SAND2019-8122C

Physics-Based Checksums for Silent-Error Detection in PDE Solvers

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

Sandia National Laboratories, P.O. Box 969, Livermore, CA 94551, USA
{mnsallo,jmayo,rob}@sandia.gov

Abstract. We discuss techniques for efficient local detection of silent data corruption in parallel scientific computations, leveraging physical quantities such as momentum and energy that may be conserved by discretized PDEs. The conserved quantities are analogous to "algorithm-based fault tolerance" checksums for linear algebra but, due to their physical foundation, are applicable to both linear and nonlinear equations and have efficient local updates based on fluxes between subdomains. These physics-based checksums enable precise intermittent detection of errors and recovery by rollback to a checkpoint, with very low overhead when errors are rare. We present applications to both explicit hyperbolic and iterative elliptic (unstructured finite-element) solvers with injected memory bit flips.

Keywords: Silent errors · partial differential equations · linear algebra · algorithm-based fault tolerance · checkpoint/restart.

1 Introduction

The effects of faults at extreme scale are a growing concern for high-performance computing (HPC) applied to scientific simulation [4]. Much resilience work deals with recovery from hard failures, such as a node that crashes. However, erroneous behavior can manifest in other ways. For example, an error may not immediately cause a crash, but may lead to an insidious wrong answer or cascade to a costly wider failure, which could be avoided if caught earlier. Thus, detecting errors with locality in space and time provides the best opportunity to mitigate them.

In scientific computations, error detection at the application level is facilitated by properties that are common in these simulations and are typically violated when errors occur: smoothness, conservation, and other numerical characteristics. In the face of uncertainty about likely error types and rates at extreme scale, improved algorithmic detection can aid both diagnosis and recovery.

Silent hardware errors, such as silent data corruption, are a prime example where precise detection is important. The future prevalence of these errors is unclear, but there is concern that they will be significant at extreme scale [4]. In addition, improved algorithmic detection could help diagnose and localize subtle software issues such as numerical instability and race conditions [2,9].

Existing work on algorithm-based fault tolerance (ABFT) has developed approaches for application-level error detection. Generic ABFT for linear algebra solvers can be achieved using checksums [13]. In addition, scientific computations often feature physical conserved quantities such as energy or momentum, which can be viewed as a type of checksum, even for nonlinear problems. Such checksums and conserved quantities enable detecting errors reliably. However, in their standard form, they are defined globally, so in a parallel solver they require expensive collective communication [1] and do not localize errors to specific processes or tasks.

Spatially local error detection offers the potential for greater scalability of resilience, reducing communication and allowing more efficient local (rather than global) recovery, just as is sought for other localized failures in parallel programming models [6,14]. Techniques explored for detecting errors locally in scientific computations include machine learning [12], comparison between different numerical methods [2], and outlier detection [11], but these techniques are empirical and inexact, with significant risk of false positives and false negatives.

Here we present a "physics-based checksum" (PBC) approach that builds on ABFT checksums and physical conservation laws applicable to scientific computations, and enables precise and efficient local error detection when such conserved quantities exist. As long as some form of checkpoint/restart remains viable, focusing purely on detection can allow recovering from occasional silent errors by rollback, as for hard failures. This is efficient for rare errors because it avoids the cost of more complex checksums that would support not only detection but also correction (roll-forward).

While the greatest expected benefit of the PBC approach is in conjunction with local recovery (restarting only the processes or tasks with errors) [14], the present work uses global checkpoint/restart (driven by local PBC detection) to illustrate the effectiveness in a familiar resilience setting. We demonstrate the approach in simple MPI-based solvers for partial differential equations (PDEs) and evaluate the effect on solver completion time and accuracy in the presence of emulated silent errors.

An abstract of this work was presented previously [10].

2 Checksum Approaches for Resilience

2.1 Error Detection Concepts

Checksums aim to introduce efficient redundancy in a solver via a smaller "side" computation that remains consistent with the solver state if all computations are correct (Figure 1). State-of-the-art linear algebra checksums (LACs) [13], when verified after a series of linear algebra operations, can indicate with very high probability whether an error (processor or memory error) occurred somewhere in those operations (including in the checksum itself). Even if multiple errors occur, precise cancellation of their effects so that the checksum still matches is very unlikely. Thus, the verification of consistency can be performed intermittently, e.g., just before each checkpoint.

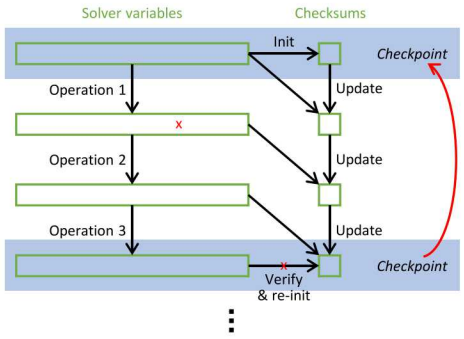

Fig. 1. Data flow in a solver using checksums. An error introduced in an intermediate step (red x) can be detected when the checksums are verified, and the solver can then restart from a valid state.

The checksum for a floating-point vector u is typically taken as the sum of its entries, $Q(u) = e^T u$, where $e = \{1, \ldots, 1\}$. When an operation is performed on u, the linearity of such a checksum allows it to be updated in a way other than directly recomputing it, thus providing the redundant error check. Even with correctly functioning hardware and software, algebraic checksum relations hold numerically only to the level of floating-point roundoff. Silent errors in low-order bits whose numerical magnitude is within the roundoff level will be false negatives (undetected). When a checksum is verified by recomputing it from the underlying data, it is prudent to re-initialize (refresh) the checksum to remove accumulated roundoff drift.

From a different perspective, physical conserved quantities can be used in a similar way. Global conservation laws of the form

$$Q = \int_{\text{space}} dV\, \rho = \text{const}, \tag{1}$$

where ρ is a density expressible in terms of solver variables, are an exact property of many continuum equations, including nonlinear ones. We here consider the preferred case of a "conservative discretization", where a version of the conservation law holds independent of the mesh size or time step and is exact up to roundoff. As with standard LACs, these conserved quantities can detect errors reliably (via comparison of Q at an initial time and a later time) but involve global communication and do not localize errors in space.

To better leverage the benefits of conservation laws and create efficient local PBCs, we consider the more fundamental, local form of a continuum conservation law, $\partial\rho/\partial t = -\boldsymbol{\nabla}\cdot\mathbf{J}$, where $\mathbf{J}$ is the flux density of the conserved quantity. Then, defining the conserved quantity in a spatial region R (e.g., a computational subdomain), $Q(R) = \int_R dV\, \rho$, we find the integrated conservation law

$$\frac{dQ(R)}{dt} = -\oint_{\partial R} d\mathbf{S}\cdot\mathbf{J}. \tag{2}$$

Thus, $Q(R)$ changes only due to the flux through the boundary ∂R. The flux is much faster to compute than $Q(R)$ itself because the integral in (2) is lower-dimensional. When a discretized form of the local conservation law holds, $Q(R)$ is a local PBC that can be updated efficiently and verified intermittently, in contrast to generic LACs [13] that are as costly to update as to verify. While this conservation derivation applies to time-dependent problems, we show in Section 4.1 that PBCs of the same form also apply to iterative elliptic solvers.

2.2 Injecting and Recovering from Errors

To demonstrate the practical effectiveness of PBC error detection, we test parallel solvers in a simple resilience framework with emulated silent errors. As in previous work [11], each solver process includes a concurrent thread that performs asynchronous, uniformly distributed bit flips in the large memory regions in use (floating-point data arrays) at an adjustable rate. Such a memory error model is representative of other error types also [3], such as processor errors.

We use a simple global checkpoint/restart scheme where verification of local checksums and writing of checkpoints occur periodically after a certain number of solver time steps or iterations, termed the verification interval. In our solvers, to establish a baseline given ideal checkpoint reliability and performance, checkpoints are stored in memory and are not subject to error injection, and time spent in checkpointing is not included in our measurements of resilience overhead. Rather, we measure the cost of updating and verifying the checksums and of redoing the computations from the previous checkpoint (global rollback) when an error is detected by any process based on a local checksum discrepancy. Checksum verification occurs together with each checkpoint, so the verification cost has the same effect as checkpointing cost. The cost could be adjusted to reflect any specific checkpoint storage technology. We seek resilience efficiency similar to that seen in standard global checkpoint/restart usage, which can achieve very low overhead using long intervals when failures are rare [5].

The impact of silent errors should be judged in relation to existing numerical inaccuracies (roundoff, discretization, and incomplete convergence) that solvers exhibit even on perfect hardware. An error rate is considered tolerated by a solver, and overhead results are reported, only when the solver reliably finishes with accuracy similar to that of an error-free run. Silent errors are stochastic and vary from run to run, so the results must be considered as a distribution. A solver is deemed to fail in the presence of errors if, in $> 10\%$ of runs, it takes longer than a cutoff time or returns a solution for which the residual or error compared to an analytic solution is more than 3 times that obtained by a run without error injection.

3 Application to 1D Hyperbolic Solvers

We describe the application of PBCs to a linear advection equation and to the nonlinear Burgers equation, and present test results for the latter.

3.1 Algorithm

The 1D linear advection equation is written as

$$\frac{\partial \phi(t,x)}{\partial t} + \nu \frac{\partial \phi(t,x)}{\partial x} = 0, \tag{3}$$

where ν is a constant. The explicit finite-difference Lax-Wendroff scheme for the linear advection equation is determined by the stencil

$$\phi_j^{n+1} = \frac{c(c+1)}{2}\phi_{j-1}^n + (1-c^2)\phi_j^n + \frac{c(c-1)}{2}\phi_{j+1}^n, \quad 0 \le j \le N-1, \tag{4}$$

where the CFL number is $c = \nu\,\Delta t/\Delta x$. This can be thought of as a linear algebra operation, a sparse matrix-vector product $\phi^{n+1} = A\phi^n$, where the tridiagonal matrix A is not explicitly stored.

The vector checksum $Q(\phi) = e^T\phi = \sum_j \phi_j$, where e is a vector of ones, is the discrete version of the quantity $\int dx\,\phi$ conserved by the continuum PDE (3). The checksum computed for each update ϕ^{n+1} should correspond to the matrix-vector product. A general LAC formula for such a checksum update is

$$Q(\phi^{n+1}) = \left(e^T A - de^T\right)\phi^n + d\,Q(\phi^n), \tag{5}$$

where d is an arbitrary scalar constant, whose choice may affect the detectability of propagated errors [13]. In general, this approach incurs the cost of the dot product of $(e^TA - de^T)$ with ϕ^n, the former being a constant precomputed vector.

However, based on our physical reasoning, it must be possible to compute the update more efficiently. The natural PBC is obtained with the choice $d = 1$. For global conservation (e.g., a periodic closed domain), all columns of A have sum 1, as is seen by adding the coefficients of the three terms in (4); so $(e^TA - e^T) = 0$ and the update is trivial: $Q(\phi^{n+1}) = Q(\phi^n)$. For local conservation (e.g., a subdomain within a parallel computation), the column sums of the local matrix A differ from 1 only at the boundaries where fluxes occur, i.e., $(e^TA - e^T)$ is a sparse vector, and the update is much more efficient than a general dot product. For the parallel Lax-Wendroff scheme, the local PBC update is

$$Q(\phi^{n+1}) = Q(\phi^n) + \frac{c(c+1)}{2}(\phi_{-1}^n - \phi_{N-1}^n) + \frac{c(c-1)}{2}(\phi_N^n - \phi_0^n), \tag{6}$$

where ϕ_{-1}^n and ϕ_N^n are values communicated from neighboring subdomains.

PBCs can also be constructed for nonlinear equations where LACs do not apply. The 1D inviscid Burgers equation is written as

$$\frac{\partial u(t,x)}{\partial t} + \nu\, u(t,x)\frac{\partial u(t,x)}{\partial x} = 0. \tag{7}$$

The explicit finite-difference MacCormack scheme for the Burgers equation is determined by the stencil

$$\begin{aligned} u_j^{n+1} &= \frac{1}{2}(u_j^n + u_j^*) - \frac{c}{4}\left((u_j^*)^2 - (u_{j-1}^*)^2\right), \quad 0 \le j \le N, \\ u_j^* &= u_j^n - \frac{c}{2}\left((u_{j+1}^n)^2 - (u_j^n)^2\right). \end{aligned} \tag{8}$$

This stencil cannot be cast purely in terms of linear algebra operations. However, the conservation principle is still valid for the MacCormack scheme, which is conservative by construction. The checksum $Q(u) = e^T u = \sum_j u_j$ corresponds to the momentum $\int dx\, u$ conserved by the Burgers equation, with the continuum flux density $J = \frac{1}{2}\nu u^2$. The corresponding PBC update is

$$Q(u^{n+1}) = Q(u^n) + \frac{c}{4}\left((u^*_{-1})^2 + (u^n_0)^2\right) - \frac{c}{4}\left((u^*_{N-1})^2 + (u^n_N)^2\right). \qquad (9)$$

Here again, the checksum can be updated from the previous time step by only adding contributions from boundary terms.

3.2 Evaluation

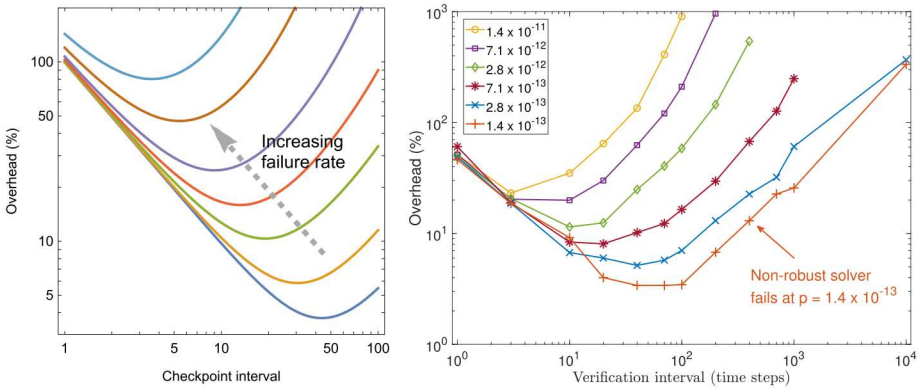

Fig. 2. Left: Example overhead behavior of global checkpoint/restart predicted by an analytic model [5]. Checkpointing and restarting each have a cost of 1 time unit, and the global failure rate per time unit varies from 10^{-3} (bottom curve) to 10^{-1} (top curve). Right: Overhead due to the detection algorithm and additional computations upon restarts when the PBC technique is used in solving the 1D Burgers equation. Results are reported for runs performed on 1024 cores with 100,000 mesh points per core for 25,000 time steps, at several bit-flip rates expressed as probability p per bit per standard time step.

Alongside a typical behavior of global checkpoint/restart for hard failures as a comparison, the overhead results for the Burgers equation are shown in Figure 2. Upon completing a given verification interval (VI), a global restart is performed if any subdomain's recomputed "true" checksum Q_t differs from its efficiently updated checksum Q by more than 10^{-2}. The cost of checksum verification is reduced with a longer VI, leading to the initial decreasing trend of overhead with VI, but as VI increases further, the overhead increases due to more restarts and more wasted work. The optimal VI increases at lower error rates.

Error injection is also performed on the non-robust version of the solver without error detection, to determine the maximum error rate tolerated. As shown, error rates significantly higher than this level can be tolerated by the robust solver with overhead of $\sim 10\%$ or less.

4 Application to 3D Elliptic Solver

To illustrate the applicability of PBCs to iterative unstructured applications, we consider a conjugate gradient solver modeled on the HPCCG and MiniFE mini-apps [7].

4.1 Algorithm

The 3D Laplace equation is a linear elliptic PDE often solved using a finite-element method. The solution is represented as a vector x encoding a superposition of basis functions (elements) defined on a mesh, and the PDE is discretized as a linear system $Ax = b$. Here A is a sparse, symmetric "stiffness matrix" determined by the basis functions, and b is a vector determined by the boundary conditions. In a parallel solver, the mesh is partitioned into subdomains and the corresponding blocks of A, b, and x are distributed among the processes. A typical iterative solver approach is the conjugate gradient method, which repeatedly updates an estimate of the solution x using linear algebra operations until the residual $b - Ax$ becomes sufficiently small. HPCCG implements an unpreconditioned conjugate gradient solver for the Laplace equation using a notional hexahedral mesh.

A key operation in the conjugate gradient solver is a sparse matrix-vector product Ap, where p is a vector generated within the algorithm. As discussed in Section 3.1, the generic LAC update for this operation is

$$Q(Ap) = \left(e^T A - de^T\right) p + d\,Q(p), \tag{10}$$

requiring a dot product that is as costly as recomputing the checksum. Again, a more efficient update is possible with the PBC approach. In our problem, e (a vector of ones) represents a superposition of elements into a constant function, and A represents a differential operator constructed from gradients; thus $e^T A$, corresponding to the derivative of a constant, is a sparse vector (zero except at boundaries). We can take $d = 0$ and obtain the simpler PBC update

$$Q(Ap) = \left(e^T A\right) p. \tag{11}$$

Even though elliptic equations do not involve time advancement and so a conservation law does not literally apply, the solver operations are mathematically analogous to time steps and PBCs can still be used.

To obtain a somewhat more generic example, we replace HPCCG's simple cubic mesh by a cylinder composed of wafers with an unstructured cross-section. Our solver reads in a corresponding stiffness matrix computed offline using basis

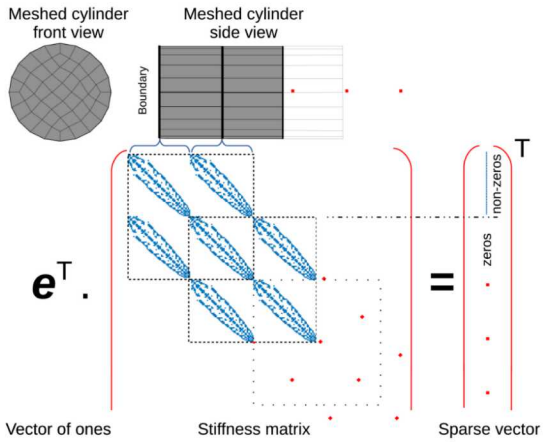

Fig. 3. Top: Schematic 3D unstructured mesh of a cylinder; each wafer (side view not to scale) corresponds to a block in the stiffness matrix. Bottom: In the physics-based approach, the vector of ones (e) is used to form the checksum of solution vectors; the vector on the right is used to update checksums when performing matrix-vector products. The vector on the right is nonzero only on the boundary where fluxes occur, reflecting conservation properties of the Laplace operator.

functions that interpolate between values assigned to each mesh node (trilinear hexahedral elements). Each process operates on a subset of the wafers. The curved surface of the cylinder uses a standard Neumann zero-flux boundary condition, so fluxes in and out of subdomains occur on the boundaries between wafers. The mesh, stiffness matrix, and PBC update are visualized in Figure 3.

In this case, due the uniformity of the cylinder, the above-diagonal blocks are copies of a square matrix B and the below-diagonal blocks are B^T. The nonzero entries in $e^T A$ arise from these B and B^T blocks that couple adjacent subdomains. If p_i denotes the part of the vector p on process i, which can span several wafers, then let $p_{i,l}$ and $p_{i,r}$ be the sub-vectors corresponding to the leftmost and rightmost of these wafers. The local PBC update on process i for the vector $q = Ap$ is then

$$Q_q = (e^T B^T)p_{i-1,r} + (e^T B)p_{i+1,l} - (e^T B^T)p_{i,l} - (e^T B)p_{i,r}. \tag{12}$$

The full conjugate gradient method including local error detection and global checkpoint/restart is shown in Algorithm 1. Steps in blue are PBC updates performed during every iteration, while steps in green are error detection and checkpointing operations performed only after each verification interval. The basis for detection is the relative discrepancy in each local checksum, e.g., $\eta_x = (Q_x - Q_{x,t})/\|x_i\|_1$, upon computing the true checksum $Q_{x,t} = e^T x_i$ on process i.

Algorithm 1 Conjugate gradient method with PBCs. Checksums Q_x, Q_p, Q_q, and Q_r correspond to local portion of vectors on each process i.

$x_0 := 0$ {Initial guess of solution}
$r_0 := b - Ax_0$, $p_0 := r_0$ {Initial residual and direction vectors}
$(R_0)^2 := r_0^T r_0$
for $n = 0, 1, \ldots$ until convergence **do**
  $q_n := Ap_n$
  $Q_q := (e^T B^T)p_{n,i-1,r} + (e^T B)p_{n,i+1,l} - (e^T B^T)p_{n,i,l} - (e^T B)p_{n,i,r}$
  $\alpha := (R_n)^2 / \left(p_n^T q_n\right)$
  $x_{n+1} := x_n + \alpha p_n$
  $Q_x \mathrel{+}= \alpha Q_p$
  $r_{n+1} := r_n - \alpha q_n$
  $Q_r \mathrel{-}= \alpha Q_q$
  $(R_{n+1})^2 := r_{n+1}^T r_{n+1}$
  $\beta := (R_{n+1})^2 / (R_n)^2$
  $p_{n+1} := r_{n+1} + \beta p_n$
  $Q_p := Q_r + \beta Q_p$
  if $\mathrm{mod}(n+1, vi) = 0$ **then**
    $Q_{x,t} := e^T x_{n+1,i}$, $Q_{p,t} := e^T p_{n+1,i}$, $Q_{r,t} := e^T r_{n+1,i}$ {Recompute checksums}
    Compute errors η between recomputed and separately updated checksums
    if $\eta \geq \epsilon$ on any process **then**
      $n \mathrel{-}= vi$ {Restart}
      Read x_{n+1}, p_{n+1}, and r_{n+1} from checkpoint
      Read Q_x, Q_p, and Q_r from checkpoint
    else
      Checkpoint x_{n+1}, p_{n+1}, and r_{n+1}
      Checkpoint Q_x, Q_p, and Q_r
      Refresh checksums
    end if
  end if
end for
Return x_n

We note several details of error detection:

- Verifying the x, p, and r checksums is sufficient because an error in q propagates to an error in r that remains detectable.
- The PBC update (11) does not itself preserve the detectability of an error in p, because Q_p is not used in computing Q_q. However, because a multiple of p is subsequently added to x, the consequence would still be a detectable error in x. Our results support that errors are detected well with $d = 0$.
- The dot products $p^T q$ and $r^T r$ require special consideration because dot products do not have checksums [13]. In our memory error model, this is not a problem because an existing error in p, q, or r that affects a dot product will also affect the subsequent use of the same vectors in a detectable way.
- Error injection is not performed on the stiffness matrix A itself. If corruption of static data like A is a concern, then there are simple protection schemes that can be used [8], but we do not consider this here.

4.2 Evaluation

Error detection thresholds are chosen based on the maximum roundoff-induced checksum discrepancies observed in the solver in the absence of any injected errors. These accumulated errors in the checksum updates increase with VI due to the nonlinear feedback in the conjugate gradient algorithm over iterations and between processes. We have fitted thresholds for our cylinder example as a function of subdomain size and VI.

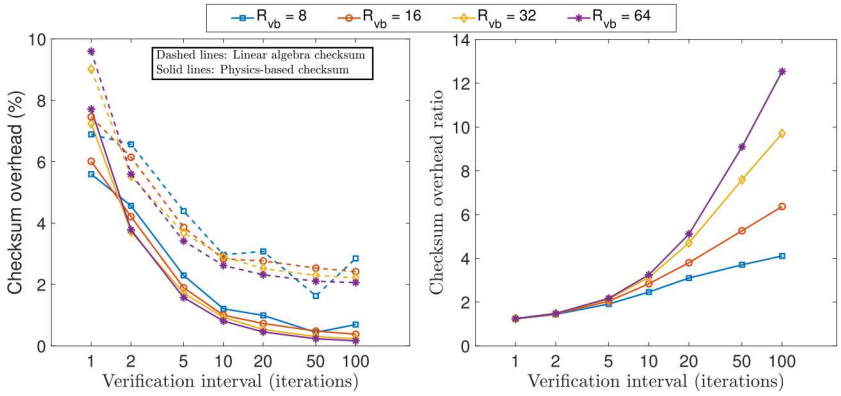

Fig. 4. Error detection overhead is plotted for LACs and PBCs (percent overhead of each technique on left, ratio of LAC to PBC overhead on right) in the conjugate gradient solver on 32 processes with no bit flips injected. Different subdomain sizes are indicated by the volume-to-boundary ratio R_{vb}.

We now examine the overhead induced by our error detection mechanism. With no error injection, we compare the overhead of PBC-based detection to a version where the LAC with $d > 0$ [13], but computed locally, is used for the matrix-vector product. As shown in Figure 4, the PBC approach has significantly lower overhead for larger computational subdomains and larger VI (infrequent verification, expected to be feasible for low error rates). This difference occurs because the LAC update requires a dot product with cost proportional to the subdomain volume at every iteration, whereas the PBC update requires computations only along the subdomain boundaries, which are smaller by a ratio R_{vb}. In the remaining results, we set $R_{\mathrm{vb}} = 8$, corresponding to a subdomain size of 8840 mesh points per process.

The results of overhead measurements with error injection and local PBC detection, shown in Figure 5, are similar to the those for explicit solvers and likewise reflect the similarity to hard-failure checkpoint/restart (left plot in Figure 2). A difference is that the conjugate gradient solver cannot afford as large a VI, because roundoff in the checksum updates propagates more strongly through the algorithm and error detection becomes less precise. Error rates and VIs plotted in Figure 5 are those for which the accuracy criteria in Section 2.2 are met.

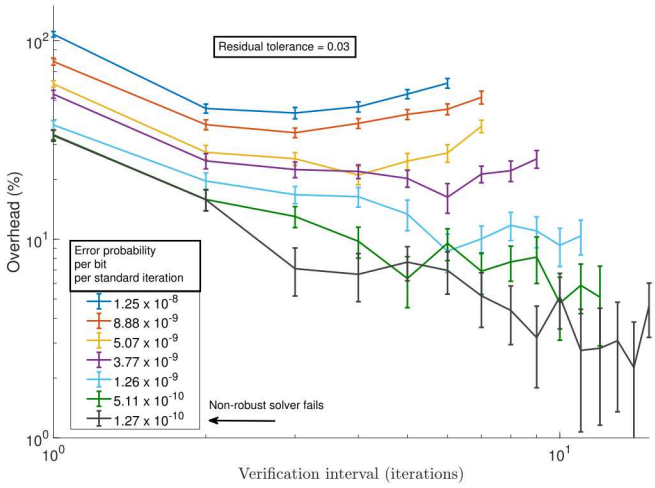

Fig. 5. For the conjugate gradient solver on 32 processes, overhead is plotted versus VI for several rates of memory bit flips. At relatively low error rates, the overhead is $< 10\%$ for suitable VI. At larger error rates, the optimal VI decreases and overhead increases due to greater rollback costs, but the solver can still complete.

5 Conclusion

We have demonstrated a streamlined approach to silent-error detection that shows promise for physics simulations. Physics-based checksums (PBCs) enable precise and efficient local error detection with intermittent verification. In conjunction with recovery by rollback, PBCs fit into a typical checkpoint/restart resilience technique. Moreover, PBCs can apply to a range of solvers and error types that may occur at extreme scale. The approach has generality for scientific computing due to its physical foundation.

While existing ABFT linear algebra checksums correspond to conserved quantities in special cases, the conservation viewpoint leads to a general and efficient method for updating subdomain checksums using boundary fluxes, including for nonlinear equations. The local detection provided by these checksums can be further leveraged with local recovery [14].

Reliable algorithmic error detection provides a risk mitigation for future HPC systems and opens a broader space for co-design in which hardware reliability requirements could be relaxed. The conditions under which resilience techniques are effective can provide useful guidance for these future system designs.

Acknowledgments. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under contract DE-NA0003525. This work was funded by NNSA's Advanced Simu-

lation and Computing (ASC) Program. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

References

1. Bautista-Gomez, L., Benoit, A., Cavelan, A., Raina, S.K., Robert, Y., Sun, H.: Which verification for soft error detection? In: Proc. 22nd IEEE International Conference on High Performance Computing (HiPC) (2015)
2. Benson, A.R., Schmit, S., Schreiber, R.: Silent error detection in numerical time-stepping schemes. Int. J. High Perform. Comput. Appl. **29**(4), 403–421 (2015)
3. Bridges, P.G., Ferreira, K.B., Heroux, M.A., Hoemmen, M.: Fault-tolerant linear solvers via selective reliability (2012), https://arxiv.org/abs/1206.1390
4. Cappello, F., Geist, A., Gropp, W., Kale, S., Kramer, B., Snir, M.: Toward exascale resilience: 2014 update. Supercomput. Front. Innov. **1**(1), 5–28 (2014)
5. Daly, J.: A model for predicting the optimum checkpoint interval for restart dumps. In: Proc. International Conference on Computational Science (2003)
6. Gamell, M., Teranishi, K., Heroux, M.A., Mayo, J., Kolla, H., Chen, J., Parashar, M.: Local recovery and failure masking for stencil-based applications at extreme scales. In: Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (2015)
7. Heroux, M.A., Doerfler, D.W., Crozier, P.S., Willenbring, J.M., Edwards, C., Williams, A., Rajan, M., Keiter, E.R., Thornquist, H.K., Numrich, R.W.: Improving performance via mini-applications. Report SAND2009-5574, Sandia National Laboratories (2009)
8. Hukerikar, S., Engelmann, C.: Resilience design patterns: A structured approach to resilience at extreme scale. Supercomput. Front. Innov. **4**(3), 4–42 (2017)
9. Rinard, M.: Parallel synchronization-free approximate data structure construction. In: Proc. 5th USENIX Workshop on Hot Topics in Parallelism (2013)
10. Salloum, M., Mayo, J., Armstrong, R.: Physics-based checksums for silent-error detection in PDE solvers. SIAM Conference on Computational Science and Engineering (2019)
11. Salloum, M., Mayo, J.R., Armstrong, R.C.: In-situ mitigation of silent data corruption in PDE solvers. In: Proc. 6th Workshop on Fault-Tolerance for HPC at Extreme Scale (2016)
12. Subasi, O., Di, S., Balaprakash, P., Unsal, O., Labarta, J., Cristal, A., Krishnamoorthy, S., Cappello, F.: MACORD: Online adaptive machine learning framework for silent error detection. In: Proc. IEEE International Conference on Cluster Computing (2017)
13. Tao, D., Song, S.L., Krishnamoorthy, S., Wu, P., Liang, X., Zhang, E.Z., Kerbyson, D., Chen, Z.: New-sum: A novel online ABFT scheme for general iterative methods. In: Proc. 25th ACM International Symposium on High-Performance Parallel and Distributed Computing (2016)
14. Teranishi, K., Kolla, H., Slattengren, N., Whitlock, M., Mayo, J., Clay, R.L., Paul, S.R., Hayashi, A., Sarkar, V.: ASC CSSE level 2 milestone #6362: Resilient asynchronous many-task programming model. Report SAND2018-9672, Sandia National Laboratories (2018)