Next generation programming environments: What we need and do not need

Michael A. Heroux
Scalable Algorithms Department
Sandia National Laboratories

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

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

Sandia National Laboratories

Observations and Strategies for Next Generation Parallel Applications

Sandia National Laboratories

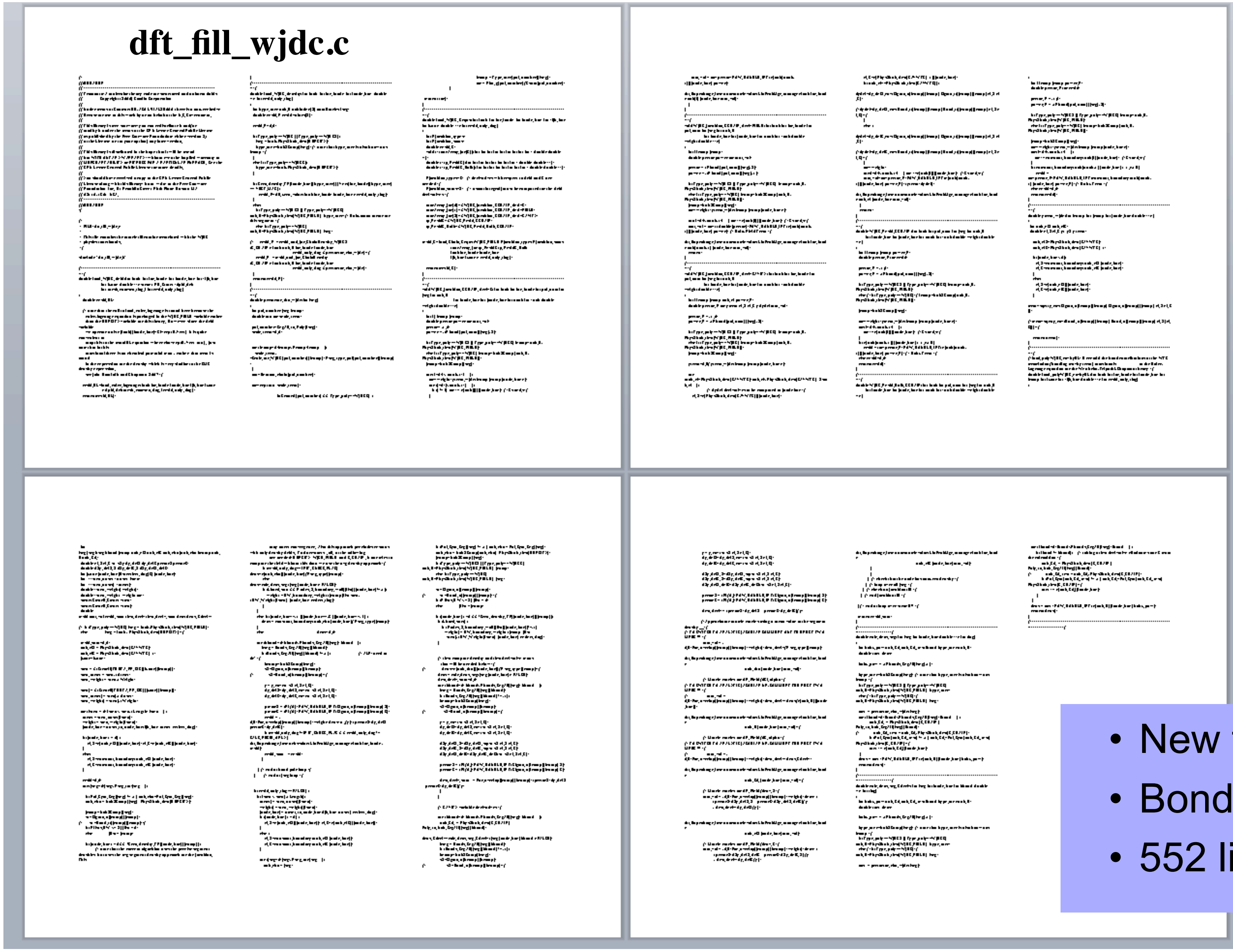
Three Parallel Computing Design Points

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

Common Element

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

Tramonto WJDC Functional



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

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

How much MPI-specific code?

dft_fill_wjdc.c
MPI-specific
code

SPMD Patterns for Domain Decomposition

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

Reasons for MPI/SPMD Success?

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

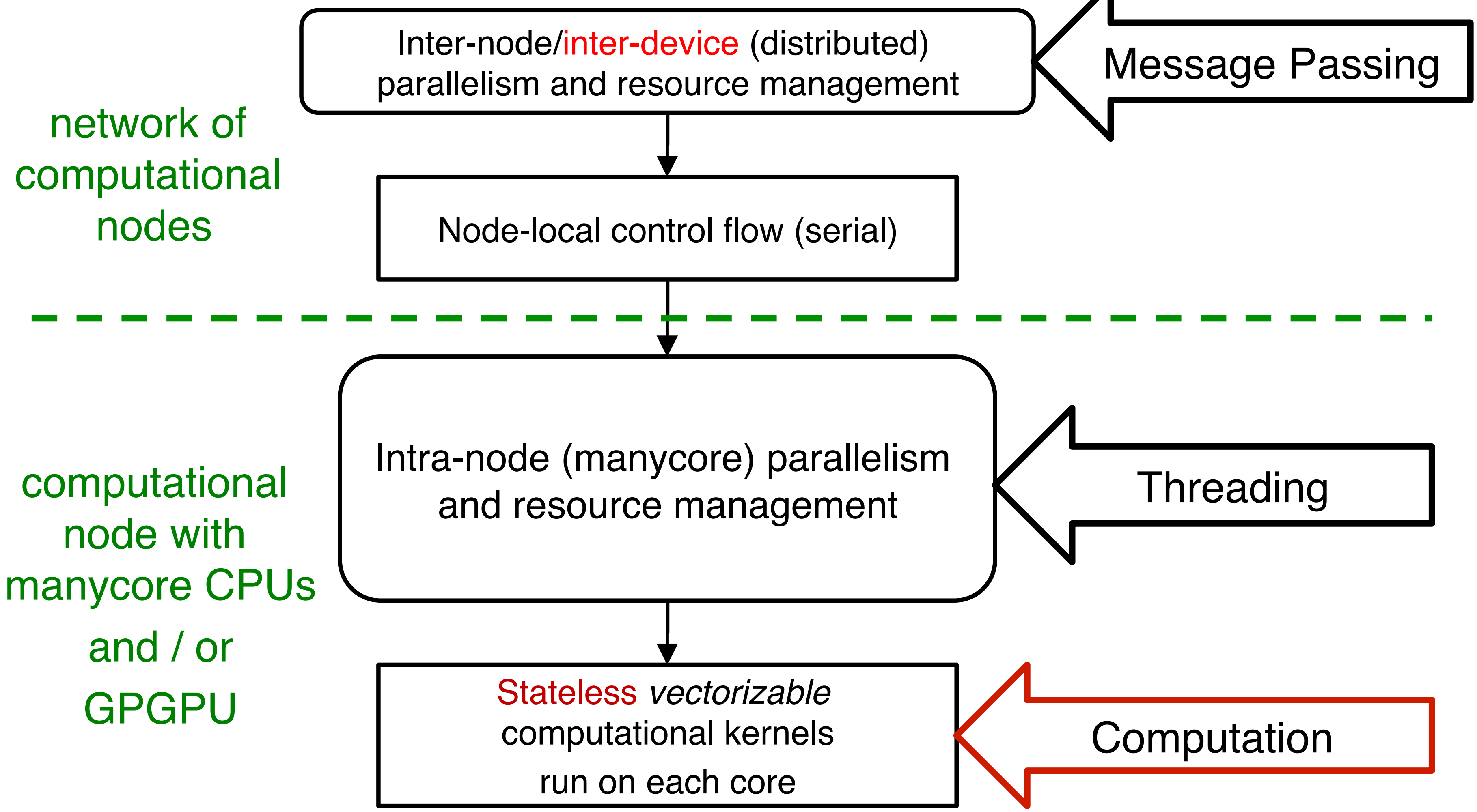
- Once framework in place:
  - Sophisticated physics added as serial code.
  - Ratio of science experts vs. parallel experts: 10:1.

- **Key goal for new parallel apps: Preserve this ratio**

Sandia National Laboratories

Evolving Parallel Programming Model

Parallel Programming Model: Multi-level/Multi-device



Domain Scientist's Parallel Palette

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

- Next-generation PDE and related applications:
  - Internode:
    - MPI, yes, or something like it.
    - Composed with intranode.
  - Intranode:
    - Much richer palette.
    - More care required from programmer.

- What are the constructs in our new palette?

Obvious Constructs/Concerns

- Parallel for:

  forall (i, j) in domain {…}

  – No loop-carried dependence.

  – Rich loops.

  – Use of local 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.

Sandia National Laboratories

Other construct: Thread team

- Multiple threads.
- Fast barrier.
- Shared, fast access memory pool.
- Example: Nvidia SM
- Supports fine-grain producer-consumer parallelism.
- X86 more vague, emerging more clearly in future.

Finite Elements/Volumes/Differences and parallel node constructs

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

Resilient Algorithms:
A little reliability, please.

Every calculation matters

Soft Error Resilience

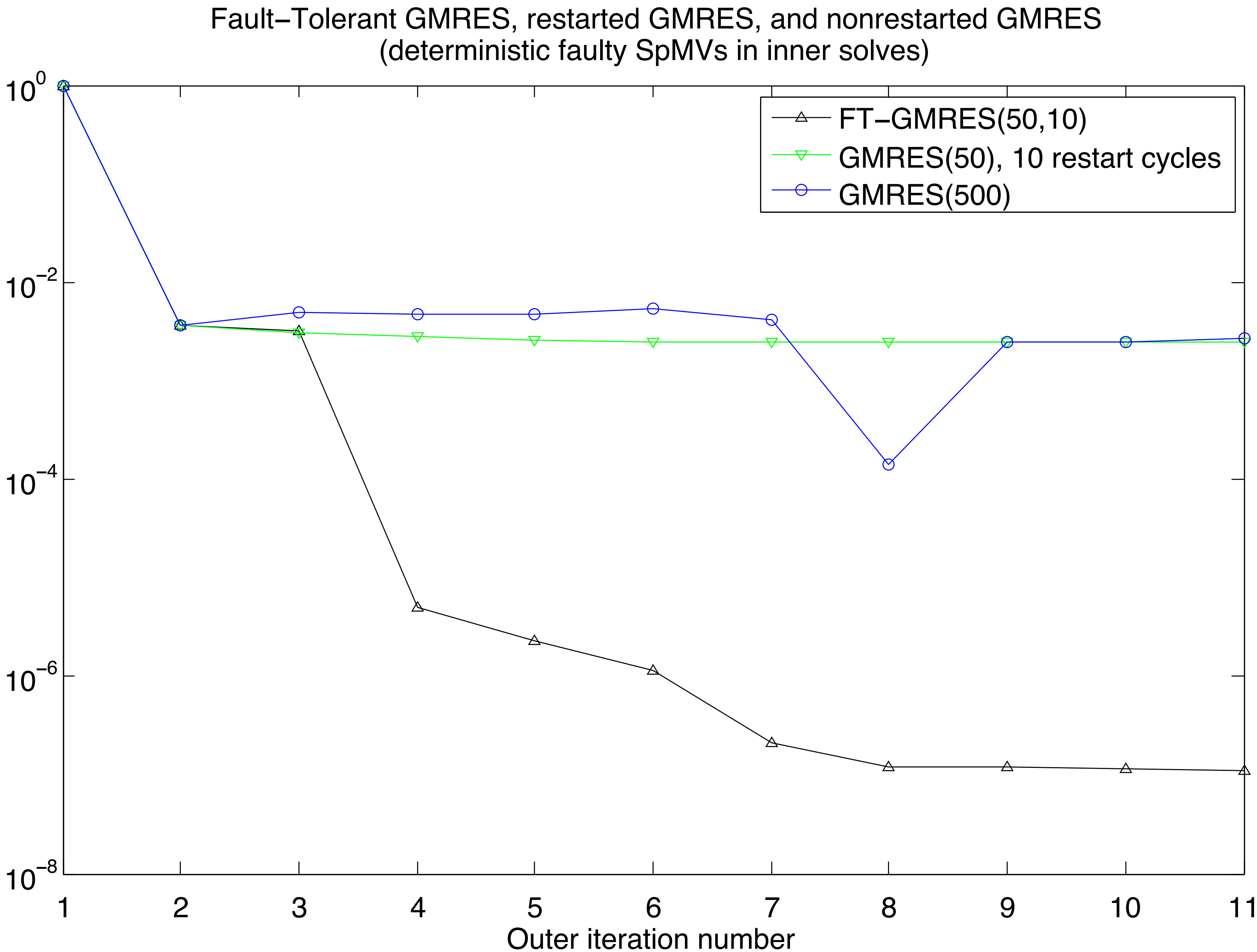
Description	Iters	FLOPS	Recursive Residual Error	Solution Error
All Correct Calcs	35	343M	4.6e-15	1.0e-6
Iter=2, y[1] += 1.0 SpMV incorrect Ortho subspace	35	343M	6.7e-15	3.7e+3
Q[1][1] += 1.0 Non-ortho subspace	N/C	N/A	7.7e-02	5.9e+5

- Small PDE Problem: ILUT/GMRES
- Correct result:35 Iters, 343M FLOPS
- 2 examples of a single bad op.
- Solvers:
  - 50-90% of total app operations.
  - Soft errors most likely in solver.
- Need new algorithms for soft errors:
  - Well-conditioned wrt errors.
  - Decay proportional to number of errors.
  - Minimal impact when no errors.

- New Programming Model Elements:
  - SW-enabled, highly reliable:
    - Data storage, paths.
    - Compute regions.
- Idea: *New algorithms with minimal usage of high reliability.*
- First new algorithm: FT-GMRES.
  - Resilient to soft errors.
  - Outer solve: Highly Reliable
  - Inner solve: “bulk” reliability.
- General approach applies to many algorithms.

M. Heroux, M. Hoemmen

FTGMRES Results



What we need and don't need

What we need from Programming Models: Support for patterns

- SPMD:
  - MPI does this well. (TBB supports the rest.)
  - Think of all that mpiexec does.
- Task graphs, pipelines
  - Lightweight.
  - Smart about data placement/movement, dependencies.
- Parallel_for, Parallel_reduce:
  - Should be automatic from vanilla source.
  - Make CUDA obsolete. OpenMP sufficient?
- 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

- 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 problem.

Other needs

- Metaprogramming support:
  - Compile-time polymorphism
  - Fortran, C are not suitable.
  - C++ is, but painful.
  - Are new languages?
- Reliability expression:
  - Bulk vs. high reliability.
- Composable with other environments.
  - Interoperable with MPI, threading runtimes.

A Different Approach

I don’t want to be considered a Luddite…

- Massively threaded approaches have promise.
- Makes coding much simpler, at least on a node.
- Key question:

  Is there enough demand to produce high quality system?

Sandia National Laboratories

What I cannot use

- Isolated tools:
  - “Great ideas with marginal chance of being products.”
  - Fortran 2003 features: Still not available!
  - CAF, UPC: Too little, too late.
  - Rose: Where is ‘sudo apt-get install rose’?
- Any programming environment effort:
  - Must have product plan, from desktop up, e.g., OpenMP.
  - Or must extend an existing product, e.g., TBB.
- We use commodity chips because only a few orgs have the billions of dollars to design and fab.
- We use commodity programming environments for the same reason.

Summary

- Building the next generation of parallel applications requires enabling domain scientists:
  - To write sophisticated computational expressions.
  - Do so with serial fragments.
  - Where fragments hoisted into scalable, resilient fragment.
- A pattern-based approach offers:
  - Parallel thinking, sequential programming.
  - A migration strategy similar to SPMD migration of early 90's.
- Massively threaded programming is attractive:
  - Is there a sufficient market to drive it?
- Progress in programming environment requires:
  - Addressing technical requirements, yes, but
  - Product planning has to be just as important.

Extra Slides

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

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

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

Emerging Architecture Programming Challenges

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

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

Factoring 1K to1B-Way Parallelism

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

Sandia National Laboratories

Data Movement: Locality

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

Memory Size

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

Sandia National Laboratories

Resilience

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

Summary of Algorithms Challenge

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

New Trends and Responses

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

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.

Placement and Migration

Placement and Migration

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

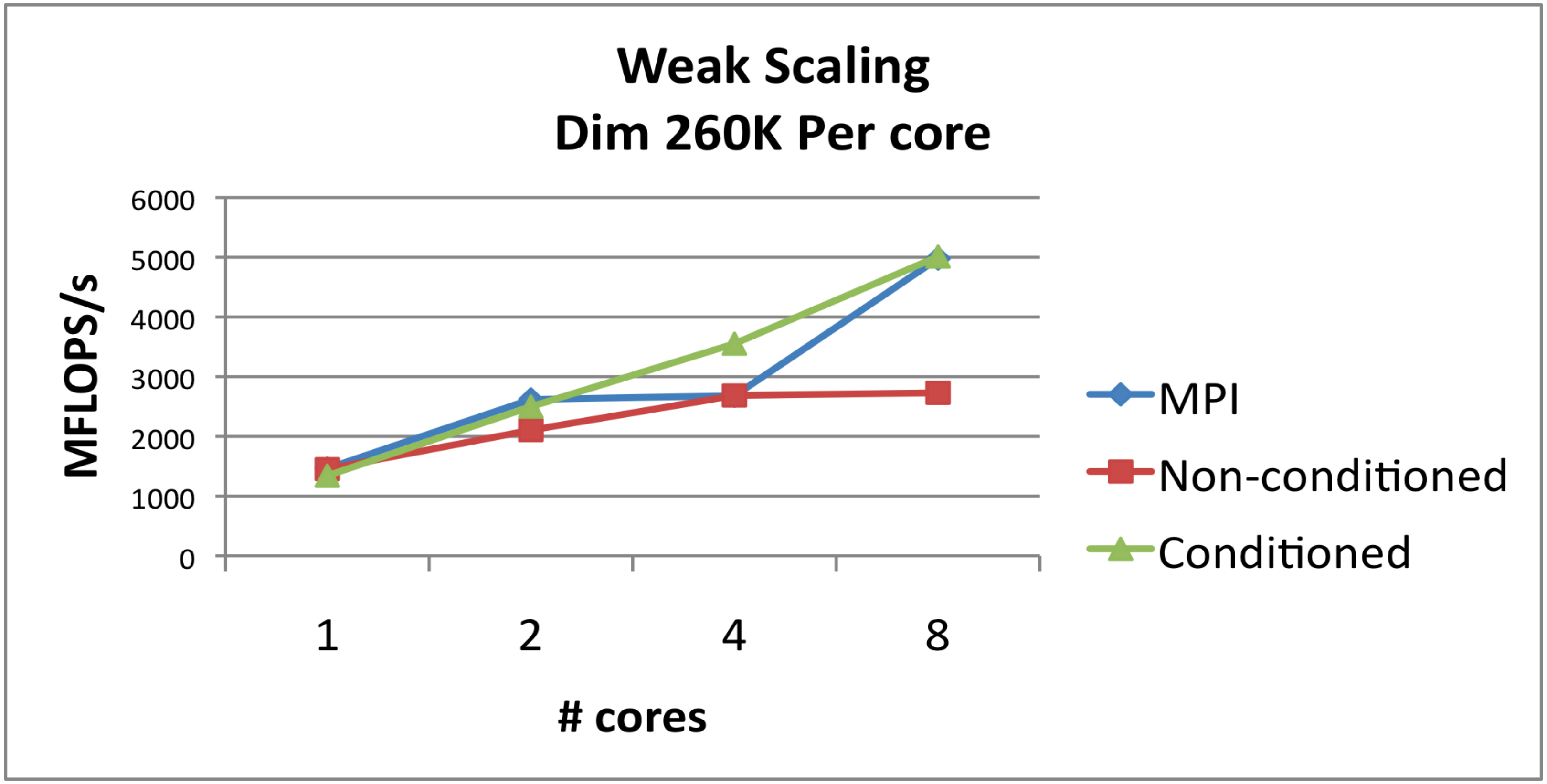
Data Placement on NUMA

- Memory Intensive computations: Page placement has huge impact.
- Most systems: First touch (except LWKs).
- Application data objects:
  - Phase 1: Construction phase, e.g., finite element assembly.
  - Phase 2: Use phase, e.g., linear solve.
- Problem: First touch difficult to control in phase 1.
- Idea: Page migration.
  - Not new: SGI Origin. Many old papers on topic.

Data placement experiments

- MiniApp: HPCCG (Mantevo Project)
- Construct sparse linear system, solve with CG.
- Two modes:
  - Data placed by assembly, not migrated for NUMA
  - Data migrated using parallel access pattern of CG.
- Results on dual socket quad-core Nehalem system.

Weak Scaling Problem



- MPI and conditioned data approach comparable.
- Non-conditioned very poor scaling.

Page Placement summary

- MPI+OpenMP (or any threading approach) is best overall.
- But:
  - Data placement is big issue.
  - Hard to control.
  - Insufficient runtime support.
- Current work:
  - Migrate on next-touch (MONT).
  - Considered in OpenMP (next version).
  - Also being studied in Kitten (Kevin Pedretti).
- Note: This phenomenon especially damaging to OpenMP common usage.