Fault Tolerance in an Inner-outer Solver: A GVR-enabled Case Study

Ziming Zheng[1], Andrew A. Chien[1], Keita Teranishi[2]

[1] University of Chicago, Chicago IL 60637, USA
[2] Sandia National Laboratories, Livermore, CA 94551, USA

Abstract. Resilience is a major challenge for large-scale systems. It is particularly important for iterative linear solvers, since they take much of the time of many scientific applications. We show that single bit flip errors in the Flexible GMRES iterative linear solver can lead to high computational overhead or even failure to converge to the right answer. Informed by these results, we design and evaluate several strategies for fault tolerance in both inner and outer solvers appropriate across a range of error rates. We implement them, extending Trilinos' solver library with the Global View Resilience (GVR) programming model, which provides multi-stream snapshots, multi-version data structures with portable and rich error checking/recovery. Experimental results validate correct execution with low performance overhead under varied error conditions.

1 Introduction

The scaling of semiconductor technology and increasing power concerns combined with system scale make fault management a growing concern in high performance computing systems [1, 4, 11, 13]. Soft errors and higher error rates all expected. Just as they played an important role in achieving scalable, high performance, we expect that widely-used numeric solvers such as Flexible Generalized Minimal Residual Method (FGMRES) will play an important key role in achieving resilience and performance for large-scale applications in future "exa" scale systems.

Flexible GMRES with restarting (see Fig. 1 [2, 17]) is robust to soft errors due to three aspects. First, the inner solver in Step 3 is inexact, and the outer solver can tolerate large changes to inner solver. Second, the minimal residual procedure can reduce the impact of error on inner solver and keep the residual decreasing (see Step 11). Third, FGMRES restarts the computation after m outer iterations (see Step 17). While the major purpose of restarting is to address the performance and memory usage, restarting can also eliminate errors in outer solver data structures. However in our experiments some bit-errors are still problematic. Errors in inner solver can incur high computational overhead for convergence. Errors in outer solver can even lead to divergence failure. Restarting may lead to stagnation of convergence.

With these insights, we design and evaluate error checking and recovery strategies. For inner solver, residual based checking is deployed to identify significant error; recomputing and multi-versioning are exploited for recovery in different cost and granularity. For outer solver, double modular redundancy and data reloading strategies are utilized for error checking and recovery. Our experiments employ the Trilinos library [12], extending FGMRES inner-outer solver with the Global View Resilience (GVR) framework [10], use 5 matrices from the Florida sparse collection [7], running on up to

128 processes. Experimental results illustrate that our GVR-enabled FGMRES solver successfully tolerates the bit flip errors and significantly reduces the impact on performance. Specific contributions include:

- Characterizing situations where bit-errors cause resilience problems for both inner and outer solvers in FGMRES.
- Employ GVR programming model with Trilinos library for portable and rich error checking/recovery strategies in inner-outer solver.
- Evaluate each recovery method, empirically validating that they are efficient and that each is best for regime of error rates.

The rest of the paper is organized as follows. Section 2 introduces the background of GVR and Trilinos for our implementation. Section 3 and section 4 explore the error impact error checking and recovery methods for inner solver and outer solver respectively. Section 5 discusses experimental results, and Section 6 surveys related work. Finally, we summarize and discuss future directions in Section 7.

Input: Linear system $Ax = b$ and initial guess x_0.
Output: Approximate solution x_m.

1: $r_0 := Ax - b, \beta := ||r_0||_2, q_1 := r_0/\beta$
2: **for** $j = 1, \ldots, m$ **do**
3: Inner solver for inexact solution z_j in $q_j = Az_j$
4: $v_{j+1} := Az_j$
5: **for** $i = 1, \ldots, j$ **do**
6: $H(i,j) := (v_{j+1}, q_i)$
7: $v_{j+1} := v_{j+1} - q_i H(i,j)$
8: **end for**
9: $H(j+1,j) := ||v_{j+1}||_2$
10: $q_{j+1} := v_{j+1}/H(j+1,j)$
11: $y_j := argmin_y||H(1:j+1,1:j)y - \beta e_1|||_2$
12: $x_j := x_0 + [z_1, \ldots, z_j]y_j$
13: **end for**
14: **if** converged **then**
15: Return x_m
16: **else**
17: $x_0 := x_m$, *go to* 1
18: **end if**

Fig. 1: Flexible GMRES with Restarting

2 GVR and Trilinos

Our implementation of fault tolerance inner-outer solver is based Global View Resilience (GVR) [10] and Trilinos[12]. Trilinos is an object-oriented software framework for solving big complex science and engineering problems. Kernel classes of Trilinos include vector, matrix, and map. It provides common abstract solvers, such as iterative linear solvers and preconditioners. Based on the kernel class and solvers, Trilinos provides comprehensive algorithmic packages such as stochastic PDEs.

GVR is a novel programming model to enable sophisticated, application-specific fault tolerance in parallel computing. It enables the application to create global data store (GDS) objects for flexible, portable and efficient fault management. We extend the kernel classes of Trilinos using GVR APIs, including the GDS object creation, put/get operations, and GDS versioning. Based on the extended kernel classes, we implement GVR enabled inner-outer solver package, which can be directly used for other Trilinos applications. Especially, GVR facilitates our inner-outer solver in the following aspects.

1. GVR provides multi-stream scheme to create multiple GDS objects for distributed basis vectors and solution vectors. Each GDS object can periodically take snapshots at application specified stable point such as the end of iteration. GVR explores the benefits of local and hierarchy storage to reduce the runtime overhead of snapshot.
2. Multiple older versions of the GDS object remain available for access. The multi-version scheme is motivated for latent error, i.e., errors that retain for some iterations. We use it for recovery inside of the inner solver.
3. It is flexible to configure different versioning, error-checking, and error-recovery schemes to each GDS object. It is helpful to customize the explored strategies thus adapting to different error rates.
4. GVR provides erasure code based on resilience mechanisms for the multi-version snapshots. Since the snapshot is used only for recovery, the overhead is negligible. It is also configurable to explore NVRAM with low error rate for snapshot resilience.
5. The application can provide each GDS object with specific callback routines for error checking and error-recovery in a uniform framework. Error-recovery routines can respond to errors raised by either the application or by the underlying system, such as uncorrectable ECC signal from operating system. Combining with multi-version, GVR can recover the application from catastrophic memory failures.

In this paper, we only use 1-4 GVR features to address soft errors. We will explore using more features in the future.

3 Inner Solver

In this study, we presume that the inner solves takes most of execution time and arbitrarily set 30 iterations inner solver. In this scenario, the inner solver takes more than 90% execution time, which is a key factor to make trade-off between system reliability and inner solve reliability. We will study other scenarios as a future work.

3.1 Error Impact

To study the impact of errors on inner solver, we randomly inject the error during SpMV or vector dot product operation as the most error-prone, or inject the result vector z_j directly as the most important data visible to the outer solver. In this study, we focus on double precision floating-point data, which consists of 1 sign bits, 11 exponent bits, and 52 bits for mantissa. Bit-flips not in the first 2 bytes only introduce a relative error $<= 2^{-4}$ [9], thus having little impact on execution correctness and convergence.

As inner solver result is approximate, if error occurs not in the first 2 bytes which only introduces a relative error $<= 2^{-4}$ [9], error impact is minimal on execution correctness and convergence. However, as shown in Fig. 2, if a bit error significantly increase the residual of a significant inner solver comparing with previous inner-outer

iteration, it generally incurred 2 or 3 additional inner-outer solver iterations, which is consistent with the study in [9]. In extreme cases, as many as 48 additional inner-outer solver iterations can be required. Further, the error impact can accumulate. As the increasing of errors, we observe $8\times$ number of inner-outer iterations in extreme cases.

3.2 Error check and recovery: outside

First, we study outside error checking and recovery; such coarse-grained recovery is relevant even in current-day error environments, and applies to many inner solvers such as GMRES and CG. We exploit two symptoms to identify significant error: 1) residual increase (vs previous iteration) and 2) the matrix $H(1:j, 1:j)$ is not full rank [2]. For these methods, checking overhead is low. Explicit residual checks can be calculated by outer solver, as well as checks for errors in A and q_j. In our experiments, the explicit residual check incurs only take 0.2% overhead per iteration. Further, checking rank deficiency of matrix $H(1:j, 1:j)$ is essentially free as the SVD-based method to calculate step-11 (see Fig. 1) computes the its rank directly.

There are two simple strategies for recovery outside of inner solver. The first is recomputing the inner solver, incurring high overhead since the inner solver is 90% of the computation. Despite that, recomputing is still viable as the significant inner solver errors generally introduce 2-3 more inner-outer solver iterations (see Fig. 2. The second is restarting the whole computation as step-17 in Fig. 1 [2]. Restarting may lead to stagnation of convergence, so it is employed only if recomputing fails.

3.3 Error recovery: inside

For higher error rates, it is necessary to handle the errors inside of the inner solver rather than recomputing the whole inner solver. In this study, we keep one snapshot of q_j at the beginning and multi-version snapshot of z_j during the inner solver iterations. If a significant error is detected outside of inner solver, we check the versions in descend order. If any version of intermediate result has significant lower residual than the final result, inner solver rolls back to that point, reload q_j and A, and executes the rest iterations. Otherwise, inner solver is recomputed from the scratch.

An alternative solution is to check and handle the error during the inner solver iterations. In this study, we do not adopt it due to two reasons. First, it is difficult to identify the error with low overhead and high coverage. Second, it is hard to predict the impact of the error on the inner-outer iterations. We will study this solution as a future work.

The crossover between outside and inside error handling happens when the overhead of error recovery within the solver is less than later recomputing. In this study, we define the *error probability* as the ratio between the number of iterations with errors and the total iterations. Suppose the probability of inner solver error is P, the error-free execution is Φ longer due to the overhead of snapshot, and the error handling inside reduce the recomputing time by Θ shorter. So the error handling inside inner solver become beneficial when $1 - P + 2P > (1 - P)\Phi + (1 + \Theta)P$. We validate these tradeoffs in our experiments in Section 5.

4 Outer solver

The outer solver typically consumes less execution time, but errors in outer solver are more critical for correctness and performance. In most cases, if significant errors occur

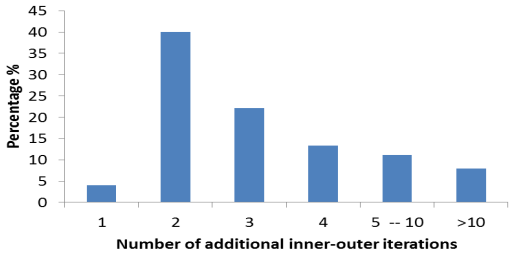

Fig. 2: Distribution of additional inner-outer solver iterations incurred by significant inner solver errors.

in the basis vectors or Hessenberg matrix H, the residual may increase or stay constant. Even a single bit-flip may lead to divergence no matter at which iteration the bit-flip occurs.

To tolerate the error in outer solver, we adopt simple double modular redundancy (DMR) [15]. It executes the outer solver twice and compare the results. DMR based method may fail to tolerate the memory error staying in both executions. To address this problem, at each error-free iteration, we take snapshots of subspace basis $[v_1, v_2, \ldots, v_{j-1}]$, $[z_1, z_2, \ldots, z_{j-1}]$, matrix H, and result vector x_j. Notice that GVR provides resilience mechanism for these snapshots. Reloading A, b, and these snapshots in previous iteration from GDS objects before the second execution, can detect memory errors in the original execution. If any inconsistency between two outer solver executions, the third one will be triggered to identify the correct execution. This approach has low overhead, high error checking accuracy and prevents error propagation to the next iteration.

5 Experiments

Based on our implementation from GVR and Trilinos, we run 128 processes with 5 matrices from the Florida sparse collection. The data is the average result of 1,000 error trials for each error probability and matrix. As the error impact studies, we mainly focus on significant error in the first 2 bytes of double precision data.

5.1 GVR Overhead in Error-free Execution

In our GVR enabled FGMRES, we create GDS objects on the solution and basis vectors, put the data into GDS objects and make the versions. We vary the number of processes to study the overhead of GVR in error-free execution. Here we take one snapshot of z_j, $[v_1, v_2, \ldots, v_{j-1}]$, $[z_1, z_2, \ldots, z_{j-1}]$, H, and x_j at each inner-outer iteration.

As shown in Figure 3, the overhead is less than 15% and keeps stable with the increasing of processes. The major overhead is on versioning since we use collective call to get consistent snapshot. We plan to bundle these vectors into one GDS object thus reducing the number of versioning operation.

5.2 Inner solver

To explore a range of error rates, we vary probability of significant error inside of inner solver computation or the result vector z_j. The recovery process is triggered only if the inner solver residual becomes $100\times$ larger than the previous iteration. We compare the total execution time without recovery and with our inside/outside resilience method. We

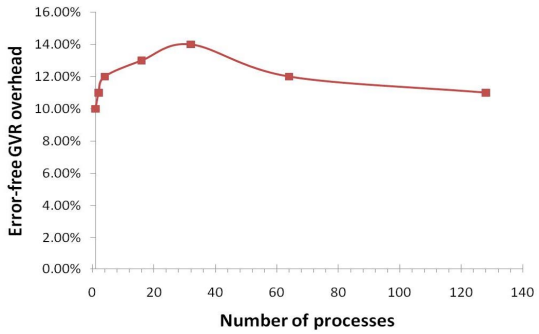

Fig. 3: GVR Overhead in Error-free Execution. Here one snapshot of related vectors are taken at each inner-outer iteration.

calculate the *slowdown* as the ratio between the total execution time with errors and the failure-free execution time.

Our results of slowdown (see Fig. 4) show that both recomputing and multi-versioning based recovery outperforms the original FGMRES with restarting. When error probability is $< 11\%$, recomputing has lower slowdown than multi-versioning based recovery due to the cost of snapshot. As the growth of error probability, multi-versioning based recovery becomes more beneficial by reducing the work loss.

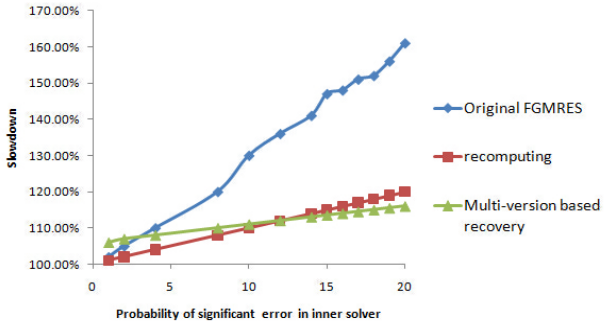

Fig. 4: Execution slowdown - original FGMRES, recomputing based recovery, and multi-version based recovery.

5.3 Outer solver

We vary probability of significant error in the basis vectors and solution vectors of outer solver and present the slowdown in Figure 5. The significant error in outer solver always leads to divergence for FGMRES without restarting. When the error probability is low, restarting can tolerate the error but introduce extreme high overhead, e.g. 1682.5% slowdown on 10% of error probability. As the increasing of error probability, it also becomes divergence. Our GVR enabled FGMRES successfully addresses the high error probability with relatively small overhead because it can isolate the errors in each iteration.

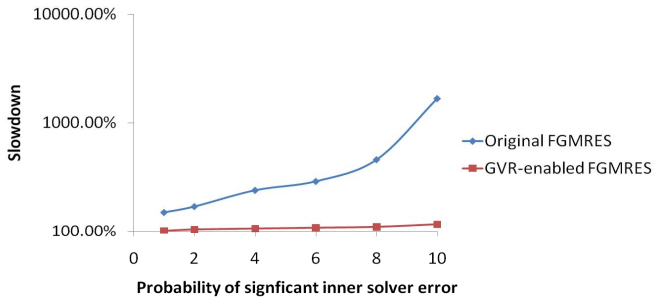

Fig. 5: Execution slowdown - original FGMRES, recomputing based recovery, and multi-version based recovery.

6 Related work

In large-scale system, traditional studies have focused on system level checkpoint/restart to tolerate fail-stop process failures [16]. As the growing concern around soft errors, more recent studies have focused on application level and cross layer solutions, especially for numeric solvers. Huang and Abraham developed the checksums based algorithm-based fault tolerance (ABFT) technique for matrix operations [14]. In [6], Chen developed theoretical conditions based error checking for Krylov subspace iterative methods. In [3], Bronevetsky analyzed soft error vulnerability for linear solvers. In [18], fault tolerant PCG solver is presented for sparse linear systems. Du presented encoding strategy for LU factorization based dense liner systems [8]. Unlike these works, this study is focusing on inner-outer solver.

The studies on fault tolerant inner-outer solver are limited. In [5], Chen analyzed flexible BiCGStab to bound the inner solver error for convergence. In [9], Elliott studied the impact of inner solver error in FGMRES. In [2] FGMRES solver was extended to tolerate inner solver error. Distinguished from these studies, this paper presents comprehensive error analysis for FGMRES and develops GVR-enabled methods for both inner and outer solvers under various error rate.

7 Summary and Future Work

We analyze the impact of bit-flip errors on the FGMRES inner-outer solver, which can lead to divergence failure or extreme high computation overhead. Based on the analysis results, we design the error checking/recovery strategies for inner solver and outer solver. We implement it by extending Trilinos solver library with our Global View Resilience (GVR) system. Our experimental results illustrate that our GVR-enabled inner-outer solver successfully tolerate the bit flip errors for execution convergence with low performance overhead.

Interesting research directions include, consideration of a wider range of inner-outer solver configurations, error checking and recover methods, and a wider range of GVR features. Also, future resilience study clearly should expand the class of memory errors considered or even including errors in other hardware elements.

Acknowledgement

We would like to thank Mark Hoemmen from Sandia National Laboratories for his advice. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Award DE-SC0008603 and Contract DE-AC02-06CH11357. This work was supported by the U.S. Department of Energy (DOE) National Nuclear Security Administration (NNSA) Advanced Simulation and Computing (ASC) program. 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 National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

1. S. Borkar and A. A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.
2. P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen. Fault-tolerant linear solvers via selective reliability. *ArXiv e-prints*, June 2012. Provided by the SAO/NASA Astrophysics Data System.
3. G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proc. of ICS*, 2008.
4. F. Cappello, A. Geist, W. Gropp, L. Kale, W. Kramer, and M. Snir. Towards exascale resilience. *International Journal of High Performance Computing Applications*, 23(4):374–388, 2009.
5. J. Chen, L. Curfman McInnes, and H. Zhang. Analysis and practical use of Flexible BiCGStab. Technical Report ANL/MCS-P3039-0912, Argonne National Laboratory, 2012.
6. Z. Chen. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proc. of PPoPP*, 2013.
7. T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011.
8. P. Du, P. Luszczek, and J. Dongarra. High performance dense linear system solver with resilience to multiple soft errors. In *Proc. of ICCS*, 2012.
9. J. Elliott, M. Hoemmen, and F. Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *Proc. of IPDPS*, 2014.
10. A. Chien et al. Global View Resilience Project (GVR) website. http://gvr.cs.uchicago.edu.
11. M. Elnozahy et al. System resilience at extreme scale, 2009. White Paper written for the Defense Advanced Research Project Agency (DARPA), with Ricardo Bianchini et al.
12. M. Heroux et al. An overview of the trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, September 2005.
13. P. Kogge et al. Exascale computing study: Technology challenges in achieving exascale systems. Technical Report TR-2008-13, University of Notre Dame CSE Department, 2008.
14. K. Huang and J. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, C-33(6):518–528, 1984.
15. J. Lidman, D. J. Quinlan, C. Liao, and S. A. McKee. ROSE::FTTransform – a source-to-source translation framework for exascale fault-tolerance research. In *DSN-W*, 2012.
16. A. Moody, G. Bronevetsky, K. Mohror, and B. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proc of Supercomputing*, 2010.
17. Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, second edition, 2003.
18. M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *Proc. of ICS*, 2012.