SAND2011-4821C

Architecture-aware algorithms for extreme-scale computing

Mark Hoemmen
mhoemme@sandia.gov

Sandia National Laboratories[1]

14 July 2011

[1] Sandia is a multiprogram 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.

Sandia National Laboratories

Research supported by

- Special thanks to the U.S. Department of Energy Office of Advanced Scientific Computing Research (ASCR), for funding this research through the Extreme-Scale Algorithms and Architectures Software Initiative (EASI) project.
- My research as a UC Berkeley student was funded by a grant by Microsoft and Intel (Award #20080469), with matching funding by a University of California Discovery grant (Award #DIG07-10227).

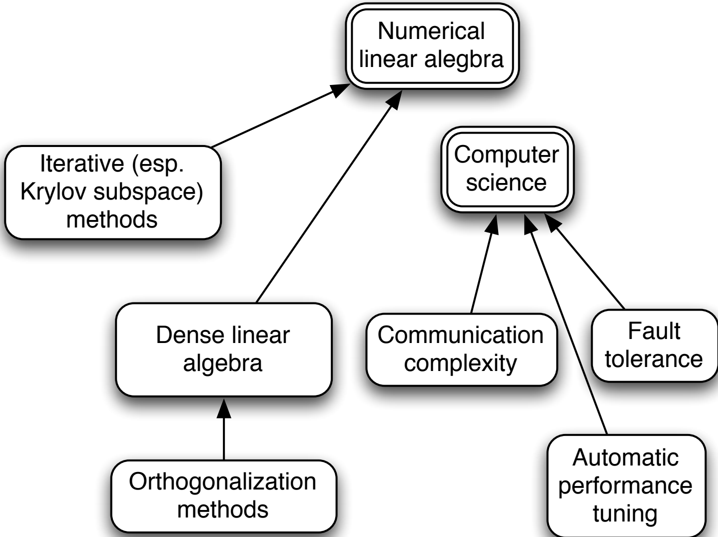
Sandia National Laboratories

Biography

- UIUC 2002: B.S. Math and Computer Science
- Fulbright at Tech. U. Berlin 2002-3: Math
- UC Berkeley spring 2010: PhD Computer Science
  - Thesis: "Communication-avoiding Krylov subspace methods"
- SNL/NM postdoc: spring 2010 - present

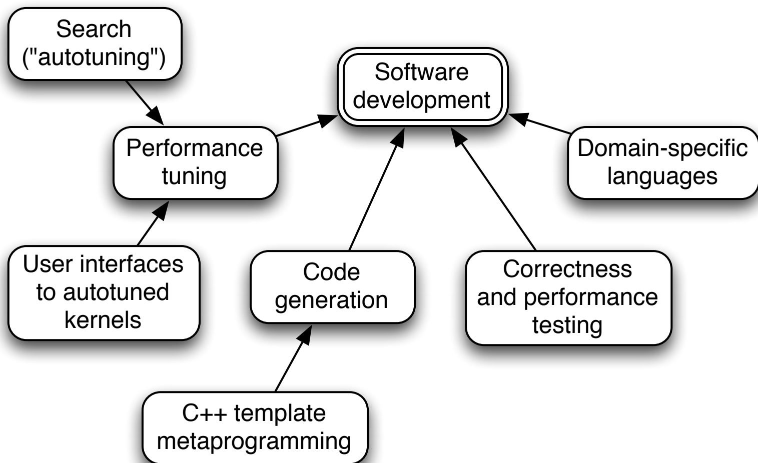
Sandia National Laboratories

Research interests



Sandia National Laboratories

Software development interests



Sandia National Laboratories

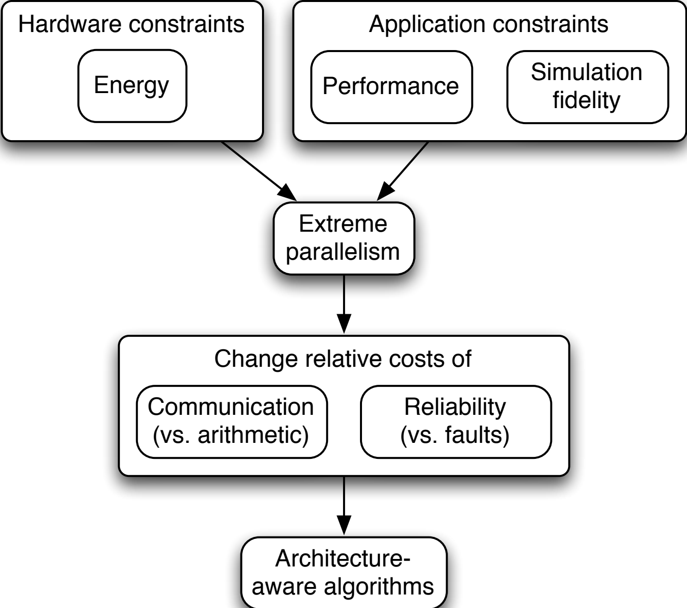
Talk outline via job title

"Algorithms R&D for Extreme-Scale Computing"

- Extreme-scale computing
  - Demands *architecture-aware* algorithms
  - Architecture awareness includes
    - Avoiding communication
    - Tolerance of soft faults
- Research and development
  - "Development" means *software*
    - Contributions to Trilinos

Sandia National Laboratories

Extreme-scale computing



Sandia National Laboratories

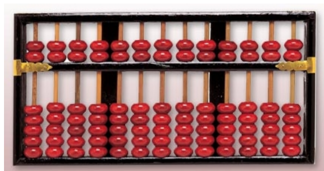
Architecture-aware algorithms

- Hardware evolves $\Rightarrow$ relative costs of operations change
  - "Operations":
    - Arithmetic, data movement, communication
    - Quality-of-service guarantees (e.g., reliability)
  - "Cost": speed, energy, ...
- Changing costs $\Rightarrow$ may need new algorithms
  - If changing hardware too costly / impossible
- "Algorithms" vs. code optimizations
  - Ideal: Asymptotic vs. $O(1)$ improvement
  - Potential accuracy / cost trade-offs
  - Require numerical linear algebra expertise

Two extreme-scale issues drive two approaches

- Communication expensive relative to arithmetic
  - $\Rightarrow$ Communication-avoiding iterative methods
- Soft faults (e.g., bit flips) more likely
  - $\Rightarrow$ Fault-tolerant iterative methods

Computers do two things

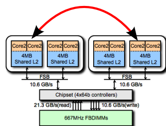


Arithmetic

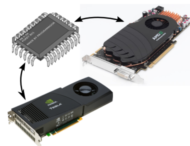
Communication

Communication is data movement

- Parallel: between processors

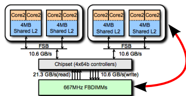


Between cores on a node, or nodes of a cluster

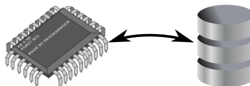


Between core(s) and coprocessor(s)

- Sequential: between levels of memory hierarchy



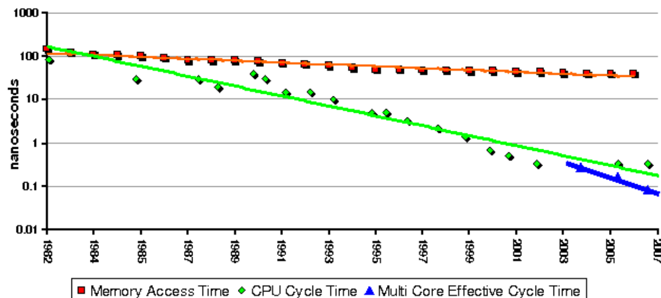
Cache to DRAM



DRAM to disk

Communication slow relative to arithmetic

- Floating-point rate $\gg$ 1/bandwidth $\gg$ latency
- Relative to arithmetic, communication gets *exponentially slower* over time
- Example: memory latency & floating-point throughput
  - Comparable in 1982
  - Memory latency $\approx 1000\times$ slower in 2007
- Symptoms: *memory wall* and *power wall*

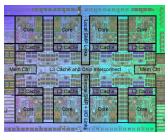


Memory latency (red) relative to arithmetic throughput (green & blue) over time (Figure 6.12, Exascale Report)

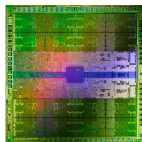
Sandia National Laboratories

Symptom 1: Memory wall

- Memory latency & bandwidth can't keep up with flop rate
- Ratio flop/s vs. how fast memory or bus can feed it:



IBM POWER7: ≈ 20



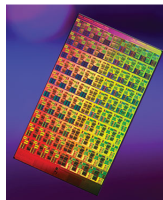
NVIDIA Fermi: ≈ 200

- True throughout memory hierarchy and between processors

- 1 MPI msg. $\approx 10^4$ flops
- 1 disk read $\approx 10^7$ flops

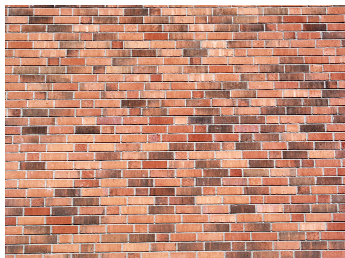
Symptom 2: Power wall

- Your cell phone will be 1000-way parallel
- Your cluster will be 10^9 - way parallel



- Can't make sequential processors faster (or they'll melt)
- Thus everything is parallel ("multicore") now
- Memory capacity isn't scaling with # cores
  - No on-node weak scaling
- More parallelism means more communication

Memory wall + power wall = brick wall



- Waiting for the next chip won't make your slow code faster
- Solution: improve algorithms – communicate less
- *Communication-avoiding algorithms*

Sandia National Laboratories

Krylov subspace methods (KSMs)

- Numerical methods for solving $Ax = b$ and $Ax = \lambda x$, e.g.,
  - For $Ax = b$: GMRES, CG, MINRES, BiCGSTAB, ...
  - For $Ax = \lambda x$: Arnoldi, $\{$symmetric, nonsymmetric$\}$ Lanczos
- Project A onto *Krylov subspace*
  - e.g., $\text{span}\{v, Av, A^2 v, \dots, A^k v\}$
- KSMs are *iterative*:
  - Grow dimension of subspace with each iteration
  - Solution is approximate and (hopefully) converges
- May involve *preconditioning*:
  - When solving $Ax = b$, may improve convergence rate
  - Solve different system w/ same answer, like $M^{-1}Ax = M^{-1}b$

Sandia National Laboratories

Who is Krylov?

- Alexei Nikolaevich Krylov (1863–1945)
- Russian naval engineer & applied mathematician
- Invented Krylov methods to solve symmetric eigenproblems
- Concerned with costs of computation
  - *Counted flops* (vs. symmetric Jacobi)
  - We also count, but what's expensive has changed...

A. N. Krylov in 1910s

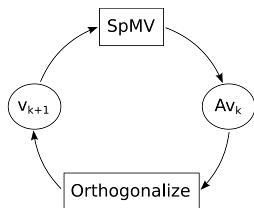
Standard KSMs are communication-bound

- Most KSMs alternate between 2 or 3 kernels:
  1. Sparse matrix-vector multiplication (SpMV)
  2. (Possibly also preconditioning)
  3. Vector-vector operations (AXPY, DOT)
- All these kernels are *communication-bound*
  1. SpMV dominated by
     - Reading the sparse matrix (bandwidth)
     - Passing messages and synchronizing (latency)
  2. Vector-vector operations dominated by
     - Reading the vectors (bandwidth)
     - Parallel reductions (latency)
- s KSM iterations:
  - Read the sparse matrix A $\Theta(s)$ times (bandwidth)
  - For vectors, $\Omega(s \log P)$ messages on P processors (latency)
- Optimal communication for s iterations:
  - Read the sparse matrix A *once*
  - For vectors, $\Theta(\log P)$ messages

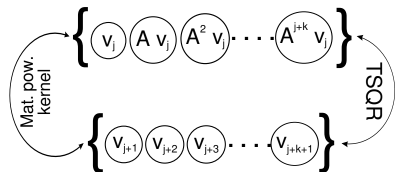
Sandia National Laboratories

Communication-avoiding approach

- Data dependency between kernels in standard KSM
  - Can't hide communication
  - Can't use faster kernels
- Break data dependency ⇒ can use faster kernels



Data dependency in standard KSMs



Data dependency in comm.-avoiding KSMs

Resulting improvements

- s iterations of standard KSMs
  - Parallel: $\Omega(s \log P)$ messages
  - Sequential: read A s times
- Our communication-avoiding KSMs
  - If the sparsity structure of A partitions with a low surface to volume ratio:
    - e.g., low-dimensional PDE discretizations
  - Parallel: $\Theta(\log P)$ messages
  - Sequential: read A only once
- Illustrate with GMRES…

Sandia National Laboratories

Original GMRES

1: v_1 is scaled initial residual
2: **for** $k = 1$ to s **do**
3: $w_k = A \cdot v_k$ ▷ Sparse matrix-vector multiplication
4: **for** $j = 1$ to k **do** ▷ Modified Gram-Schmidt
5: $H_{jk} := \langle v_j, w_k \rangle$
6: $w_k := w_k - v_j H_{jk}$
7: **end for**
8: $H_{k+1,k} := \sqrt{\langle w_k, w_k \rangle}$
9: $v_{k+1} := w_k / H_{k+1,k}$
10: **end for**
11: Compute solution using H

Sandia National Laboratories

New kernels

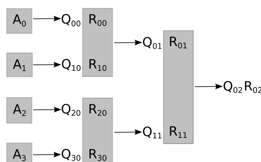
1. **Matrix powers kernel**: Given A and v, compute Av, $A^2 v$, $\ldots$, $A^s v$, for about the same communication cost as $A \cdot v$
   - Sequential: read the matrix $1 + o(1)$ time(s)
   - Parallel: $\approx$ same number of messages as $A \cdot v$
2. **Tall Skinny QR** (TSQR): $[v_1, \ldots, v_{s+1}] = Q \cdot R$
   - As accurate as Householder QR
   - Sequential: read and write vectors $\Theta(1)$ times
   - Parallel: one reduction, instead of s or s^2
3. **Block Gram-Schmidt** (BGS): like Gram-Schmidt orthogonalization, but on "blocks" of contiguous columns
   - Similar advantages to TSQR, when s is # columns / block
   - Combine with TSQR for multiple groups of $s(+1)$ vectors
   - Can do reorthogonalization efficiently & accurately

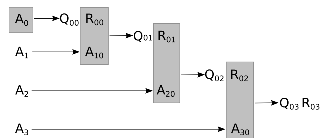
Build new iterative methods from these.

Sandia National Laboratories

Tall Skinny QR factorization

- Existing orthogonalizations communicate too much
  - $\Theta(s)$ or even $\Theta(s^2)$ reductions
- Solution: new "Tall Skinny" QR factorization (TSQR)
  - *One* reduction, with QR as reduction operator
    - See Demmel et al. 2008 (tech report)
  - Implemented in Trilinos as orthogonalization method
    - See IPDPS 2011





Sandia National Laboratories

New version of GMRES

- Idea: Any Krylov subspace basis will do
- Let's pick one not requiring dot products

1: $W = [v_1, Av_1, A^2 v_1, \ldots, A^s v_1]$ ▷ Matrix powers kernel
2: Compute $QR = W$ ▷ TSQR factorization
3: Compute upper Hessenberg $H(1:s+1, 1:s)$ using R
4: Compute solution using $H(1:s+1, 1:s)$

Sandia National Laboratories

Basis $v, Av, A^2v, \ldots$ unstable

- $v, Av, A^2v, \ldots$ looks familiar...
- It's the power method!
  - *Converges* to principal eigenvector of A
  - Bad: "converging" $\rightarrow$ NOT independent!
- Basis condition number *exponential* in s

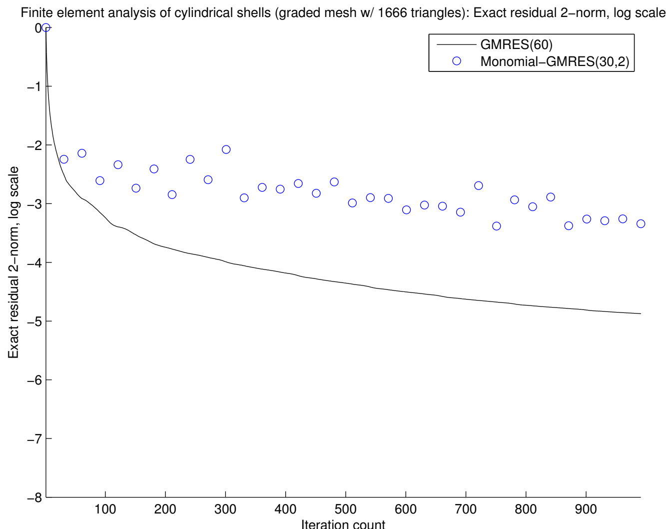
Sandia National Laboratories

Basis v, Av, $A^2 v$, ... unstable

- $v, Av, A^2 v, \ldots$ looks familiar...
- It's the power method!
  - *Converges* to principal eigenvector of A
  - Bad: "converging" $\rightarrow$ NOT independent!
- Basis condition number *exponential* in s

Sandia National Laboratories

Basis v, Av, A^2v, ... unstable



Matrix s3rmt3m3 from Matrix Market: FEM analysis of cylindrical shells, 5357×5357, $\kappa_2(A) = 2.4 \times 10^{10}$, monomial basis

Pick a different basis

- Intuition from polynomial interpolation
  - Monomial basis gets linearly dependent fast
  - a.k.a. "Vandermonde matrices are ill-conditioned"
- Use a different basis, e.g. Newton basis
  $W = [v, (A - \theta_1 I)v, (A - \theta_2 I)(A - \theta_1 I)v, \ldots]$
  - θ_k: "shifts" from Ritz values
  - "For free" from the KSM itself
  - Details later
- Same optimizations work as for v, Av, $A^2 v$, … basis

Sandia National Laboratories

Newton basis improves convergence

Finite element analysis of cylindrical shells (graded mesh w/ 1666 triangles): Exact residual 2−norm, log scale

GMRES(60)
Monomial−GMRES(30,2)
Newton−GMRES(30,2)

Exact residual 2−norm, log scale

Iteration count

Matrix `s3rmt3m3` from Matrix Market: FEM analysis of cylindrical shells, 5357×5357, $\kappa_2(A) = 2.4 \times 10^{10}$, monomial and Newton bases

Sandia National Laboratories

My contributions

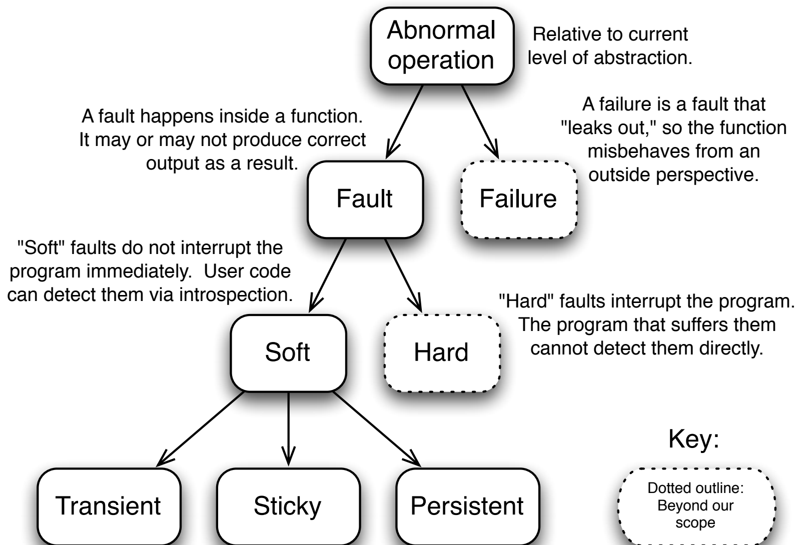
- Above technique: "*s*-step Krylov methods"
  - CG: Van Rosendale 1983, Chronopoulos and Gear 1989
  - GMRES: Bai et al. 1994, Joubert and Carey 1994
- Contributions in my thesis
  - Communication-avoiding matrix powers kernel, TSQR, BGS
  - New iterative methods and algorithmic improvements
  - Prototype shared-memory parallel implementation
    - Joint work with Marghoob Mohiyuddin (UC Berkeley)
  - New stability improvements for "*s*-step basis"
  - Incorporating preconditioning
- Post-doc contributions
  - Implemented hybrid-parallel TSQR in Trilinos
    - Improves block iterative methods as well
  - EASI-funded collaboration on matrix powers kernel

Sandia National Laboratories

Reliability of arithmetic and data

- Correct arithmetic & data cost energy
  - Redundant storage & computation (parity bits, RAID)
  - Communicating agreement (checksums, voting)
- Extreme-scale parallelism: correctness is costly
  - More components, so faults more likely
  - Extremely energy-constrained
    - High redundancy not acceptable
- Consumer applications drive hardware
  - Many consumer apps tolerate some faults
  - Mobile devices also energy-constrained

Terminology: fault vs. failure, soft vs. hard



Sandia National Laboratories

Current reliability model: Fail stop

- "Fail stop" means
  - System tries to detect all soft faults
  - Turn all detected soft faults into hard faults
  - Checkpoint / restart is the only recovery model
- Problem: Checkpoint / restart unacceptably expensive
  - Exascale mean time between failures: < 1 day
  - App may waste > half its time checkpointing
  - Checkpoints heavy on I/O, thus energy
- Problem: Overconstrains reliability...
  - ...because many current numerical algorithms do!
  - Iteration ⇒ *Latent* fault tolerance, but...
  - *Certain parts* require reliable computation

Sandia National Laboratories

Better reliability model: Sandbox

- Sandbox model
  - Isolate unreliable computation in a box
  - Reliable code invokes box as a function
  - App gets flexibility to define recovery model
- Additional desired model features
  - Detection: report faults to application
  - Transience: “refresh” unreliable data periodically
  - Type system embedding: let compiler help you

Sandia National Laboratories

Desired properties of a fault-tolerant iterative method

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

Sandia National Laboratories

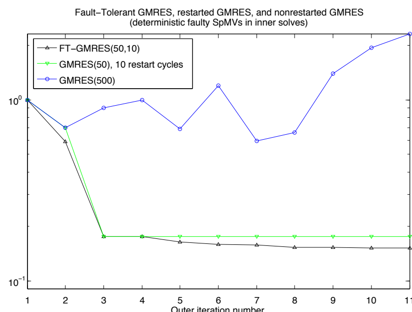
Origin of Fault-Tolerant GMRES

Inspired by existing algorithm: Flexible GMRES (FGMRES)

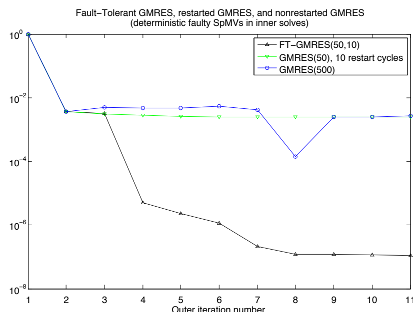
- FGMRES converges eventually
  - As long as Krylov subspace keeps growing
  - The algorithm tells you otherwise
- FGMRES allows changing preconditioner
  - Arbitrarily large changes allowed
  - Fault = "changing" preconditioner
- Make "preconditioner" your current solver & preconditioner
  - Can reuse software stack
  - Likely must adjust algorithmic parameters
- Fault-Tolerant GMRES (FT-GMRES) =
  - Flexible GMRES as an inner-outer iteration
  - Inner solves run unreliably, outer solver runs reliably
  - Expect inner solves to take most of the time

FT-GMRES can run 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).

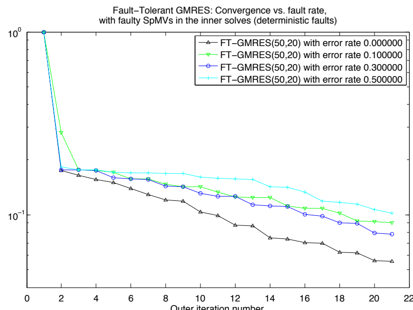


FT-GMRES vs. GMRES on mult_dcop_03 (a Xyce circuit simulation problem).

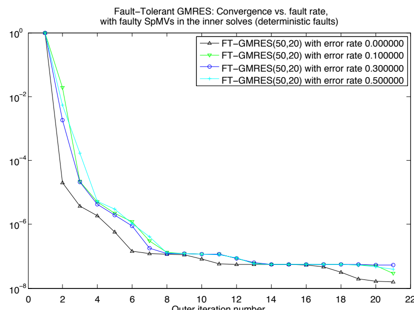
Sandia National Laboratories

Observed gradual degradation of convergence

- Empirical observation: FT-GMRES convergence slows gradually as fault rate increases.



FT-GMRES on Ill_Stokes problem, with different fault rates in inner solves' SpMVs.



FT-GMRES on mult_dcop_03 problem, with different fault rates in inner solves' SpMVs.

Sandia National Laboratories

Advantages of our approach

Existing approach:

- System overconstrains reliability
- "Fail-stop" model
- Checkpoint / restart
- App is ignorant of faults, but suffers from them

Our approach:

- System lets application control reliability
- Tiered reliability
- "Run through" faults
- Application listens for and responds to faults

See PDF at `http://www.sandia.gov/~maherou/`

Sandia National Laboratories

Future work (1 of 3)

In progress: Collaboration with systems researchers

- Current: allow ECC memory detect-no-correct faults
  - Current OS policy kills process; we don't
  - Decide memory reliability per-allocation
  - App can ask system whether faults occurred
  - Good proxy for all kinds of hardware faults
- Current: User-space fault injection
- Current: FT-GMRES performance prototype (Trilinos)
  - Custom Kokkos ("unreliable compute buffers")
  - Stock Tpetra, Ifpack2, Belos
- Future: integration with incremental checkpointing
  - Refresh "unreliable memory" from reliable backing store

Sandia National Laboratories

Future work (2 of 3)

- Near-term: Statistical performance experiments / model
  - Determine fault rate at which FT-GMRES pays off
  - Explore new hardware's energy / reliability trade-offs
    - Hardware / software co-tuning
- Medium-term: Study FT-GMRES convergence
  - Do first inner solves matter more than later ones?
    - Inexact Krylov analogy
    - Gradually relax reliability?
  - Co-tune inner and outer solves' parameters
  - Can we prove better than "eventual convergence"?
- Longer-term
  - Leverage fault detection
    - If no fault, inner solves need not restart
    - System may not detect all faults. . .
  - Mix in algorithmic fault detection

Sandia National Laboratories

Future work (3 of 3)

Future projects of larger scope:

- Develop other fault-tolerant algorithms
  - Multigrid
    - Smoothing? Updates? Coarse-grid solves?
  - Domain decomposition
    - Use overlap to force convergence despite faults?
    - Asynchronous ("chaotic") iteration ideas?
  - Nonlinear iterations
    - e.g., preconditioned Newton-Krylov
- Co-tune whole solver stack, based on expected fault rate

Fault tolerance summary

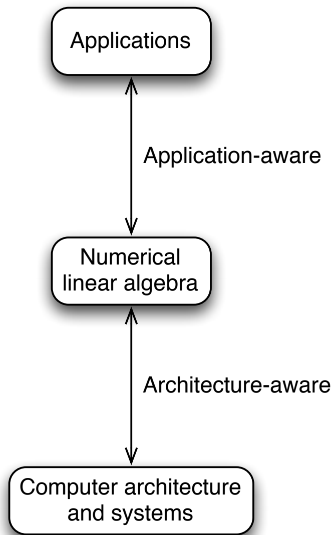
- Hardware reliability costs energy
- Current algorithms overconstrain reliablility
- Algorithm / system codesign approach:
  - System exposes on-demand reliability
  - Algorithms demand reliability only when needed
- Example: Fault-Tolerant GMRES (FT-GMRES)

Sandia National Laboratories

Questions?

Sandia National Laboratories

Numerical linear algebra as a dialogue

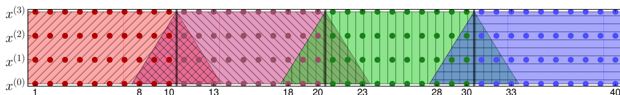


- What problem do you really want to solve?
- Mathematical structure

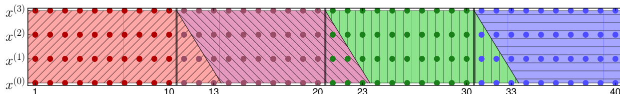
- Relative cost of operations
- Reliability promises
- System introspection

Matrix powers kernel

- First partition matrix and vectors into subdomains
- Expand boundary layers so each subdomain only has to ask neighbors once to compute *s* SpMVs' worth
- Parallel: redundant computation of boundary layers
- Sequential: not necessarily redundant
- Can nest parallel and sequential (we do!) in different ways



Parallel

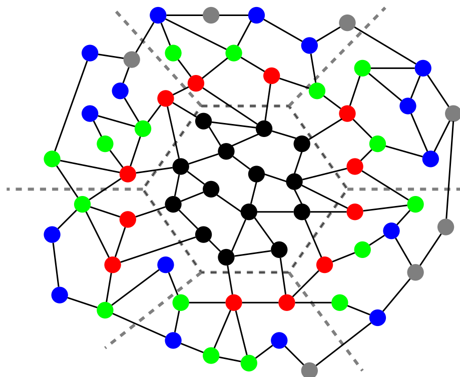


Sequential

Sandia National Laboratories

Not just tridiagonals, or regular meshes

- Works on general sparse matrices, but
- Only beneficial if low surface-to-volume ratio
- i.e., if graph partitioners work well
- Prior work: see SC09



Desired properties of a fault-tolerant iterative method

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

Sandia National Laboratories

Reliability models

Model → can reason about code behavior

Current model: Fail-stop

- System tries to detect all soft faults
- Turn all detected soft faults into hard faults
- Checkpoint / restart is the only recovery model

Our model: Sandbox

- Isolate unreliable computation in a box
- Reliable code invokes box as a function
- App gets flexibility to define recovery model

Additional desired model features

- Detection: report faults to application
- Transience: “refresh” unreliable data periodically
- Type system embedding: let compiler help you