FTXS Workshop · 31 May 2016

SAND2016-4995C

In-Situ Mitigation of Silent Data Corruption in PDE Solvers

Maher Salloum, Jackson R. Mayo, and Robert C. Armstrong

Sandia National Laboratories, Livermore, CA 94551, USA

{mnsallo, jmayo, rob}@sandia.gov

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.

Problem: Future platforms will face tradeoffs imperiling correct hardware function

- Hardware correction already attempts to hide many "out-of-nominal" behaviors from the application
  - Error correction for bit flips in DRAM and caches is important and largely effective
- Increasing scale and constrained power may push toward exposing silent hardware errors (of possibly unexpected kinds) – corrupting an unaware application's results
- A primary concern is silent data corruption (SDC), where the computation appears normal except for wrong numerical values
  - Undetected DRAM errors at exascale for one type of ECC memory could be $\sim$1 per day
  - Low-voltage processors and accelerators will likely have increased rates of arithmetic errors; ECC doesn't protect data transformation

Objective: Enable practical SDC mitigation targeted at physics simulation

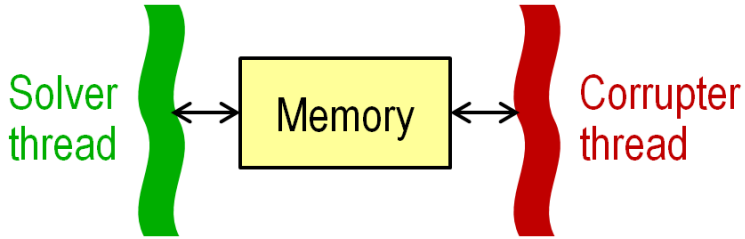
- For Advanced Simulation & Computing (ASC) codes, we seek better understanding of how to anticipate, mitigate, and/or diagnose the effect of silent errors
- Basic principles aim at building efficient algorithm-based fault tolerance (ABFT) for SDC into physics solvers
  - Predictive simulations of physical systems must already be robust to various numerical and statistical errors/uncertainties
  - Hardware errors can be damped in analogous ways
- Ultimate goal is to contribute to practical resilience toolbox for production codes
  - Leveraging existing partial differential equation (PDE) solver implementations and maintainable as they evolve
  - Adaptable to respond to future hardware characteristics
  - Flexible to unanticipated sources of silent errors

We are more focused on *application-level response* vs. other work on silent errors

- There has been much study of SDC detection techniques, including for PDEs, but the recovery mechanism is crucial
- Generic brute-force approach: triple modular redundancy with periodic voting to detect and extinguish any error
- Absent such replication, can detected SDC simply be treated like any other fault?
  - Recovery by rollback to a checkpoint – but:
  - Any false-positive detections will hinder forward progress
  - Doesn't take advantage of bulk of computation done correctly
- Our thesis: The dynamics of physical PDEs can support efficient ultralocal (within cache) detection and recovery, achieving stability to isolated occurrences of SDC
  - Handling silent errors quickly and transparently (like standard numerical errors) reduces the cost of a false positive

Initial error model describes memory bit flips, but can be generalized

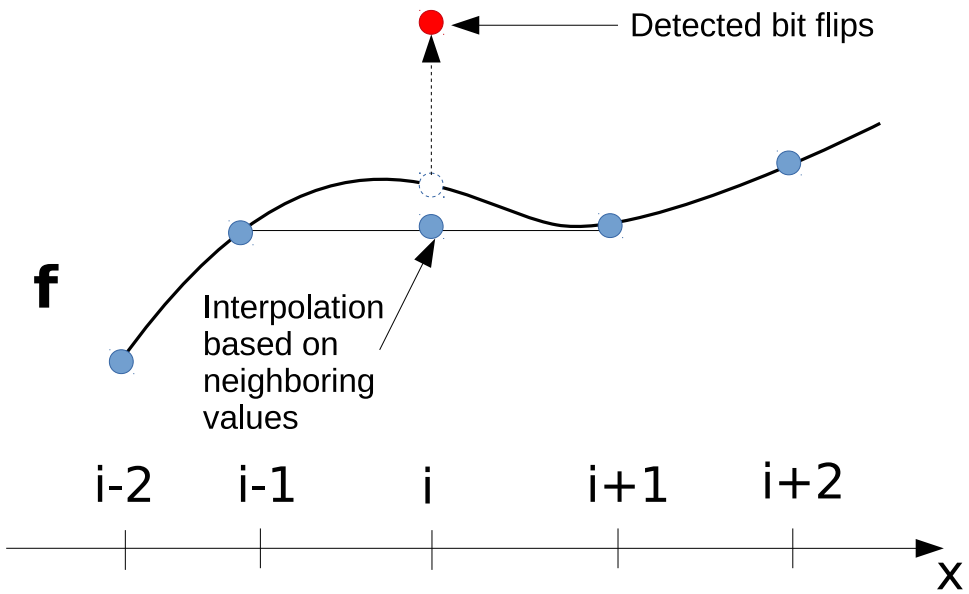
- In extreme-scale scientific computing, floating-point (FP) data are an obvious concern for SDC
  - FP data often constitute the bulk of memory usage
  - Data corruption can also be a proxy for *processor* FP glitches
  - Corruption in other places (control logic, pointers) is more likely to cause outright crashes, which will be mitigated by other means
  - Relaxing FP correctness can benefit accelerators such as GPUs
- Our error-injection framework for solvers: Asynchronously perform raw memory bit flips in the FP solution array



- Other injection and mitigation techniques will be pursued for related sources of SDC such as processor arithmetic errors

Outlier detection and interpolation can leverage smoothness properties of physics

- Aim is to correct accumulated bit flips in data values when they are loaded from memory, just before they are used – so that large corruptions will not propagate
- Corrupted values are detected via relative deviation from neighbors and replaced with an interpolation, as a linear-algebra-type computation is sweeping the vector



A nonlinear advection equation can be solved by a variant of sparse matrix-vector

- The 1D *linear* advection equation

$$\partial\phi/\partial t + \nu\,\partial\phi/\partial x = 0$$

  can be solved by Lax-Wendroff finite-difference stencil based on CFL number $c = \nu\,\Delta t/\Delta x$

- The shock-forming Burgers equation

$$\partial\phi/\partial t + \partial\left(\tfrac{1}{2}\phi^2\right)/\partial x = 0$$

  can be solved by corresponding stencil with local CFL number $\gamma_j^n = \phi_j^n\,\Delta t/\Delta x$ (Roe 1980)

$$\phi_j^{n+1} = \left[\tfrac{1}{2}\gamma_j^n + \tfrac{1}{2}(\gamma_j^n)^2\right]\phi_{j-1}^n + \left[1-(\gamma_j^n)^2\right]\phi_j^n + \left[-\tfrac{1}{2}\gamma_j^n + \tfrac{1}{2}(\gamma_j^n)^2\right]\phi_{j+1}^n$$

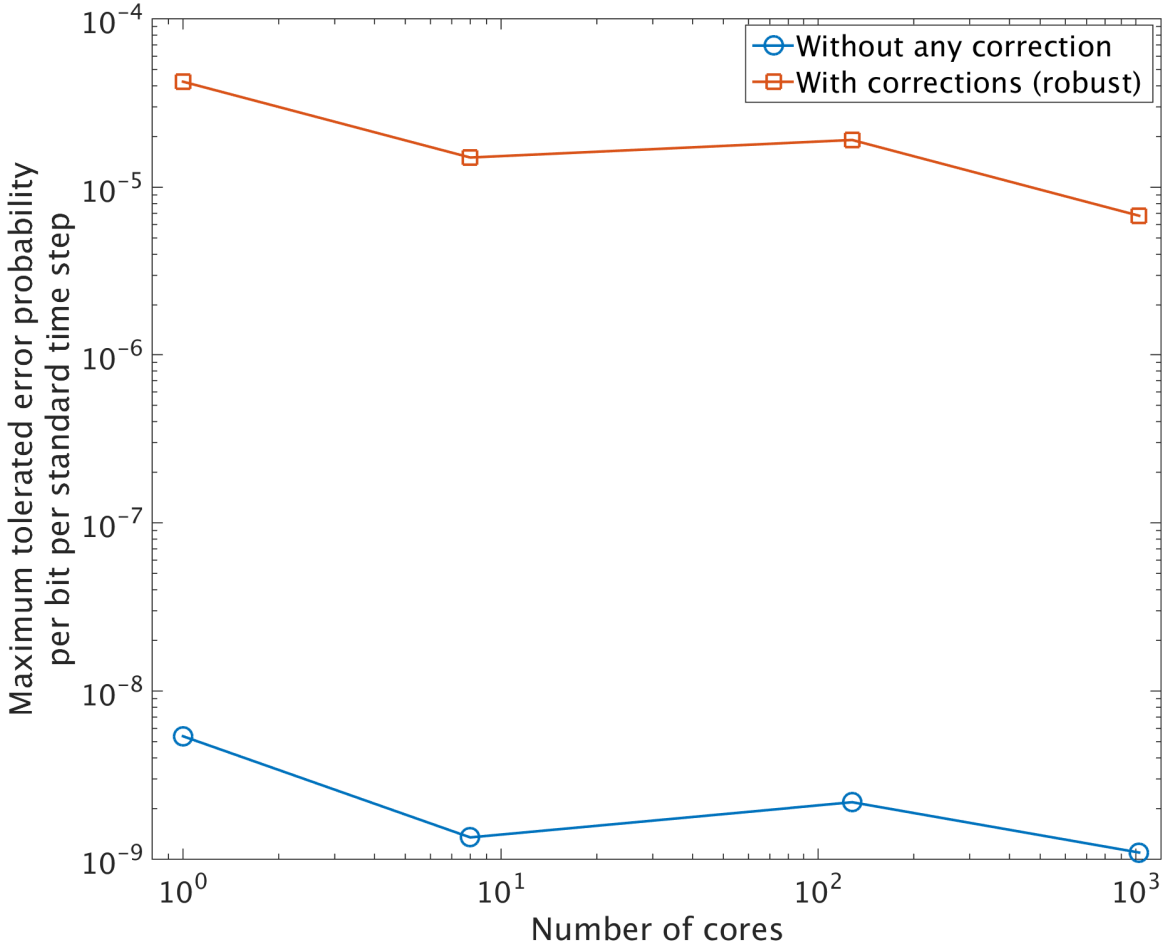
- Interpolation is performed as the operation sweeps the vector ϕ^n (robust nonlinear version of sparse matrix-vector)

Effect of bit-flip injection and mitigation is measured via error norm and wall time

- Corrupter thread injects random bit flips based on a probability parameter
- Each solver run tracks the total wall time and number of bit flips, used to calibrate the emulated memory error rate
  - For ease of comparison, the unit of wall time is taken as *one timestep* of the standard solver on one core
    - Or, for later conjugate gradient example, *one iteration*
- Time-based memory error model means that a slower solver incurs a penalty in resilience because more bit flips accumulate
- Measure wall time to solution, and final-state error norm
  - Explicit solver is considered as tolerating the bit flips if this norm is less than $3\times$ that of the standard solver with no bit flips
    - Or, for later conjugate gradient example, iterative solution is simply required to converge within a predefined residual (may diverge instead if bit-flip rate is too high)

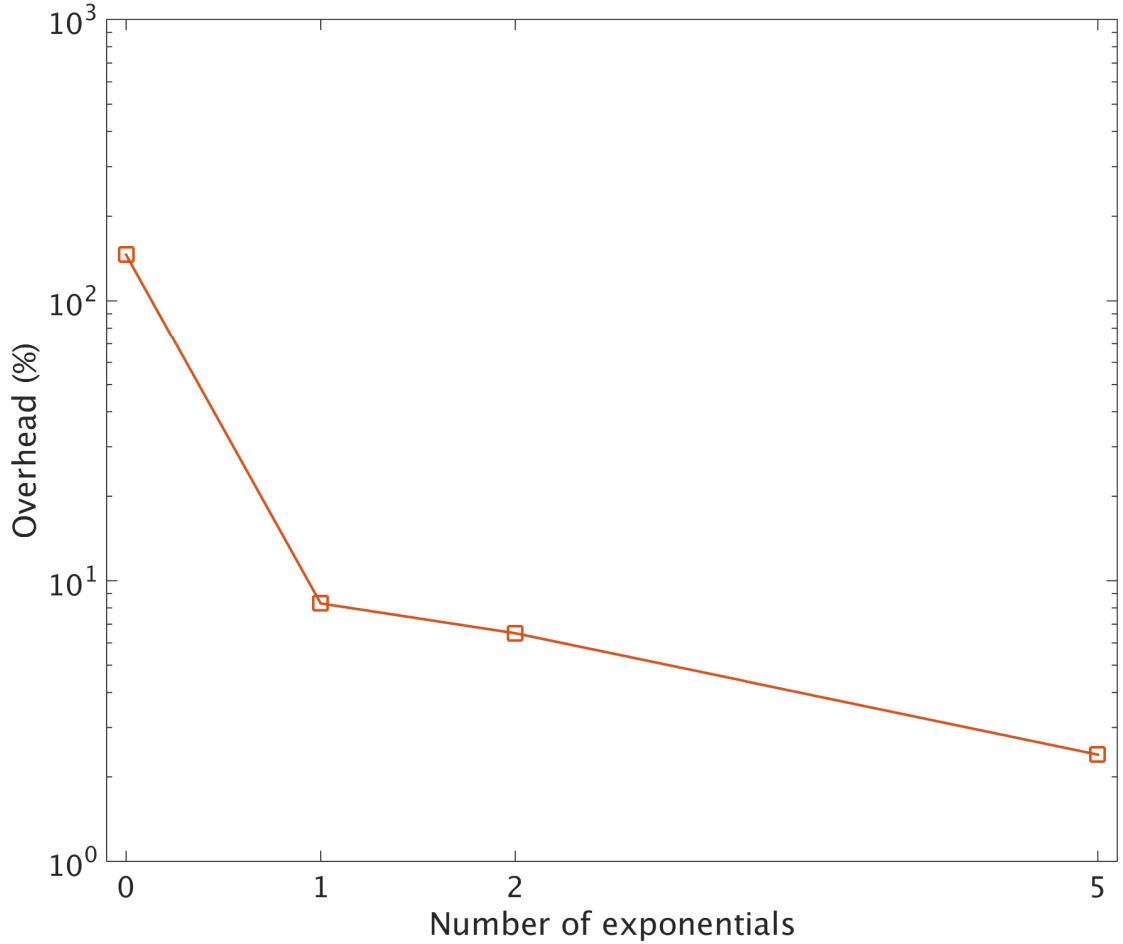
Interpolation maintains strong SDC-tolerance improvement under weak scaling

- Robust hyperbolic solver tolerates error rates $\sim 10^4\times$ larger than standard solver



Robustness overhead for hyperbolic solver decreases with chemistry-like forcing terms

- Overhead of error detection/interpolation is masked to ∼2% by adding local computations to both original and robust schemes

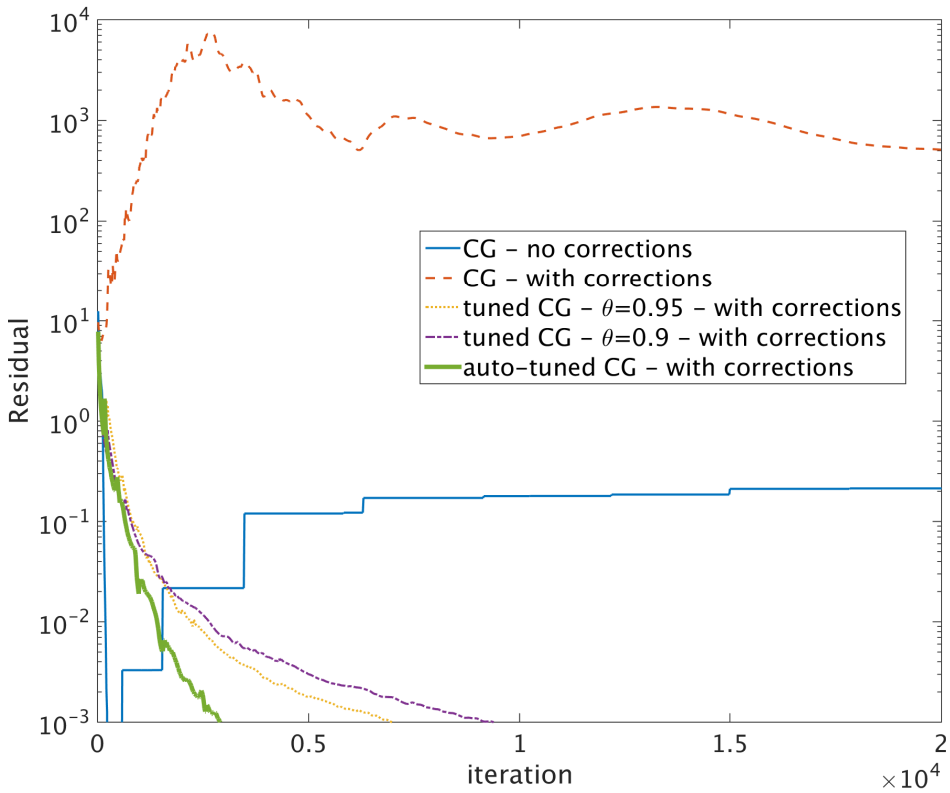


Example elliptic solver is an instance of conjugate gradient, built from linear algebra

- HPCCG mini-app uses unpreconditioned conjugate gradient (CG) method as solver for a linear elliptic PDE in a 3D domain
  - Vary problem scale by stacking "wafer" domains in one direction
- Each iteration of CG involves one *smpv* call, three *daxpy* calls, and two *ddot* calls
- More communication-intensive than explicit solvers, especially at large scale
- We demonstrate concept of robust linear algebra "building blocks" that can be reused in other PDE solvers
  - Step toward larger goal of easing resilience programmability for other custom PDE solver needs, such as nonlinear and stencil-like operations

CG modification helps promote convergence in the presence of SDC

- Key CG iterations: $u_{k+1} = u_k + \alpha p_k$ and $p_{k+1} = r_{k+1} + \beta p_k$
  - Less aggressive steps are obtained by $\alpha \leftarrow \theta\alpha$ and $\beta \leftarrow 1 - \theta(1 - \beta)$, where $\theta < 1$ is a function of the residual
  - This "tuning" enables convergence with interpolation – can even *accelerate* convergence of early iterations (behavior might differ with preconditioner)

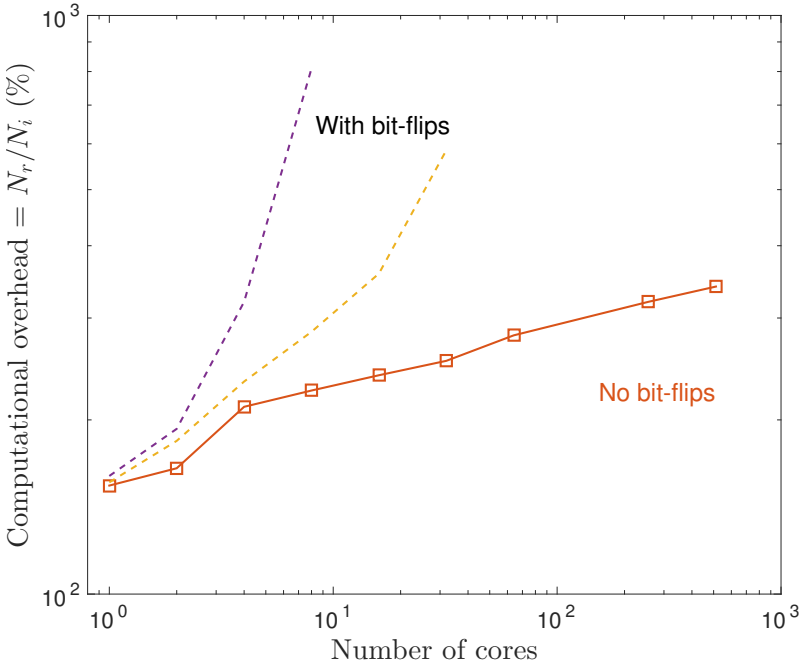


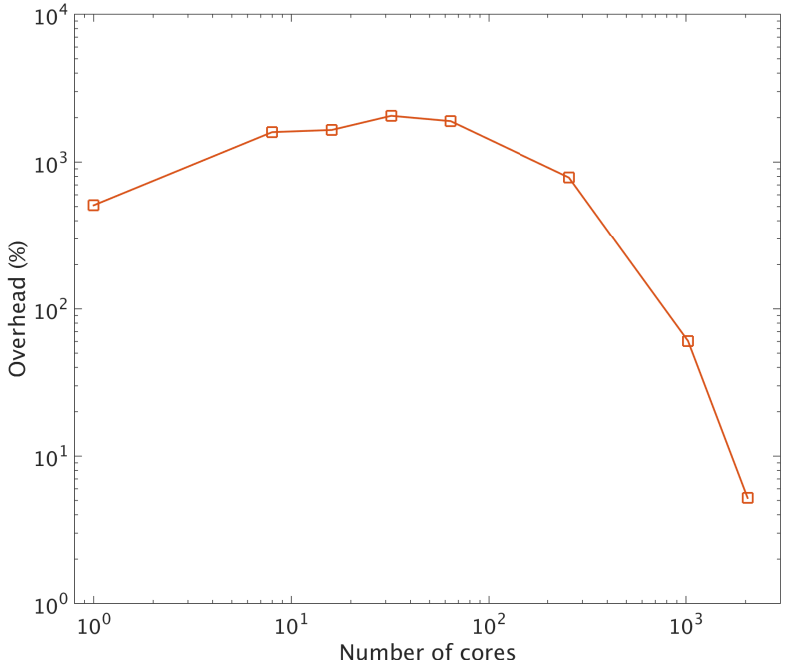
We consider checkpoint-based mitigation as a comparison

- Checkpointing is a standard practice for resilience – can we take advantage of it?
- For recovery from SDC by rollback to a checkpoint, would like:
  - SDC detected as soon as possible (while it's still a clear outlier)
  - Checkpoint as recent as possible (to minimize lost progress)
- Key problem: For PDE solvers, there is no simple definitive test for SDC – hardware errors are on a spectrum with the normal discretized solver behavior
  - The only "acid test" is recomputation ($> 100\%$ overhead)
- While many strategies are possible, we implemented simple global checkpoint/restart for HPCCG
  - Checkpoint very frequently (every iteration) and assume zero checkpoint cost (exclude I/O time)
  - Use same SDC *detection* as in our other implementations

Roll-forward interpolation appears more scalable: Reduces false-positive cost

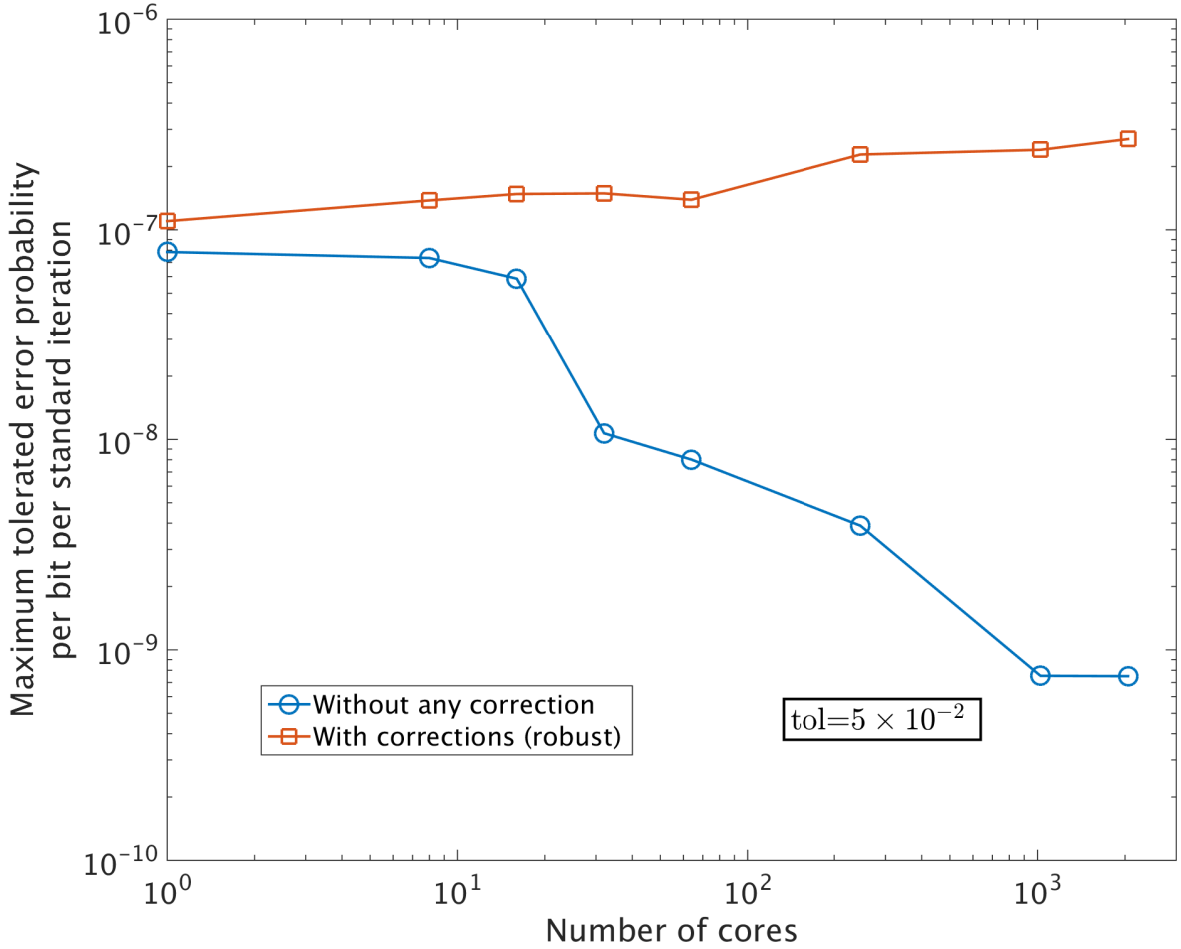
- Checkpoint-based (rollback): Even with checkpointing assumed free, false positives lead to frequent costly restarts
- Interpolation-based (roll-forward): At larger scale, interpolation overhead declines due to communication dominance





Interpolation is effective in tolerating bit flips at larger scale

- Robust CG method can tolerate higher SDC rates that prevent standard method from converging at all



This work helps confront likely errors to make extreme-scale HPC workable

- Scale will eventually cause previously negligible hardware errors to dominate, likely from unanticipated sources and modes
- Two fundamental ways to mitigate these errors
  1. Traditional: enhanced hardware correction – may be slow, costly
  2. As part of the algorithm for computation – leveraging characteristics of the simulation problem (e.g., smoothness)
- Goal: Mitigate errors in-situ as an add-on to the computation
- Benefits reliability, speed, and cost of next-gen computation
  - Mitigates errors that might be unpredictable at extreme scale
  - Provides a diagnostic capability to detect silent errors
  - Guides choices of both hardware & solver algorithms (co-design)
- Future work will improve maintainability of resilient solvers, and demonstrate them at larger scale and with broader error models