How Applications Use GVR

GVR Group
Department of Computer Science, University of Chicago
1100 E 58th Street, Chicago, IL 60637, United States
gvr@cs.uchicago.edu
http://gvr.cs.uchicago.edu

April 10, 2014

Abstract

As scientific computation moves from petascale to exascale, we must contend with a corresponding increase in hardware faults. In order to retain correct results and acceptable time to completion, some aspects of existing systems or software (or both) need to be hardened against an increasing fault rate. Generic fault tolerance, such as global checkpoint/restart or dual-modular redundancy, can handle faults, but only at a great cost in terms of compute time or extra hardware. More efficient fault tolerance likely requires that developers make each application fault tolerant individually.

We present the Global Resilience View (GVR) framework, which aims to ease the task of augmenting existing applications with fault-tolerance mechanisms that are tailored to the requirements of the application. Then, we discuss our experiences in providing fault-tolerance for a number of existing applications (miniMD, ddcMD, miniFE, Trilinos, Preconditioned Conjugate Gradient, GMRES, and OpenMC) using GVR. We find that GVR is useful for adding resilience to a number of diverse applications in application-specific ways.

1 Introduction

The high-performance computing community is focused on a significant increase in performance from todays petascale systems (10^{15} sustained floating point operations per second) to exascale systems (10^{18} sustained floating point operations per second). That leap poses major challenges in terms of increased parallelism, variability of performance, and our focus here—an increase in system fault and error rates due to hardware scaling to deep sub-micron features and low-voltage for energy efficiency [15, 22, 5, 17, 24, 21]. These rising error rates combined with the large size of future systems (100,000 to 1M nodes) threaten the usability of such exascale systems for large scientific computations (see blue-ribbon panel reports [2, 6, 10]) with projected MTTI ranging from a few minutes to an hour [9, 17, 24, 7] and making traditional, global checkpoint-restart approaches too expensive. Alternatively, the problem could be addressed using redundant or humbled hardware, but we expect that this hardware will also be prohibitively expensive. In short, there is a need for novel approaches to ensure reliable computing for these systems.

An approach that tends to be more successful than traditional, coarse-grained fault tolerance is application-specific fault tolerance. We have seen success with a number of fault-tolerance schemes that are tailored to particular algorithms or applications [13, 3, 23]. If a fault-tolerance scheme is agnostic to the application, it can only implement fault tolerance by employing a large amount of temporal or spacial redundancy (e.g. Checkpoint/Restart or Dual-Modular Redundancy). In contrast, if a fault-tolerance scheme has intimate knowledge of the application that it is protecting, the scheme is granted flexibility in terms of which data needs to be protected, how the data is protected, and how errors are detected. For example, some data may be cheaper to recalculate than they are to protect, or algorithms may be able to tolerate certain errors without requiring any sort of recovery, or applications may present much less inexpensive ways to check for errors than performing calculations twice and comparing the answers.

It follows that, rather than attempting to provide a fault-tolerance solution that automatically provides fault-tolerance to every application, it would be more useful to provide a solution that allows application-developers to easily and flexibly add fault tolerance to each application. In Section 1.1, we present the Global View Resilience (GVR) framework—a library which aims to provide fault tolerance to existing applications. Through the rest of this paper, we discuss our experiences adding fault tolerance to existing applications with GVR. These applications are:

miniMD A simplified version of a molecular dynamics application (Section 2)

ddcMD A domain-decomposed molecular dynamics application (Section 3)

miniFE A simplified finite element solver (Section 4)

PCG/Trilinos An implementation of the Preconditioned Conjugate Gradient linear solver built using the scientific computing library Trilinos (Section 5)

GMRES/Trilinos An implementation of the Generalized Minimal Residual linear solver built using the scientific computing library Trilinos (Section 5)

OpenMC A Monte Carlo reactor simulator (Section 7)

1.1 GVR

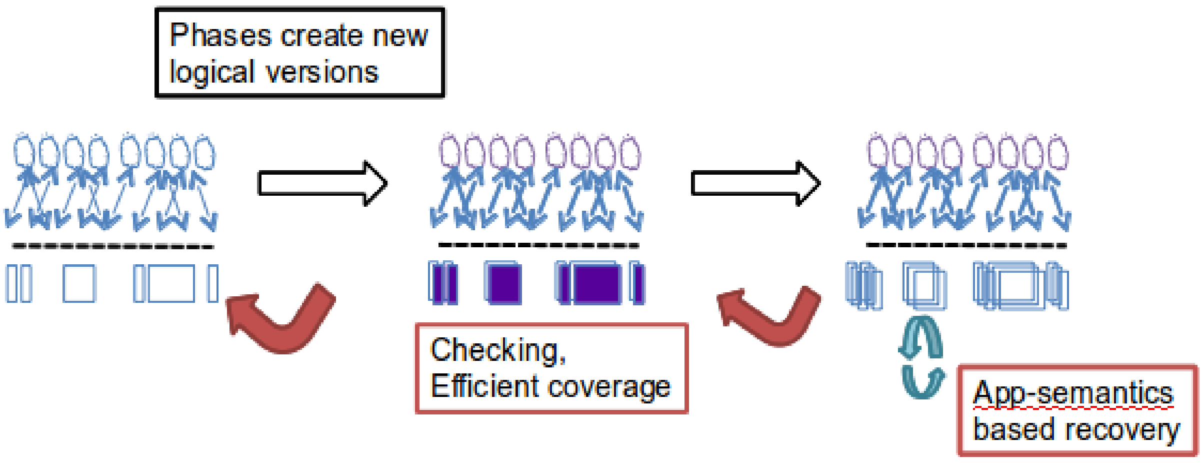
Phases create new logical versions
Checking, Efficient coverage
App-semantics based recovery

Figure 1: GVR utilizes a data-oriented view in which data in a given GDS is in a stable state, then the application operates on data, then the data is in another stable state. In this figure, data is represented by rectangles below the dotted lines, while some application-defined calculation is represented by the arrows and clouds above the lines. In the transition from the state on the left to the state in the middle, the data has been transformed by the application and reaches another stable state. When the application declares this new stable state, GVR may preserve the old version of the data, while checking to make sure that the new version is consistent with the expectations of the application. Transitioning from the middle state to the right state, GVR takes another version of the data. Finally, the application may utilize many preserved versions simultaneously in order to recreate a stable state after an error has occurred.

The Global View Resilience project (GVR) [1, 14, 8, 16, 25] provides a library for parallel scientific applications to perform application-directed fault tolerance. GVR enables the application to create global data store (GDS) objects, which have a number of important properties:

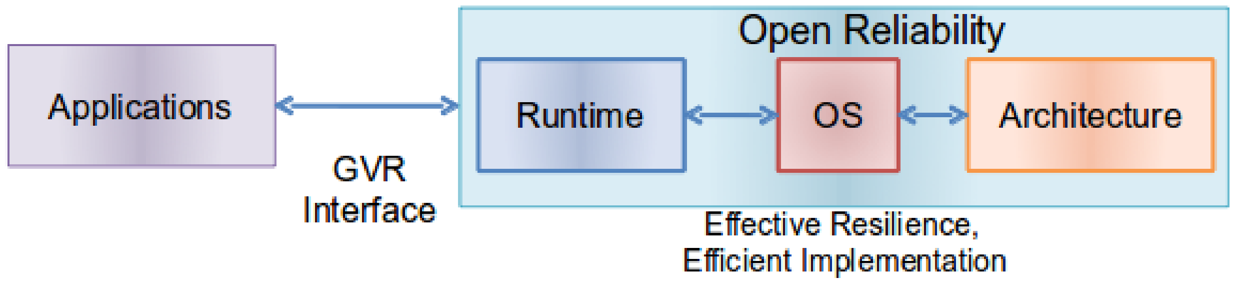

Figure 2: The GVR library provides a unified interface for checking for, signalling, and recovering from errors originating in either the application, or in different parts of the underlying system (runtime, OS, or underlying hardware).

1. A GDS object is has a global name, and is efficiently accessible via one-sided remote memory access (RMA or RDMA).
2. A GDS object can be versioned (user-defined persistent snapshot), and these persistent copies are used in error recovery (See Figure 1). These versions may be taken at application-defined "stable points," which are points in the computation at which the data are considered to be in a consistent state, and, consequently, fit for preservation. The GVR framework may choose either to take or not to take a version at a stable point, depending on hints from the application about the relative importance of performance versus being up to date for the GDS.

   Multiple versions are particularly useful for latent (or silent) error recovery. If an error persists across multiple versions, then the error may be present in at least one of the most recent snapshots. Consequently, it is necessary to recover from an older version in order to restore correctness.
3. Each GDS object has application-specific callback routines for error-checking and error-recovery (see Figure 2). Error-recovery routines can respond to errors raised by either the application or the system.

   When an error is raised, an application-specified error recovery routine can do a number of things. For example, it can recover using a number of old versions of the data in the GDS, it can recovery in some other way, or it can raise another error that will be handled by another recovery routine. Errors can be handled either by processes acting alone, or by processes acting in a coordinated manner.
4. Each GDS object has custom multi-versioning, error-checking, and error-recovery schemes. For example, GVR can take a snapshot of the GDS that preserves the x vector every iteration, while taking a snapshot of the GDS that preserves the p vector on every other iteration. Still other GDS objects may only utilize one persistent snapshot, and may utilize error checks and recovery methods that are not discussed in this paper, like using parity to verify correctness.

The applications described in this paper make use of GVR's capabilities in various ways, which will be elucidated below.

2 miniMD

In order to understand the characteristics of large, complicated science and engineering applications and further improve the performance, Sandia National Laboratory has developed a number of mini-applications which pare down larger applications and focus on the features that most impact performance impact application performance [11].

MiniMD is one of these mini-applications. It is intended as a proxy for the larger molecular dynamics (MD) application LAMMPS . In particular, it focuses on the force computations that are common in typical

MD applications. Like LAMMPS, MiniMD uses spatial decomposition MD, where individual processors in a cluster own subsets of simulation box and simulate the behavior of atoms. Problem size, atom density, temperature, timestep size, particle interaction cutoff distance and other parameters are specified by the user.

2.1 How We Apply GVR

A MiniMD computation proceeds in a series of timesteps. Each timestep of MiniMD consists of the following steps:

1. update velocity using force
2. update position using velocity
3. build neighbor lists
4. compute force using position
5. apply constraints & boundary conditions on force
6. update velocity using new force
7. output

Three essential parameters for the computations are position, velocity, and force of atoms. Therefore, to make MiniMD fault tolerant, we use a GDS object to protect these three variables. Upon an error being detected, MiniMD reads back the stored values of position, velocity and force and restarts the computation.

2.2 Results

We performed the error-injection experiments to test the recovery capabilities of MiniMD with GVR. Errors were randomly injected into position, velocity and force in the runtime. We added application-level error detection codes into MiniMD. When errors were detected, MiniMD could restore the computation and generate the correct results.

3 ddcMD

Domain-decomposition molecular dynamics (ddcMD) is a collection of atomistic simulation programs developed by Lawrence Livermore National Laboratory. It is designed to achieve scalability and efficiency. The existing implementation of ddcMD can tolerate L1 cache parity error on BG/L by utilizing a simply rally checkpoint/restart scheme. To used GVR to make ddcMD tolerant to more types of errors.

3.1 How We Apply GVR

ddcMD uses a more complicated model of physics than MiniMD. We first analyzed the data structures in ddcMD and identified a set of variables that are essential for computation and recovery. We then preserved these variables in GDS objects. We designed two error detection methods.

Furthermore, we conducted error-injection experiments in two steps. First, we injected errors into different variables. Second, we varied the granularities of errors injected. For these different sorts of errors, we tested the correctness of recovery and compared the sensitivities of our error-detection methods.

3.2 Results

We demonstrated that GVR-augmented ddcMD was tolerant of detected errors. The error injection experiment shown in Figure 3 traces the total energy of atoms during the computation. Normally, the total energy changes smoothly in a small granularity. After timestep 2000, errors are injected, causing a remarkable change. At timestep 2010, the error detection is invoked and detects the error. Then the application rolls back to timestep 2000 and restart the computation correctly.

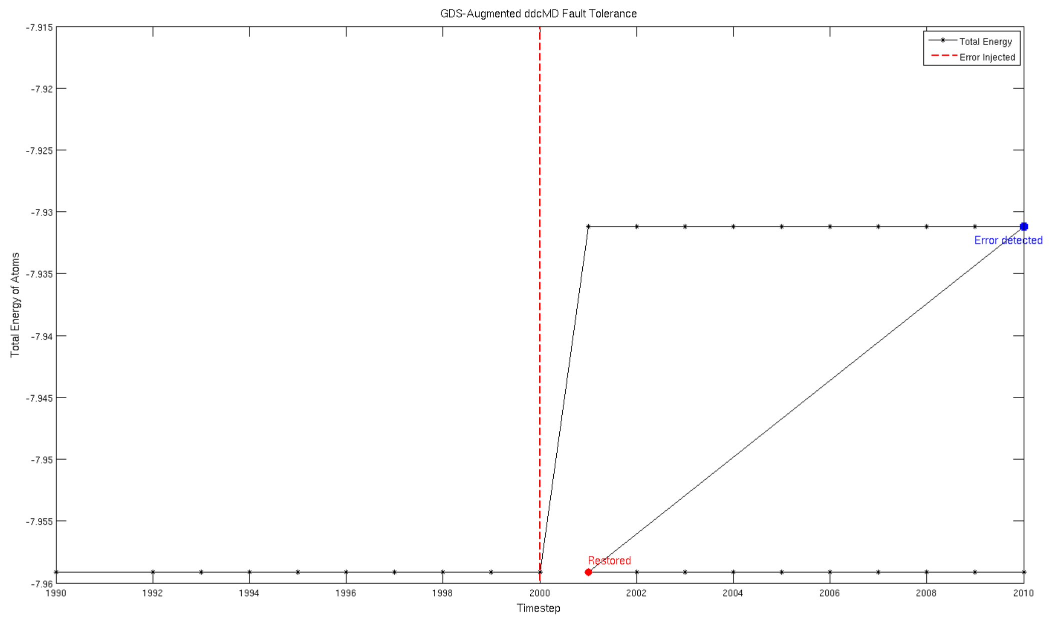

Figure 3: The total energy trend in ddcMD given error injection and recovery

Furthermore, our error-injection experiments concluded that

1. augmented with GVR, ddcMD can correctly recover from errors.
2. with GVR, ddcMD can tolerate errors beyond L1 cache errors.
3. different data structures show different sensitivities towards errors.
4. sophisticated error detection methods are important for fault tolerance.

4 miniFE

MiniFE is one of the suite of Montevo miniapps [12]. MiniFE is a proxy for a class of applications that require an implicit solution to a set of nonlinear equations. In particular, miniFE simulates solving an unstructured grid problem with finite element method. A large proportion of computational time is spent inside a linear solver kernel– in particular, Preconditioned Conjugate Gradient method (PCG).

4.1 How We Apply GVR

Finite element solvers have two primary phases of computation. The first phase generates a system of linear equations to solve based on the decomposition of the domain and the problem to be solved. The second phase solves the system of linear equations. Making the first phase fault-tolerant is a different problem than making the second phase fault-tolerant. We focused on making the second phase fault-tolerant [19] and left the first for future work.

For miniFE, we used GVR to preserve critical elements of the state of computation and then restore them in the event of drastic increase in the distance between the approximate answer and the correct answer. We expanded our work with PCG in a further study, described in Section 5.

4.2 Results

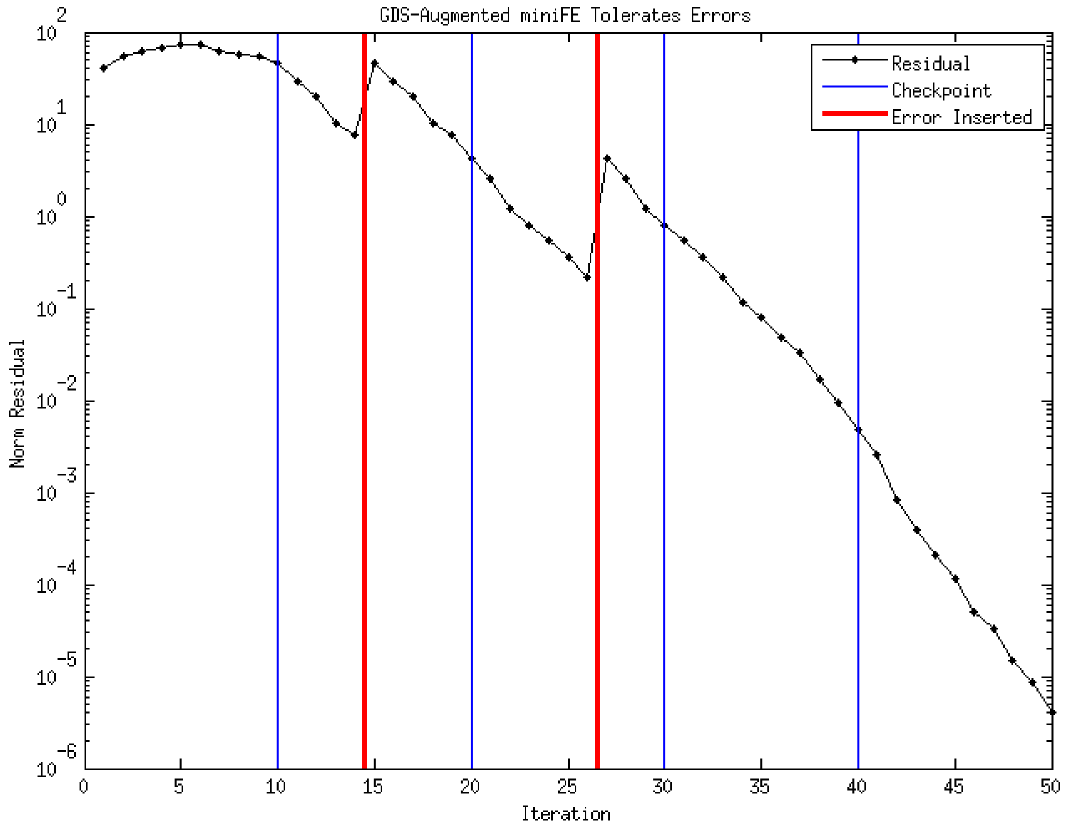

Figure 4: The norm residual trace for the PCG portion of miniFE given error injection and recovery. After an error is injected, rather than growing unbounded, the norm residual returns to its value at the last checkpoint when GVR restores the state of the computation

We performed an exploratory error-injection experiment in which we severely corrupted every element of the critical r vector in PCG. We then used GVR to periodically take snapshots of critical variables, and then, in the event of a drastic norm residual increase, restore the solver to its previous state. In figure 4 we see the trace of the norm residual during the PCG phase of miniFE. After an error is injected, norm residual returns to its value at the last snapshot rather than drastically growing converging at a slower rate.

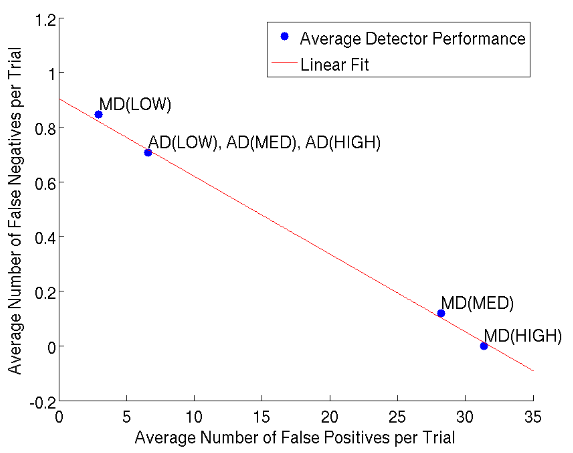

Figure 5: In the case of residual-based methods, any gain in recall comes with a high cost in precision. Each point represents the performance of a detection scheme. The cluster of points that nearly overlap are the various AD schemes. Shown with a linear fit (R^2=0.9977).

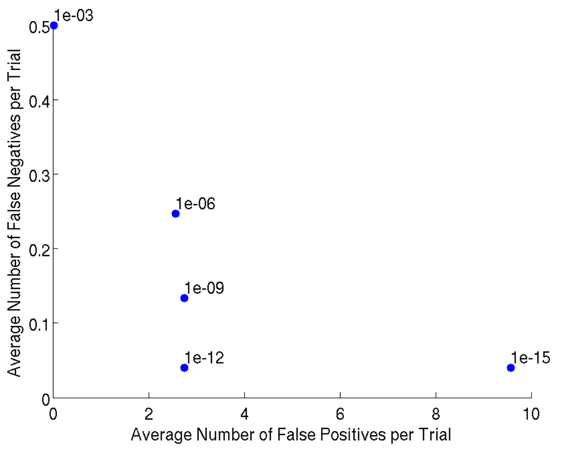

Figure 6: Until machine precision is reached, algorithm-based detection methods can decrease false negatives by reducing tolerance without severely increasing the number of false positives. The performance of all detectors is shown with the SR(1) correction scheme in order to mitigate the impact of uncorrected errors.

5 PCG/Trilinos

The Trilinos project [11] is a C++ library that provides scalable primitives for linear algebra operations, linear and nonlinear solvers, and other useful scientific computing algorithms. In this study, we utilized Trilinos' linear algebra primitives in order to implement a PCG solver and expand on the work discussed in 4.

Preconditioned conjugate gradient is a common way to iteratively solve the linear system $Ax = b$. In addition, it is the simplest of the class of Krylov subspace solvers which solve linear systems by moving the approximate answer in one dimension of Krylov subspace at a time. It is not clear how errors in PCG should be efficiently detected and corrected. For example, norm residual does not decrease monotonically as computation proceeds, so, barring errors that cause extreme divergence in state as in Section 4, it is difficult to detect errors by monitoring the norm residual. When an error is detected, there are a number of conceivable ways to recover, including restoring old state; ignoring errors and depending on numerical resilience; and replacing corrupted data with some approximation of the correct values.

5.1 How We Apply GVR

We can take advantage of the abstraction that Trilinos offers by building GVR-provided resilience into linear algebra primitives rather than requiring the application developer to interact with GVR directly. We decorated Trilinos vector objects with methods to snapshot and restore state on demand with GVR [19]. These methods were then used in conjunction with application-directed error detection in order to find errors and restore to a previous application state as appropriate.

5.2 Results

We found that detection methods make a good deal of difference when correcting errors in PCG. We experimented with inexpensive methods based on monitoring the norm residual and more expensive, algorithm-aware methods that performed extra linear algebra operations to verify PCG-specific invariants.

We found that: 1) Though inexpensive, residual-based detection performs poorly. To achieve acceptably low false negative rates, high (30x number of mitigated false negatives) false positives rates are required 5. 2) Though more expensive, algorithm-based detection performs better overall, achieving much lower false negative rates at one seventh the false positive rate 6. Even this relatively expensive error detection is inexpensive compared to a single solver iteration, and therefore is viable for linear solvers—particularly in high error-rate systems.

6 GMRES/Trilinos

Like PCG, Generalized Minimal Residual Method (GMRES) is a Krylov subspace method for solving systems of linear equations. A variation of GMRES that is particularly interesting to the realm of fault tolerance is Flexible GMRES [20] (FGMRES) or the similar Fault-Tolerant GMRES [13] (FTGMRES). In this variation, each iteration of GMRES utilizes an inner solver to solve a linear system that is simpler than the system that FGMRES is ultimately trying to solve. In principal, FGMRES will eventually return correct results regardless of the results of the inner solve. In addition, about 90% of execution time is spent in the inner solver [25]. Consequently, we can afford to employ light-weight fault-tolerance methods on the inner solver and employ more heavy-weight fault-tolerance methods on the outer solver, and still converge to correct results with good performance.

As in the PCG project, this work utilized Trilinos library. The work employed both Trilinos' implementation of linear algebra primitives and Trilinos' GMRES implementation.

6.1 How We Apply GVR

We used GVR to preserve critical data structures in FGMRES and restore them in the event in the event that an error was detected [25]. One scheme used GVR to preserve multiple versions of critical objects during the inner solve so that, if an error was detected after the completion of an inner solve, the inner solver could be resumed from the version before the error occurred rather than having to restart the entire inner solve.

6.2 Results

We found that, even though FGMRES is inherently resilient to inner solver errors, performance in a faulty environment could be significantly improved by utilizing GVR for fault tolerance (Figure 7). In addition, utilizing even very expensive Dual-modular redundancy in the outer solver significantly improved performance over restarting.

7 OpenMC

OpenMC is a production code for conducting direct full-core reactor simulation by using Monte Carlo methods [18]. OpenMC is capable of simulating 3D models based on constructive solid geometry with second-order surfaces. It was originally developed by members of the Computational Reactor Physics Group at the MIT in 2011. The application is written in FORTRAN, with support for a hybrid MPI/OpenMP parallelism. OpenMC is an open source software project available online, with contribution from various universities, laboratories, and other organizations.

During a simulation, there are three categories of data need to be stored in memory:

- Geometry — Geometry in Monte Carlo simulation is non-mesh based, read-only data and can be represented using *constructive solid geometry.*

- Interaction cross sections — The random events of each particle are determined by experimentally pre-measured probability distributions, i.e., cross sections. The cross section is also read-only data and accessed randomly by each process during the simulation. The size of cross section storage depends

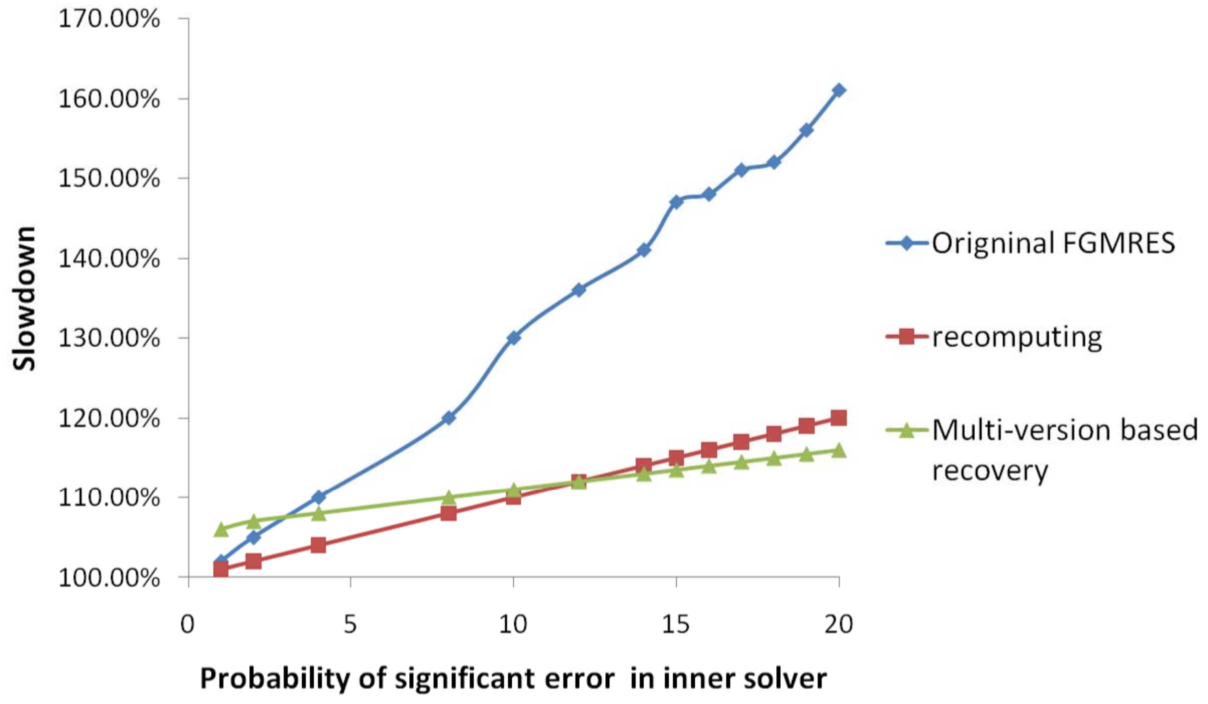

Figure 7: With FTGMRES, we used two GVR to implement two different recovery schemes. The "recomputing" scheme (red) will repeat the entire inner solver step in the event of an error. The "Multi-version based recovery" scheme (green) uses GVR to preserve multiple versions of critical data structures throughout the computation and continues the inner solver from the last good version rather than restarting the entire computation. Multi-version performs better than recompute until the error rate grows to a particular crossover point.

on the energy and temperature, as well as the number of nuclides present in the system. Therefore, the actual cross section size may vary significantly and depends on the specifics of the application. In an application with considerable temperature intervals and energy points, the cross section data can exceed 100 GB.

- Tallies — Tally data is region-based and accumulated (i.e., fetch-and-add) data, where the region, or tally region, is the volume over which the tallies should be integrated. The size of total tally data is directly proportional to the number of physical quantities to be tallied and the number of tally regions. In a realistic reactor simulation, that tally could reach terabytes size of data. Unlike geometry and cross sections, tally is only output data and not required for particles simulation; it is possible to process tally data in an asynchronous way.

7.1 How We Apply GVR

We make OpenMC resilient by applying data versioning to its two major data structures: cross sections and tally data. The reliability of OpenMC data structures are categorized as shown in 1.

Table 1: Data Reliability in OpenMC

Data Structure	**Property**	**Management**	**Recovery**
Geometry	Read-Only	Caching	Reread from non-volatile storage
Cross section	Read-Only	Caching	Recompute from cached good data
Tally data	Accumulate	Versioning	Remove bad contribution and compensate by recomputing

Cross section and geometry data are stored on low-level, read-only , and is replicated over nodes. Therefore, they can be simply recovered by rereading from storage or fetching from other close nodes that have the replication. It is also possible to recompute cross section data from cached good data.

Data versioning is applied to tally data. At the end of one batch simulation (batch i), tally data is snapshotted as a version T_i. Thus, we have a history of tally data $T_1 \ldots T_n$. Since the tally scoring is Monte Carlo accumulation, if one latent error is detected at batch n, then we are able to recover the T_n with error to correct $T_n^{'}$ by

$$T_n^{'} = T_n - (T_i - T_{i-1}) + Recompute(batch_i) \tag{1}$$

This approach is different from checkpointing/restart because we preserve the computation effort from batch $i+1$ to batch n, while checkpoint/restart needs to roll back and start over from batch $i+1$.

7.2 Results

We developed a prototype and illustrated that using versioned tally data can successfully recovery tally data from latent errors. The recovery overhead, versioning overhead and optimal versioning frequency is a work in progress. We found that GVR's versioning feature is an effective recovery mechanism for Monte Carlo computation.

In addition, by applying GVR to OpenMC, one can also straightforwardly decompose large tally data to address the on-node memory limit problem in full-core reactor analysis. The performance evaluation shows that using GVR with RMA can achieve 70% of efficiency by full replication of tally approach (ideal performance with data decomposition) and scales well up to 256 processes.

8 Conclusions and Future Work

In order to efficiently address the problem of increasing fault rate at exascale, it is probably necessary to write application-specific fault tolerance We present GVR, a framework that allows fault tolerance to be flexibly added to existing applications. We demonstrate the utilization of GVR in a number of applications.

In the future, we plan to utilize GVR to add resilience to more types of applications. For example, we are currently engaged in using GVR to make the the Chombo adaptive multigrid package [4] resilient to single-process failures. We also plan to exploit more of the capabilities of GVR. In particular, work with cross-layer integration is still at a very early stage, but presents interesting possibilities.

References

[1] Global view resilience (gvr). `https://sites.google.com/site/uchicagolssg/lssg/research/gvr`. Accessed: 2013-12-07.

[2] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. Toward exascale resilience. *International Journal of High Performance Computing Applications*, 23(4):374–388, 2009.

[3] Zizhong Chen. Online-abft: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '13, pages 167–176, New York, NY, USA, 2013. ACM.

[4] P Colella, DT Graves, TJ Ligocki, DF Martin, D Modiano, DB Serafini, and B Van Straalen. Chombo software package for amr applications-design document, 2000.

[5] Cristian Constantinescu. Trends and challenges in vlsi circuit reliability. *Micro, IEEE*, 23(4):14–19, 2003.

[6] Elmootazbellah Nabil Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, 34(3):375–408, 2002.

[7] M. Elnozahy. System resilience at extreme scale: A white paper, 2009. DARPA Resilience Report for ITO, William Harrod.

[8] Hajime Fujita, Rob Schreiber, and Andrew A. Chien. It's time for new programming models for unreliable hardware. In *ASPLOS 2013 Provocative Ideas session*, March 2013.

[9] Al Geist. A paradigm shift is coming-continuous failure. In *Collaboration Technologies and Systems (CTS), 2012 International Conference on*, pages 371–371. IEEE, 2012.

[10] Robert Geist and Kishor S. Trivedi. Reliability estimation of fault-tolerant systems: Tools and techniques. *Computer*, 23(7):52–61, 1990.

[11] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, et al. An overview of the trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423, 2005.

[12] Michael A Heroux, Douglas W Doerfler, Paul S Crozier, James M Willenbring, H Carter Edwards, Alan Williams, Mahesh Rajan, Eric R Keiter, Heidi K Thornquist, and Robert W Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep. SAND2009-5574*, 2009.

[13] Mark Hoemmen and M Heroux. Fault-tolerant iterative methods via selective reliability. In *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC). IEEE Computer Society*, 2011.

[14] Sean Hogan, Jeff Hammond, and Andrew A. Chien. An evaluation of difference and threshold techniques for efficient checkpointing. In *2nd Workshop on Fault-Tolerance at Extreme Scale FTXS 2012 at DSN 2012*, June 2012.

[15] Andy A Hwang, Ioan A Stefanovici, and Bianca Schroeder. Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design. *ACM SIGARCH Computer Architecture News*, 40(1):111–122, 2012.

[16] Guoming Lu, Ziming Zheng, and Andrew A Chien. When is multi-version checkpointing needed? In *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale*, pages 49–56. ACM, 2013.

[17] Peter Kogge, et. al. Exascale computing study: Technology challenges in achieving an exascale systems, 2008. DARPA IPTO Study Report for William Harrod, available from `http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf`.

[18] Paul K. Romano and Benoit Forget. The OpenMC Monte Carlo particle transport code. *Annals of Nuclear Energy*, 51:274–281, 2013.

[19] Zachary Rubenstein, Hajime Fujita, Ziming Zheng, and Andrew Chien. Error checking and snapshot-based recovery in a preconditioned conjugate gradient solver. Technical Report TR-2013-11, Department of Computer Science, University of Chicago, November 2013.

[20] Youcef Saad. A flexible inner-outer preconditioned gmres algorithm. *SIAM Journal on Scientific Computing*, 14(2):461–469, 1993.

[21] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *High Performance Computing for Computational Science–VECPAR 2010*, pages 1–25. Springer, 2011.

[22] Vilas Sridharan, Jon Stearley, Nathan DeBardeleben, Sean Blanchard, and Sudhanva Gurumurthi. Feng shui of supercomputer memory: positional effects in dram and sram faults. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, page 22. ACM, 2013.

[23] Hubertus JJ van Dam, Abhinav Vishnu, and Wibe A de Jong. A case for soft error detection and correction in computational chemistry. *Journal of Chemical Theory and Computation*, 9(9):3995–4005, 2013.

[24] et. al. Vivek Sarkar. Exascale software study: Software challenges in extreme scale systems, 2009. DARPA IPTO Study Report for William Harrod, available from `http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf`.

[25] Ziming Zheng, Andrew A. Chien, and Keita Teranishi. Fault tolerance in an inner-outer solver: A gvr-enabled case study. In *11th International Meeting High Performance Computing for Computational Science-VECPAR 2014*, 2014.