Fault-tolerant iterative methods via selective reliability

SAND2011-8603C

Patrick G. Bridges, Kurt B. Ferreira,
Michael A. Heroux, and Mark Hoemmen

bridges@cs.unm.edu,
{kbferre,maherou,mhoemme}@sandia.gov

University of New Mexico and Sandia National Laboratories[1]

Supercomputing 2011
Seattle, WA
14 Nov. 2011

[1] Sandia National Laboratories 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

Quotes

"Parity is for farmers."

– Seymour Cray, on the CDC 6600

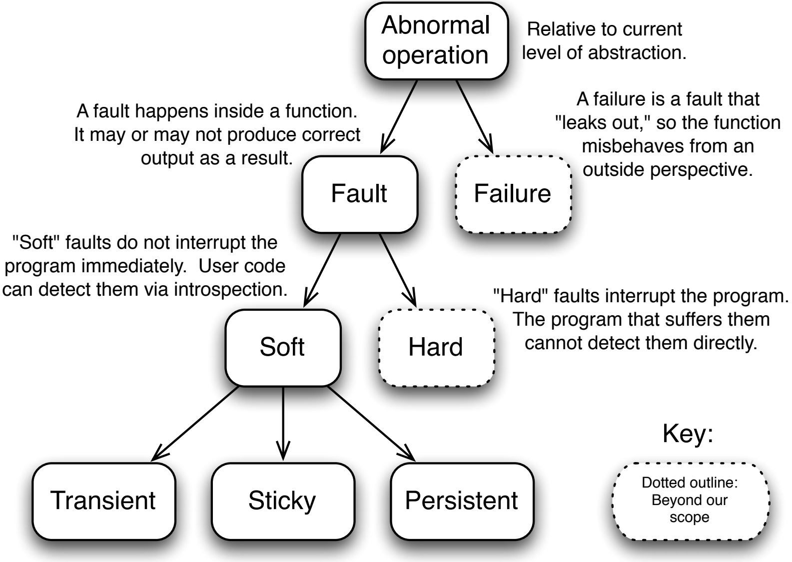
". . . I remarked to Dennis [Richie] that easily half the code I was writing in Multics was error recovery code. He said, 'We left all that stuff out. If there's an error, we have this routine called `panic`*, and when it is called, the machine crashes, and you holler down the hall, "Hey, reboot it." ' "*

– Tom van Vleck, Multics developer

Motivation

- Correct arithmetic & data cost energy
  - Redundant storage & computation
  - Communicating agreement (checksums, voting)
- Extreme-scale parallelism: correctness is costly
  - More components, so faults more likely
  - Extremely energy-constrained
- Consumer applications drive hardware
  - Many consumer apps tolerate some faults
  - Mobile devices also energy-constrained
- Current numerical algorithms overconstrain reliability
  - Latent fault tolerance, but. . .
  - Certain parts require reliable computation

Fault terminology



Sandia National Laboratories

Reliability models

Model $\rightarrow$ 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

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

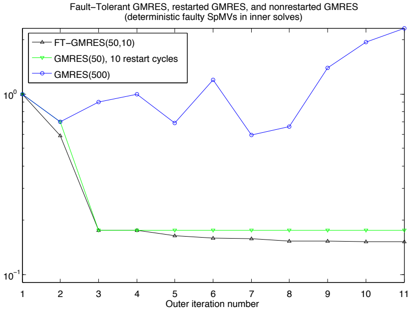
Origin of Fault-Tolerant GMRES

Inspired by existing algorithm: Flexible GMRES (FGMRES)

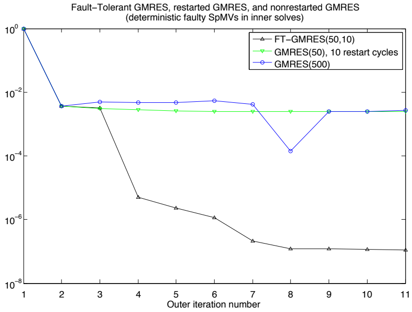
- FGMRES converges eventually
  - As long as Krylov subspace keeps growing
  - The algorithm tells you otherwise
- FGMRES allows changing preconditioner
  - Arbitrarily large changes allowed
  - Fault = "changing" preconditioner
- Make "preconditioner" your current solver & preconditioner
  - Can reuse 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).

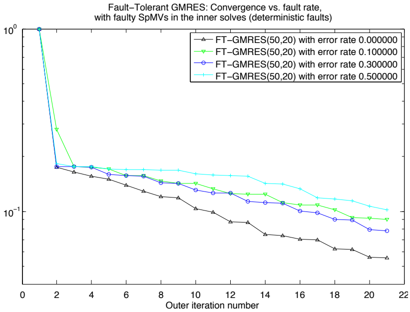


FT-GMRES vs. GMRES on mult_dcop_03 (a Xyce circuit simulation problem).

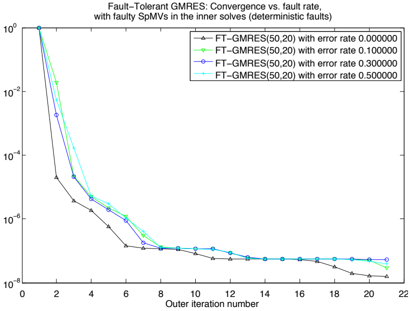
Sandia National Laboratories

Observed gradual degradation of convergence

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



FT-GMRES on III_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.

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/`

Performance prototype

Collaboration with systems researchers

- 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
- User-space fault injection
- FT-GMRES performance prototype (Trilinos)
  - Custom Kokkos ("unreliable compute buffers")
  - Minor modifications to Tpetra, Belos, and Ifpack2
- Future: integration with incremental checkpointing
  - Refresh "unreliable memory" from reliable backing store

Future work (1 of 2)

- 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
  - Better leverage fault detection
    - If no fault, inner solves need not restart
    - System may not detect all faults. . .
  - Mix in algorithmic fault detection

Future work (2 of 2)

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

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)

Extra slides

Fault-Tolerant GMRES (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 $q_j = Az_j$ ▷ The only unreliable part

  $v_{j+1} := Az_j$

  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**

      Global recovery (rolling back) is required

    else

      Cannot continue; 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