SAND2016-3591C

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

Abstract

We present algorithmic techniques for parallel PDE solvers that leverage numerical smoothness properties of physics simulation to detect and correct silent data corruption within local computations. We initially model such silent hardware errors (which are of concern for extreme scale) via injected DRAM bit flips. Our mitigation approach generalizes previously developed "robust stencils" and uses modified linear algebra operations that spatially interpolate to replace large outlier values. Prototype implementations for 1D hyperbolic and 3D elliptic solvers, tested on up to 2048 cores, show that this error mitigation enables tolerating orders of magnitude higher bit-flip rates. The runtime overhead of the approach generally decreases with greater solver scale and complexity, becoming no more than a few percent in some cases. A key advantage is that silent data corruption can be handled transparently with data in cache, reducing the cost of false-positive detections compared to rollback approaches.

Keywords: Silent data corruption; resilience; algorithm-based fault tolerance; partial differential equation; numerical methods

1 Introduction

The increasing scale of HPC will eventually cause previously negligible hardware errors to become important. Some of these errors will likely arise from unanticipated sources and modes, including silent (undetected) errors [3]. Attempting to hide such faults through enhanced hardware error correction may make computing platforms slower and/or more expensive. Thus, future hardware may regularly expose silent errors that can corrupt application results.

A primary concern is silent data corruption (SDC), which leads to wrong numerical values in a computation that otherwise appears normal. Though empirical information on real silent errors in HPC is limited, a recent study [14] measured DRAM errors on a current machine to estimate how many would be undetected by one type of error-correcting memory (SECDED). Resulting

extrapolated SDC rates at exascale are of the order of one per day. Besides in DRAM, SDC could occur in processing, communication, and storage.

Algorithm-based fault tolerance [2] (ABFT) is a promising strategy that seeks to efficiently mitigate the effects of hardware errors, including SDC, at the software level for particular kinds of applications. The effectiveness of ABFT is dependent on the characteristics of the application (such as numerical stability properties) and on the assumed hardware error model.

We specifically target physics simulation applications that solve partial differential equations (PDEs). We also initially assume an SDC model consisting of DRAM bit flips. This paper builds on previous "robust stencils" for explicit finite-difference solvers [9, 15], and describes the more general use of robust versions of vector and sparse-matrix operations representing discretized physical space, which are typical ingredients of PDE solvers. This contrasts with other efforts on SDC-tolerant linear algebra operations and linear solvers [12, 4, 5] that do not attempt to leverage the dynamics of physical PDEs. Some work on SDC-tolerant solvers for hyperbolic PDEs [7] has similarity to the robust stencil approach but does not systematically evaluate solver behavior under an error model. Other work on localized detection of SDC based on physical PDE dynamics [16, 1, 13] does not propose a recovery mechanism other than rollback and recomputation.

Our approach seeks algorithm-level *stability* to isolated occurrences of SDC, via detection and recovery that ideally occur in cache as a low-cost add-on to the numerical computation. Here we describe preliminary implementations and evaluations of this in-situ approach, demonstrating its feasibility and showing the potential for extension to physics solvers of increasing scale and realism.

2 Methodology

2.1 Operations in PDE Solvers

Physical systems are often modeled by partial differential equations (PDEs) which, when discretized, result in a system of equations with a large yet finite number of unknowns [11]. Algorithms to solve for these variables can typically be expressed in terms of linear systems of equations $Au = b$ with a large sparse matrix A. Modification of variables often takes place using basic linear algebra operations [8] such as dot products (*ddot*), vector sums (*daxpy*), sparse matrix-vector products (*spmv*), etc.

Our team developed SDC-tolerant algorithms for the explicit finite-difference solution of linear and nonlinear advection equations [9]. The algorithm developed for the parallel solution of a 1D advection problem can be viewed as computing a relevant sparse product Au using a matrix-free scheme. On this basis, we have generalized the robust stencil technique to a broader class of PDE solvers via robust linear algebra operations, which we now describe.

2.2 Error Model and Mitigation Approach

The hardware SDC model we have considered to date consists of independent bit flips in DRAM occurring uniformly in space and time (Poisson process), parameterized by the error probability (rate) per bit with respect to wall time. We have tested standard and modified PDE solvers in a C++/MPI framework that allows evaluating a variety of algorithms on HPC platforms to understand the scaling of key resilience properties, e.g., against SDC. Each MPI task consists of a solver thread in addition to a corrupter thread that asynchronously performs raw memory bit flips in the floating-point solution arrays. This closely approximates the memory error model because the floating-point arrays constitute the vast bulk of the memory usage in the solvers considered. A separate core is allowed for each thread so that the corrupter threads have as little impact as possible on the behavior of the solver threads except for the errors they inject.

We seek to develop a technique to correct this type of SDC when performing linear algebra operations in PDE solvers. The correction method is based on numerical interpolation to replace suspected corrupt points with values computed from neighboring points. Our goal is that, when an isolated, potentially corrupted floating-point value is read from memory to perform an operation, no error that is numerically large (i.e., significantly larger than existing discretization errors) will propagate to subsequent steps in the solver algorithm. We can meet our goal provided that the assumed-reliable processor operations, performed on values in the registers and cache, correct the erroneous value (to within discretization error) before results are written back to memory. That is, the appropriate algorithmic building blocks for resilience are those in which each input value is read from memory *once*, computations are performed using registers/cache, and resulting output values are written. This is exactly how basic linear algebra operations are implemented, for performance reasons.

A key advantage of this approach is that, since isolated errors are mitigated throughout the algorithm, a catastrophic breakdown in accuracy should occur only when two or more errors occur in close proximity, such that one contaminates the correction of the other. This resilience property is analogous to that of (expensive) triple modular redundancy.

Consider a data vector $f(x)$ that is a smooth field in the physical domain where it is defined. An operation on f such as *daxpy* involves a loop on points x_i. During the loop, if any value $f(x_i)$ is detected as outlier, it is interpolated based on values at spatially neighboring points as illustrated in Figure 1. A value $f(x_i)$ is deemed an outlier if it is NaN or Inf, or if its deviation from a set of neighboring points exceeds a threshold (chosen empirically and measured relative to the variation in the neighboring points). For smooth fields without corruption, large outliers tend not to occur, so interpolation is not required for most points when corruption is rare.

This bit-flip detection and correction method is applicable to linear algebra and array-based computations defined on spatial meshes. We apply this strategy to two problems as a proof of concept for systematically improving SDC tolerance in large-scale solvers.

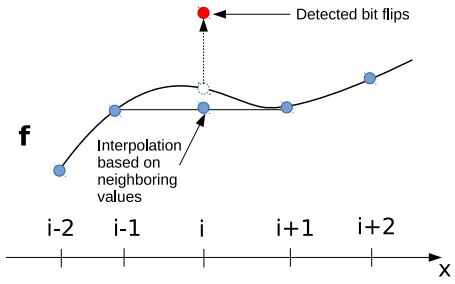

Figure 1: Schematic of a data vector $f(x)$ with a corrupted entry (red point). If this corruption is detected as an outlier, it is interpolated based on neighboring values. This is performed as the linear algebra computation is sweeping the vector.

3 Application to the 1D Burgers Equation

Consider a 1D hyperbolic PDE

$$\frac{\partial \phi(t,x)}{\partial t} + \nu \frac{\partial F[\phi(t,x)]}{\partial x} = 0, \tag{1}$$

where ν is a constant. Assume a standard finite-difference discretization that approximates the field at grid points in time and space,

$$\phi_j^n \simeq \phi(n\,\Delta t, j\,\Delta x). \tag{2}$$

If $F(\phi) = \phi$, then Eq. (1) is the classical linear advection equation, and the CFL number is defined as the constant

$$c = \nu \frac{\Delta t}{\Delta x}. \tag{3}$$

An explicit finite-difference scheme for the linear advection equation is determined by a stencil

$$\phi_j^{n+1} = \sum_{m=-l}^{l} A_m(c) \phi_{j+m}^n, \tag{4}$$

taken to be linear and to depend on a $(2l+1)$-point neighborhood. Eq. (4) can be thought of as a linear algebra operation (sparse matrix-vector product), and can be modified so that the operation performs interpolation while sweeping the vector ϕ^n.

We previously developed a library of stencils of the form (4) that can filter corrupted data due to silent errors while solving 1D hyperbolic equations. For example, we derived a family consisting of the standard three-point Lax-Wendroff stencil and two other stencils that omit some of the three points in favor of information from other neighboring points [9].

3.1 Roe's Formalism

The stencils (or equivalently, robust linear algebra operations) we developed were then transformed to solve nonlinear advection equations by adapting the technique of Roe [10]. For scalar hyperbolic PDEs, the idea behind Roe's formalism is that the coefficients A_m in Eq. (4) are computed as a function of the local CFL number defined as

$$\gamma(x,t) = \frac{\nu \Delta t}{\Delta x} \frac{\partial F}{\partial \phi}. \tag{5}$$

As such, Eq. (4) is written as

$$\phi_j^{n+1} = \sum_{m=-l}^{l} A_m(\gamma_j^n) \phi_{j+m}^n, \tag{6}$$

so that the solution is updated at each time step by sweeping the vector ϕ^n similarly to the linear case with the difference that the CFL number has to be computed at each data point. Furthermore, the same interpolation approaches developed in the linear case are applicable here.

3.2 Evaluation using In-Situ Interpolation

Consider Eq. (1) with $F(\phi) = \phi^2/2$, which corresponds to the shock-forming Burgers equation

$$\frac{\partial \phi}{\partial t} + \nu \phi \frac{\partial \phi}{\partial x} = 0. \tag{7}$$

We solve this equation on a large 1D domain distributed among different processors. Using Roe's formalism, we compute the local CFL number

$$\gamma_j^n = \frac{\nu \Delta t}{\Delta x} \phi_j^n, \tag{8}$$

and the solution at time step $n+1$ is given by the generalized Lax-Wendroff stencil

$$\begin{aligned} \phi_j^{n+1} = &\left[\tfrac{1}{2}\gamma_j^n + \tfrac{1}{2}(\gamma_j^n)^2\right]\phi_{j-1}^n + \left[1 - (\gamma_j^n)^2\right]\phi_j^n \\ &+ \left[-\tfrac{1}{2}\gamma_j^n + \tfrac{1}{2}(\gamma_j^n)^2\right]\phi_{j+1}^n. \end{aligned} \tag{9}$$

We assume periodic boundary conditions, and solve Eq. (7) while injecting bit flips in the vector ϕ at different rates, with and without using the interpolation-based correction strategy described above. The initial condition is

$$\phi^0 = -0.1\pi \cos(2\pi x) + 0.21\pi \sin(42\pi x). \tag{10}$$

We perform weak-scaling computations, i.e., we increase the spatial size of the problem proportionally with the number of computing cores. Communication between processors takes place at the domain boundaries, where values of ϕ are exchanged. Runtimes are nearly independent of problem size under weak

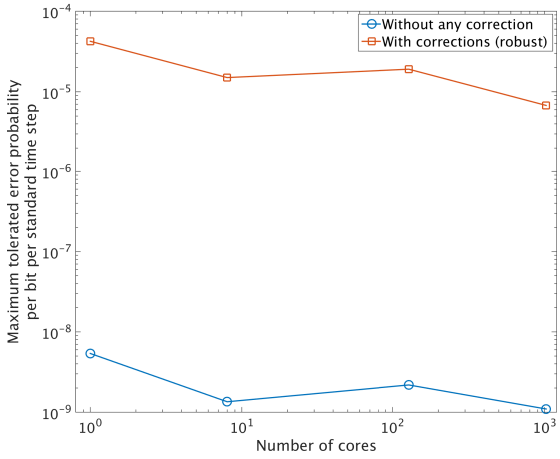

Figure 2: Weak-scaling results for the 1D Burgers equation solved using a Roe scheme, showing the maximum tolerated error probability as a function of the number of cores. The solution is computed for 160 time steps on 10^5 grid points per core with $\nu = 1$, $\Delta x = 5 \times 10^{-4}$, and $\Delta t = 2.5 \times 10^{-4}$.

scaling for this low-communication explicit scheme. The maximum tolerated error rates (defined by the final-state error norm reaching 3 times that of the standard solver with no bit flips) are plotted in Figure 2. The correction strategy enables error rates about four orders of magnitude larger than a standard solver.

Runtime overhead for the robust solver versus the standard solver is due to the operations needed to perform the detection and the additional computations when interpolating over a larger size stencil. In our memory error mitigation approach, the absolute amount of this overhead is driven by the basic structure of the equation (scalar advection) and does not depend on other local details such as boundary conditions, source terms, or constitutive laws. For instance, in realistic simulations, Eq. (1) may have a computationally expensive forcing term on the right-hand side in both the original and robust schemes. In this case, the effect of the robustness overhead would relatively decrease. We demonstrate this by introducing exponential forcing terms into Eq. (1) computed at each mesh point at each time step. Such terms are analogous to chemical reaction rates in the S3D turbulent combustion solver [18], where the typical number of reactions per chemical species would translate to approximately five exponential terms per state variable. Measurements (see Figure 3) show that the resulting relative overhead decreases, reaching $\sim 2\%$ for five exponentials.

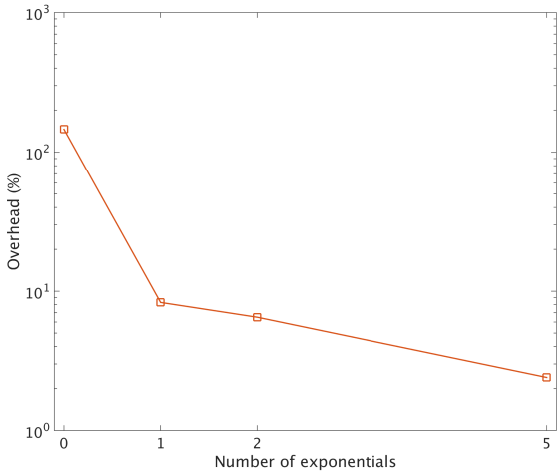

Figure 3: The overhead induced by the robust Roe scheme used to solve the 1D Burgers equation when additional exponentials are computed at each mesh point as a "forcing" term. The equation is solved on 128 cores and the solution is computed for 160 time steps on 10^5 grid points per core with $\nu = 1$, $\Delta x = 5 \times 10^{-4}$, and $\Delta t = 2.5 \times 10^{-4}$.

4 Application to the Conjugate Gradient Solver

The building blocks of robust linear algebra operations we develop are applicable to a wide variety of iterative solvers. Here we report the performance of the building blocks on a conjugate gradient (CG) algorithm for a linear system of equations, used in solving an elliptic PDE.

4.1 The Basic Conjugate Gradient Solver

The basic CG solver is given in Algorithm 1 [11]. It is used when A is symmetric and positive definite, which is the case in elliptic PDE problems. At each iteration, CG involves one *spmv* call, three *daxpy* calls, and two *ddot* calls.

Algorithm 1 Conjugate gradient method for a linear system of equations $Au = b$.

$u_0 = 0$ {Initial guess of the solution}
$r_0 = b - Au_0$, $p_0 = r_0$ {Initial residual and direction vectors}
$(R_0)^2 = r_0^T r_0$
for $k = 0$ until convergence **do**
  $q_k = Ap_k$
  $\alpha = (R_k)^2 / \left(p_k^T q_k\right)$
  $u_{k+1} = u_k + \alpha p_k$
  $r_{k+1} = r_k - \alpha q_k$
  $(R_{k+1})^2 = r_{k+1}^T r_{k+1}$
  $\beta = (R_{k+1})^2 / (R_k)^2$
  $p_{k+1} = r_{k+1} + \beta p_k$
end for
Return u_{k+1}

The CG method computes two scalar coefficients α and β governing the variation of the solution between consecutive iterations: α is the multiplier on the direction vector p used to update the solution u, and β is the ratio of two consecutive residual norms. A large α and small β mean that the algorithm is taking large steps toward convergence.

When run in parallel, CG requires communication between the compute nodes during the *ddot* and *spmv* steps. The CG method is more communication-intensive than an explicit scheme, especially at large scale, due to the presence of collective operations.

Our target solver is the HPCCG 1.0 mini-app [6], which uses the CG method to solve an elliptic PDE in a 3D domain of size $N_x \times N_y \times PN_z$, where P is the number of processes. This entails solving a linear system of equations $Au = b$, where A is a sparse heptadiagonal matrix of size $\left(PN_xN_yN_z\right)^2$ equally distributed among the P processes. We run weak-scaling computations such that each process's portion of the vector u is of fixed size $N_xN_yN_z$. The full solution vector is formed by stacking the domains of each process in the z-direction.

4.2 Controlling Convergence of the CG Algorithm

The CG algorithm relies on two scalars α and β that govern the solution steps toward convergence. With these coefficients, the method achieves a rapid convergence rate on a reliable machine, but we find that it can become unstable when using our interpolation-based robust linear algebra operations to mitigate SDC throughout the computation. We propose a method to control the CG step size: Define a "tuning parameter" $\theta \leq 1$ as a scaling factor on α. If $\theta < 1$, the step size is decreased and the CG convergence rate is decreased. Since α and β are related, we apply a corresponding scaling to drive β toward 1 as α

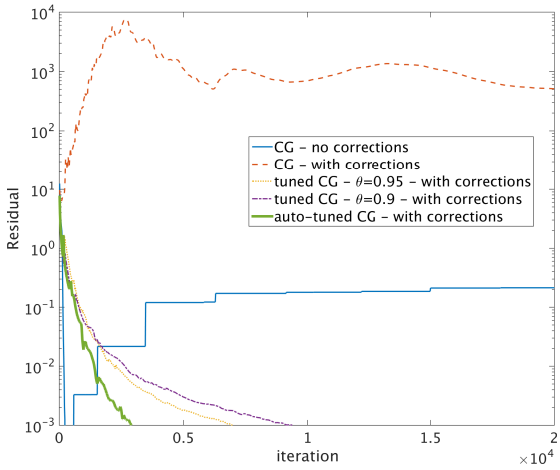

Figure 4: Convergence of a CG solver under emulated memory bit flips with different tuning strategies. The original CG solver diverges as-is (blue curve) and shows further instability when using corrective interpolation techniques throughout (red curve). Imposing a constant value of the tuning parameter θ (orange and purple curves) results in a convergent solver. Automatically tuning θ as a function of the residual (green curve) further accelerates convergence ($\theta_1 = 0.975$, $\theta_2 = 0.8$, and $R_c = 0.001$). Results are generated at a bit-flip rate of 2.3×10^{-8} per standard iteration for the HPCCG elliptic PDE on 32 cores with $64 \times 64 \times 4$ degrees of freedom per core.

is reduced. We therefore propose in Algorithm 1, after computing α and β, the additional steps $\alpha \leftarrow \theta\alpha$ and $\beta \leftarrow 1 - \theta(1 - \beta)$. For $\theta = 1$, these additions have no effect on the original CG algorithm.

Without error correction and without injected bit flips, we find that a slight decrease in θ below 1 results in a large deceleration of convergence, but makes the residual initially decrease *faster* in the early iterations. We can further control the convergence of the CG algorithm by varying the tuning parameter θ as the solution is evolving towards convergence. Therefore, we propose using $\theta < 1$ early in the solution process and then gradually increasing θ toward 1 based on the residual R. We compute θ as a linear interpolation between a smaller value θ_2 at the large initial residual R_0 and a larger value θ_1 when the residual reaches a small value R_c. Thus, θ is expressed as

$$\theta = (\theta_1 - \theta_2)\frac{R_0 - R}{R_0 - R_c} + \theta_2 \tag{11}$$

The benefit of automatically tuning the parameter θ can be seen in Figure 4. The CG algorithm with an automatically tuned parameter θ according to Eq. (11) converges two to three times faster than with a constant tuning parameter.

One reason that HPCCG is not a fully representative solver is that a production CG implementation would likely use a preconditioner to promote convergence. The comparison of robust and standard solver behavior might change if a realistic preconditioner were used.

4.3 Evaluation using Checkpoint/Restart Files

As a point of comparison, we tested our outlier-based error detection approach when used with a checkpoint-based response strategy [17] that performs recomputation, rather than interpolation, to correct possible SDC. We implemented the use of checkpoint/restart files for the HPCCG 1.0 mini-app. A checkpoint file is dumped to disk once per iteration, and the outlier-based error detection described in Section 2.2 is performed at each matrix/vector operation. Any detected error prompts a restart of the current iteration by all computing processes, proceeding forward once either no error is detected or errors are detected in the same locations after recomputation (indicating a false positive, an apparent corruption that is actually correct). Figure 5 shows that the runtime overhead due to recomputation (i.e., with checkpointing time excluded for an optimistic evaluation) is more than 100% even in the absence of actual bit flips. This is because false positives are frequent enough that most iterations require recomputations to confirm that corruption has not occurred. The overhead increases with scale as more false positives occur. These results indicate that standard checkpointing alone, even if it could be done very frequently at negligible cost, may not support efficient mitigation by rollback in the presence of false positives in error detection. Less frequent checkpointing would incur similar false-positive overhead from costlier rollbacks.

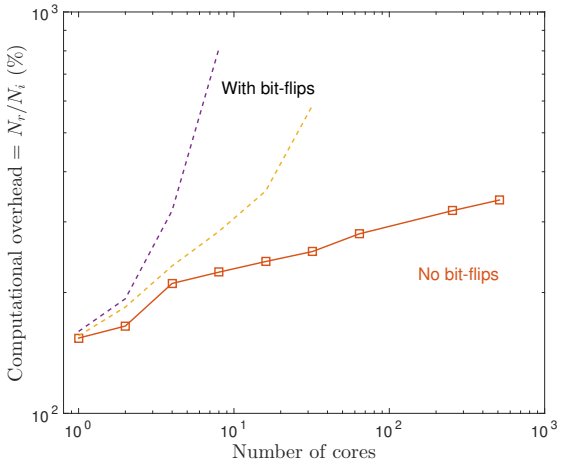

Figure 5: Weak-scaling results for the CG solver in a simple checkpoint/restart framework, showing the computational overhead evaluated via the iterations required including initial computation (N_i) and recomputation (N_r), versus those required with no restarts (N_i). Checkpointing I/O time is optimistically excluded. The solid curve corresponds to the case when no bit flips are injected. The dashed curves show the results for two finite bit-flip rates. Results are generated for the HPCCG elliptic PDE with $64 \times 64 \times 4$ degrees of freedom per core and a chosen convergence threshold (residual of 5×10^{-2}).

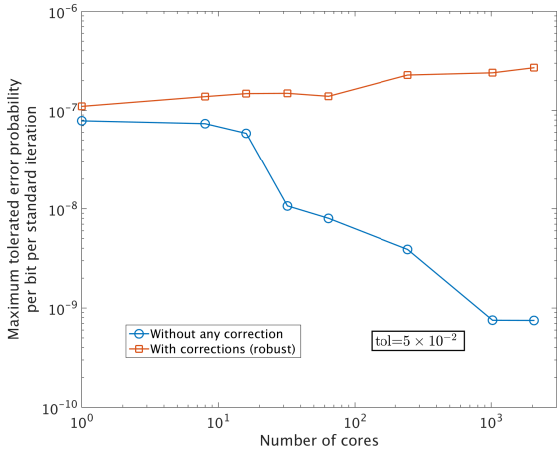

Figure 6: Maximum tolerated error probability for the CG solver with and without interpolation.

4.4 Evaluation using In-Situ Interpolation

In contrast to rollback techniques, the roll-forward error mitigation that we describe in this paper helps reduce the cost of false positives. This mitigation technique has been implemented for the HPCCG 1.0 mini-app. We inject bit flips in all u, p, q, and r vectors involved in the CG method (see Algorithm 1). We report weak-scaling computations performed with the robust operations and the variable tuning parameter technique described in Section 4.2. We compare to the standard HPCCG implementation without use of robust linear algebra operations or the tuning parameter.

As in Section 4.3, the HPCCG elliptic PDE is solved with $64 \times 64 \times 4$ degrees of freedom per core, and the solver is required to reach an assumed acceptable accuracy (residual of 5×10^{-2}). A linear interpolation scheme is used to mitigate the corrupted data points. The weak-scaling results are plotted in Figures 6 and 7. The correction strategy enables increased tolerance of silent errors depending on the number of cores P, i.e., problem size. As shown in Figure 6, the maximum tolerated error rate is modestly higher than that of the standard solver for smaller problems with $P \leq 16$. However, the correction approach enables tolerating an error rate up to two orders of magnitude higher than a standard solver for larger problems. The robust solver converges in approximately the same number of iterations as the standard solver. Hence, as seen in Figure 7, the wall time shows decreasing overhead at larger scale from the robust solver operations – down to $\sim 5\%$ at 2048 cores – because the wall time becomes dominated by the communication cost of each iteration. Comparing Figure 7 with Figure 5, we see the advantage at large scale of avoiding restarts due to false-positive detections. Also, we reemphasize that according to Figure 6, the robust method can tolerate higher error rates that prevent the standard method

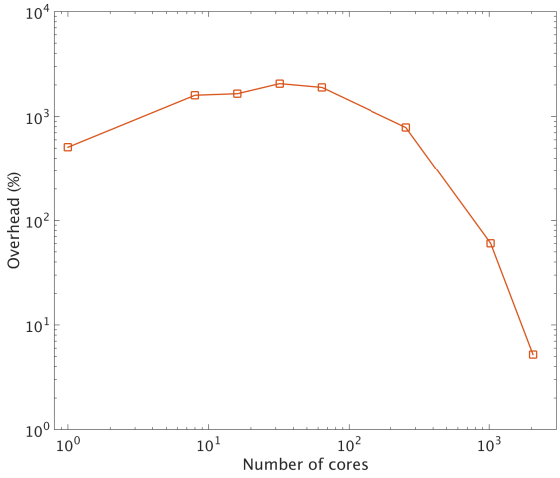

Figure 7: Percent overhead of the interpolation-based robust CG solver, measured at bit-flip rates low enough for the standard solver to converge.

from converging at all.

We have also performed initial tests of our interpolation-based approach in nonlinear variants of HPCCG with forcing terms. Using nonlinear CG or Newton-CG methods, we find behavior similar to the linear CG solvers. The relative overhead does not show significant further decrease from the cost of nonlinear operations because of the much greater number of linear algebra operations performed in the CG method.

5 Conclusion

We have developed a technique to correct silent data corruption when performing linear algebra operations in PDE solvers, based on a model of DRAM bit flips. The technique operates on one or more data vectors, representing spatially discretized fields, and interpolates during linear algebra operations to replace suspected corrupt points. The aim is to maintain accuracy efficiently by correcting accumulated bit flips in data values when they are loaded from memory, just before they are used. This technique can be generalized into a framework for robust linear algebra computations when solving a variety of PDEs.

Test results have shown the potential for improving resilience to silent data corruption in PDE simulations. The technique has been applied to parallel solvers for a hyperbolic 1D nonlinear advection equation (explicit scheme) and a 3D elliptic PDE (conjugate gradient, HPCCG mini-app).

This in-situ approach to SDC tolerance offers the potential to mitigate faults from a variety of unpredictable hardware origins, while minimizing impact on the performance and energy budget. The approach is also generalizable to other linear-algebra-based algorithms used in physics simulations. Further work could

increase the scale of implementations and extend the error model to include other sources of SDC such as processor arithmetic errors.

6 Acknowledgments

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 (NNSA) under contract DE-AC04-94AL85000. This work was funded by NNSA's Advanced Simulation and Computing (ASC) Program.

References

[1] L. Bautista-Gomez and F. Cappello. Detecting silent data corruption for extreme-scale applications through data mining, 2014. `http://dx.doi.org/10.2172/1177404`.

[2] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou. Algorithm-based fault tolerance applied to high performance computing. *J. Parallel Distrib. Comput.*, 69:410–416, 2009.

[3] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomput. Front. Innov.*, 1(1):5–28, 2014.

[4] J. Elliott, M. Hoemmen, and F. Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *Proc. 28th IEEE International Parallel and Distributed Processing Symposium*, 2014.

[5] M. Fasi, J. Langou, Y. Robert, and B. Ucar. A backward/forward recovery approach for the preconditioned conjugate gradient method, 2015. `http://arxiv.org/abs/1511.04478`.

[6] M. Heroux, D. Doerfler, P. Crozier, J. Willenbring, C. Edwards, A. Williams, M. Rajan, E. Keiter, H. Thornquist, and R. Numrich. Improving performance via mini-applications. Report SAND2009-5574, Sandia National Laboratories, 2009.

[7] J. A. F. Hittinger and J. Loffeld. Bend but don't break: Prospects for resilience without recovery in algorithms for hyperbolic systems. Salishan Conference on High-Speed Computing, 2015. `http://www.lanl.gov/conferences/salishan/salishan2015/hittinger.pdf`.

[8] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Software*, 5:308–323, 1979.

[9] J. Ray, J. R. Mayo, and R. C. Armstrong. Finite difference stencils robust to silent data corruption. SIAM Conference on Parallel Processing for Scientific Computing, 2014. `https://www.pathlms.com/siam/courses/477/sections/716`.

[10] P. L. Roe. The use of the Riemann problem in finite difference schemes. In *Proc. Seventh International Conference on Numerical Methods in Fluid Dynamics*, 1980.

[11] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, second edition, 2003.

[12] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *Proc. 26th ACM International Conference on Supercomputing*, 2012.

[13] V. C. Sharma, G. Gopalakrishnan, and G. Bronevetsky. Detecting soft errors in stencil based computations, 2015. `https://e-reports-ext.llnl.gov/pdf/792530.pdf`.

[14] V. Sridharan, N. DeBardeleben, S. Blanchard, K. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. In *Proc. Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.

[15] P. Strazdins, B. Harding, C. Lee, J. R. Mayo, J. Ray, and R. C. Armstrong. A robust technique to make a 2D advection solver tolerant to soft faults. In *Proc. International Conference on Computational Science*, 2016 (to appear).

[16] A. Suresh and J. Sartori. Automated algorithmic error resilience for structured grid problems based on outlier detection. In *Proc. International Symposium on Code Generation and Optimization*, 2014.

[17] P. Widener, K. Ferreira, S. Levy, and N. Fabian. Canaries in a coal mine: Using application-level checkpoints to detect memory failures. In *Euro-Par 2015: Parallel Processing Workshops, Lecture Notes in Computer Science*, volume 9523, pages 669–681, 2015.

[18] C. Yoo, E. Richardson, R. Sankaran, and J. Chen. A DNS study on the stabilization mechanism of a turbulent lifted ethylene jet flame in highly-heated coflow. *Proc. Combust. Inst.*, 33:1619–1627, 2011.