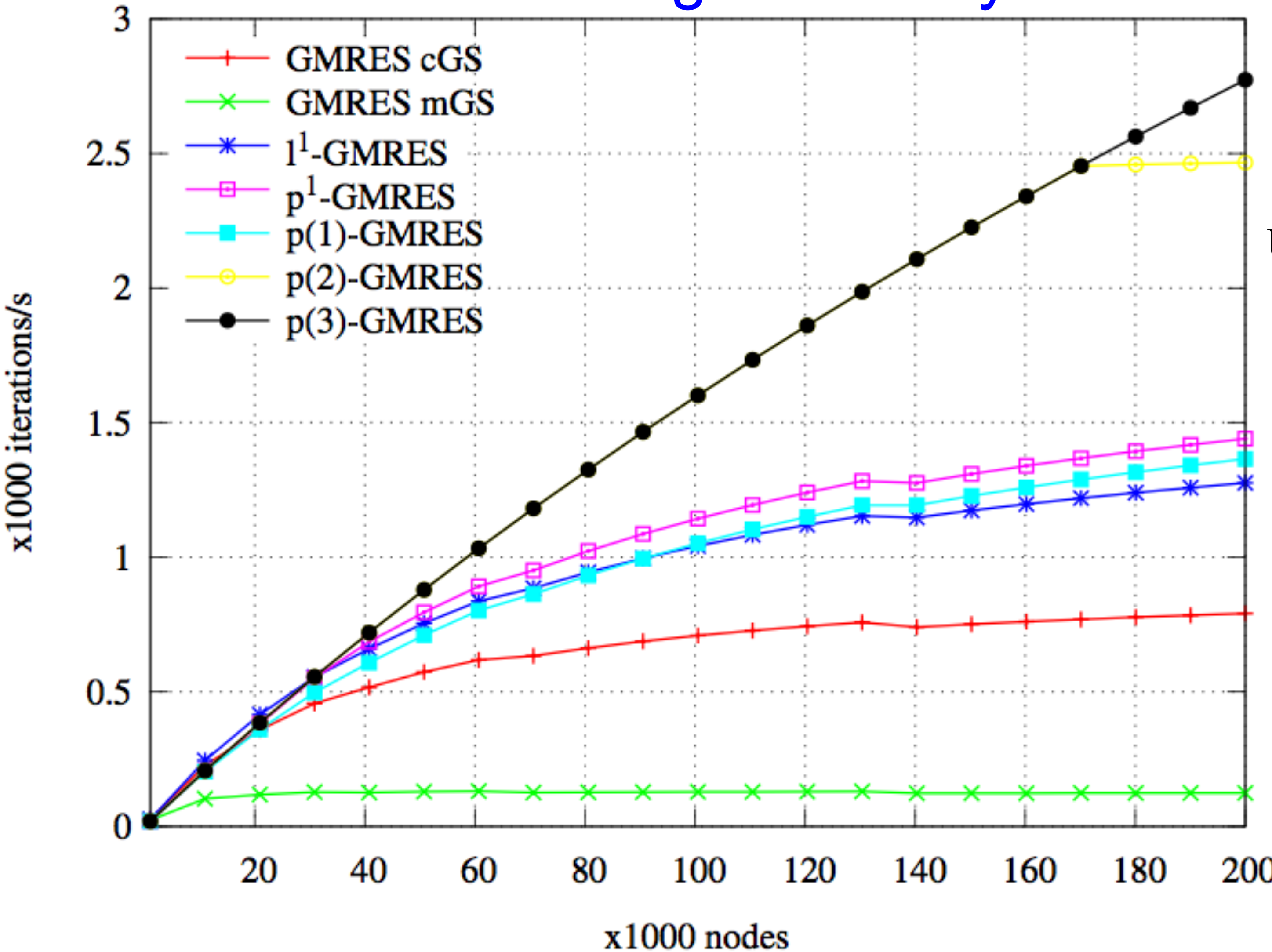
Resilience Issues Already Here

- 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:
  - $t_c = \max_i\{t_i\}$



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

Latency-tolerant Algorithms + MPI 3: Recovering scalability



Up is good

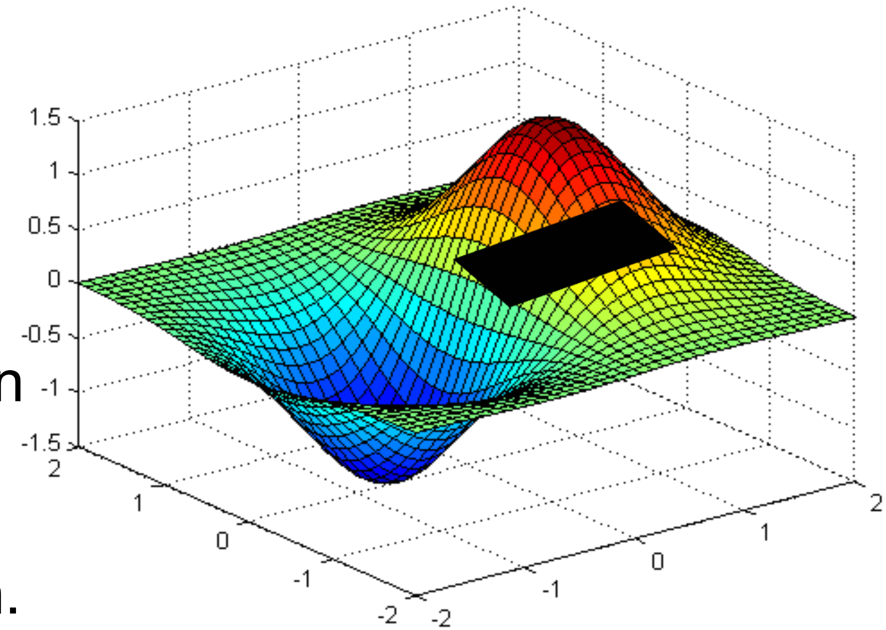
Hiding global communication latency in the GMRES algorithm on massively parallel machines, **P. Ghysels T.J. Ashby K. Meerbergen W. Vanroose, Report 04.2012.1, April 2012, ExaScience Lab Intel Labs Europe**

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.

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.



Sandia National Laboratories

Local Recovery from Local Faults Advantages

- 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

Every calculation matters

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.

Soft Error Resilience

- 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.

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 $j = 1, 2, \ldots$ until convergence **do**

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

  $v_{j+1} := Az_j$

  for $i = 1, 2, \ldots, k$ **do** ▷ Orthogonalize v_{j+1}

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

  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

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

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

end for

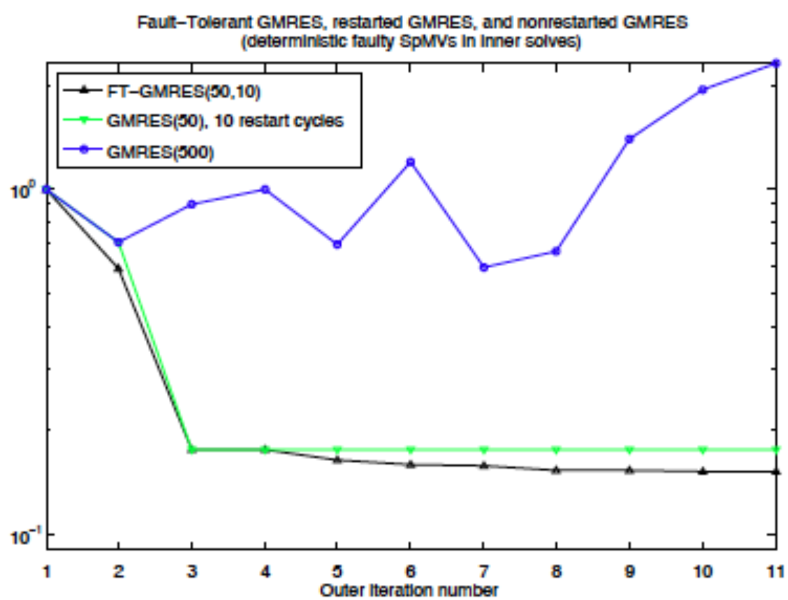
"Unreliably" computed.
Standard solver library call.
Majority of computational cost.

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

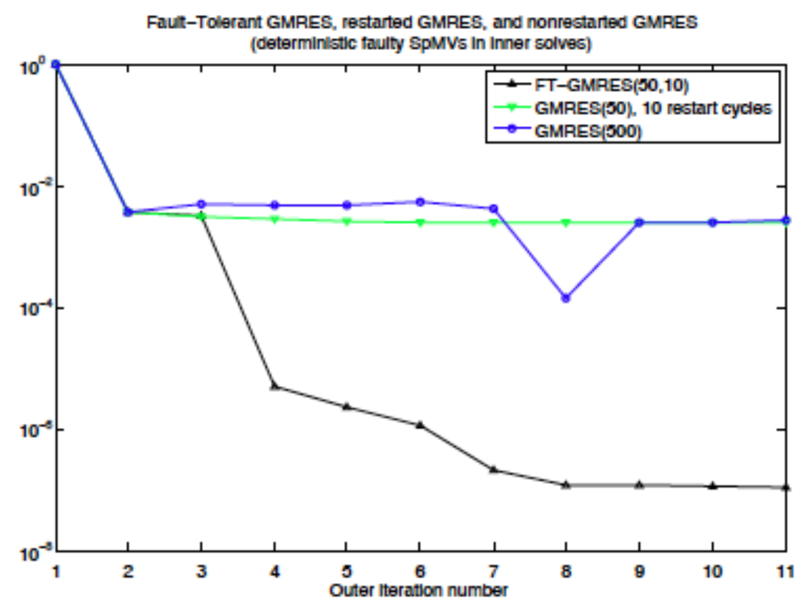
Sandia National Laboratories

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).

Sandia National Laboratories

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

Sandia National Laboratories

What is Needed for Selective Reliability?

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

- Containment domains a good start.

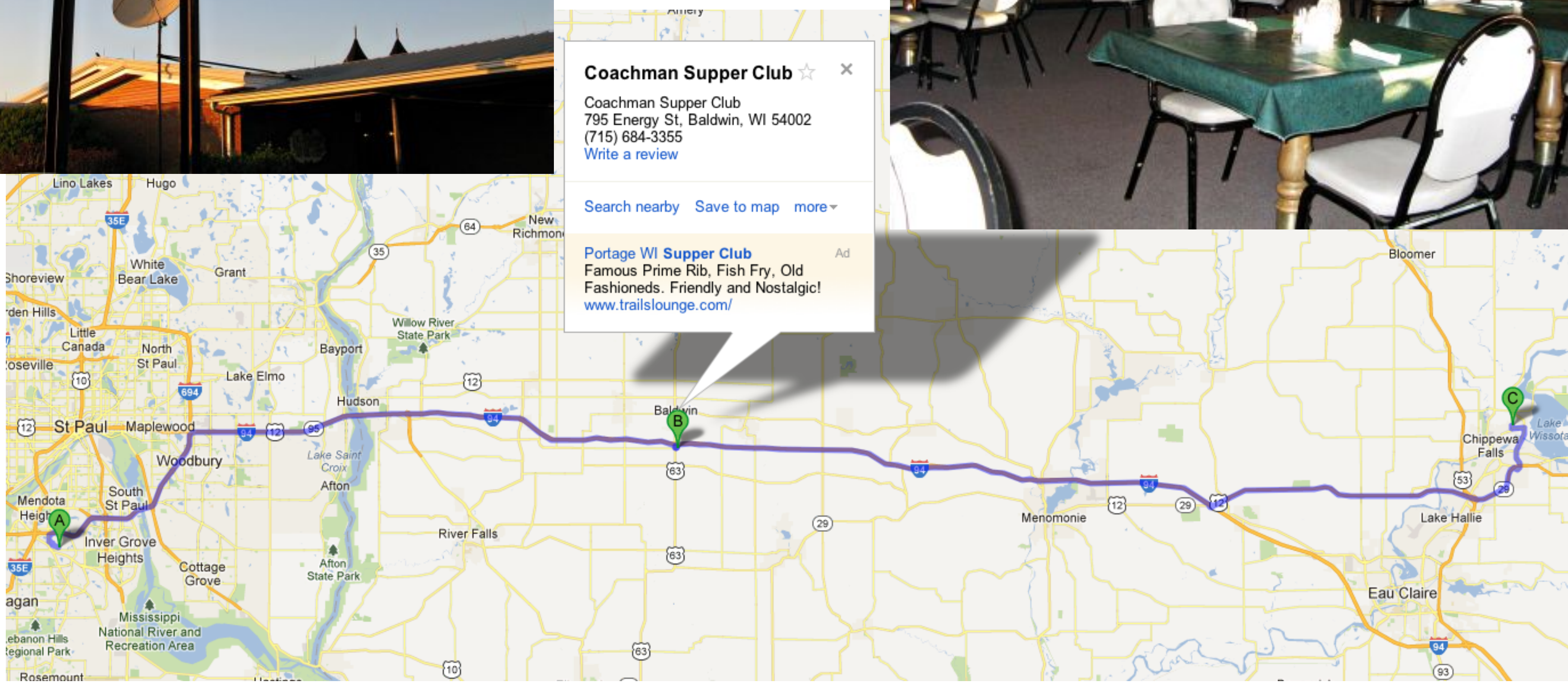
Strawman Resilient Exascale System

- Best possible global CP/R:
  - Maybe, maybe not.
  - Multicore permitted simpler cores.
  - Resilient apps may not need more reliable CP/R.
    - "Thanks, but we've outgrown you."
- Async collectives:
  - Workable today.
  - Make robust. Educate developers.
  - Expect big improvements when apps adapt to relaxed BSP.
- Support for LFLR:
  - Next milestone.
  - FT in MPI: Didn't make into 3.0…
- Selective reliability.
- Containment domains.
- Lots of other clever work: e.g., flux-limiter, UQ, …

Comments on Code Migration Strategy

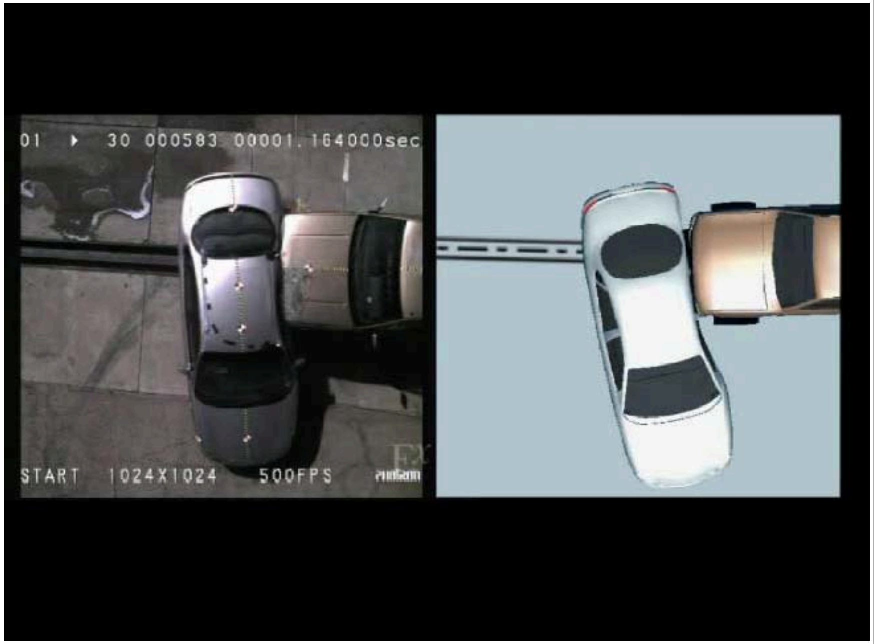
Co-Design Cray-style (circa 1996)

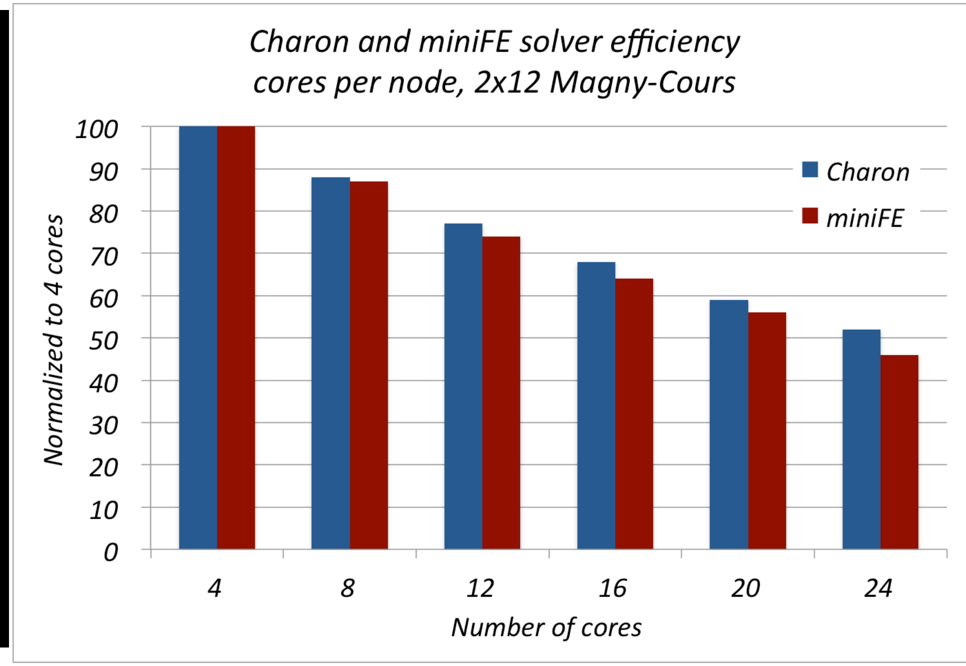
(This is not a brand new idea)



Sandia National Laboratories

Proxy Apps & Performance Modeling





Source: http://www.exponent.com/human_motion_modeling_simulation

- Scientific apps are a *model* of real physics.
- Proxy apps are a *model* of real application performance.
- Analogy is strong:
  - Model are simplified, known strengths, weaknesses.
  - Validation is important.

Sandia National Laboratories

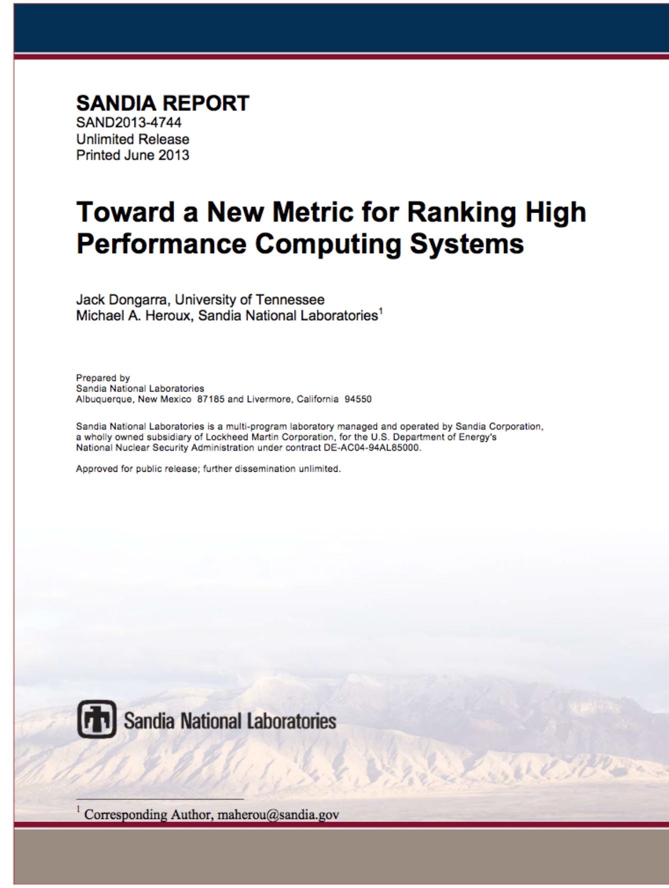
Proxy Apps are *not* Benchmarks

- Benchmarks:
  - Static.
  - Rigid specification of algorithm.
  - Reduces design space choices.
- Proxy apps:
  - About *design space exploration.*
  - Meant to be re-written.
  - Meant to enable “co-design”.
- Note: Actively working to avoid benchmarkification.
- Mantevo: Started 6+ years ago.
- Miniapps: Central to 3 ASCR Co-design Centers.

HPCG Benchmark

Toward a New Metric for Ranking
High Performance Computing Systems

– Michael Heroux and Jack Dongarra


SANDIA REPORT
SAND2013-4744
Unlimited Release
Printed June 2013

Toward a New Metric for Ranking High Performance Computing Systems

Jack Dongarra, University of Tennessee
Michael A. Heroux, Sandia National Laboratories[1]

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

[1] Corresponding Author, maherou@sandia.gov

http://www.sandia.gov/~maherou/docs/HPCG-Benchmark.pdf

Software Engineering and HPC
Efficiency vs. Other Quality Metrics

Verification (→ Correctness)

Validation (→ Accuracy)

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

- Node-level parallelism is new commodity curve (today):
  - Threads (laptop: 8 on 4 cores), vectors (Intel SB/KNC: Gath/Scat).
- Domain experts need to “think” in parallel.
  - Building a parallel pattern framework is an effective approach.
- Most future programmers won’t need to write parallel code.
  - Pattern-based framework separates concerns (parallel expert).
  - Domain expert writes sequential fragment. (Even if you are both).
- Fortran/C can be used for future parallel applications, but:
  - Complex parallel patterns are very challenging (impossible).
  - Parallel features lag, lack of compile-time polymorphism hurts.
  - Storage association is a big problem.
- Resilience is a major front in extreme-scale computing.
  - Resilience with current algorithms base is not feasible.
  - Need algorithms-driven resilience efforts.

Sandia National Laboratories