SAND2014-0174C

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

Ziming Zheng[1], Andrew A. Chien[1], Mark Hoemmen[2], and Keita Teranishi[3]

[1] University of Chicago, Chicago IL 60637, USA
[2] Sandia National Laboratories, Albuquerque NM 87123, USA
[3] Sandia National Laboratories, Livermore, CA 94551, USA

Abstract. Resilience is a major challenge for large-scale systems. Inner-outer solvers such as Flexible Generalized Minimal Residual Method (FGMRES) are widely-used for scientific applications, playing a key role in performance and resilience. Though FGMRES is robust to soft error, we show that single bit flip errors can lead to high computation overhead or even divergence (failure). 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 FGMRES 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 [15, 12, 14, 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 [1, 2]) 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.

With these insights, we design and evaluate error checking and recovery strategies for inner-outer solver. Our experiments employ the Trilinos library [6], extending FGMRES inner-outer solver with the Global View Resilience (GVR) framework [5], use 5 matrices from the Florida sparse collection [7], running on up to 128 processes. Specific contributions include:

– Characterizing situations where bit-errors cause resilience problems for both inner and outer solvers in FGMRES.

– Define error rate regimes where recovery with different methods (cost, granularity) is appropriate.
– Employ GVR programming model to implement portable and rich error checking/recovery strategies.
– 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 3 explores the impact of errors in key data structures. Section 2 introduces the background of GVR and Trilinos for our implementation. Section 4 presents error checking and recovery methods. 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) [5] and Trilinos[6]. 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 and design flexible, portable and efficient fault management for each GDS object. We extend the kernel classes of Trilinos using GVR APIs. 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. GDS objects are created 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.
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 undetected for some iterations.
3. 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.
4. 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.

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

3 Error Impact on an Inner-Outer 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.

To study the impact of errors on Flexible GMRES, we inject errors randomly into key data structures. For the inner solver, we focus on the result vector z_j as the most important data visible to the outer solver. For outer solver, we focus on the basis vectors $[v_1, v_2, \ldots, v_j]$, $[z_1, z_2, \ldots, z_{j-1}]$ and Hessenberg matrix H.

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. For both inner and outer solvers, 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. Instead, a bit error in the first 2 bytes has high impact on the solver behaviors. Here we define the errors in the first 2 bytes of the double precision data as *significant error*.

3.1 Inner solver

The inner solver takes longest execution time and thus has the highest probability of experiencing a data corruption. However, inner solver result is approximate, so error impact is minimal if the error occurs in less significant bits. However, as shown in Fig. 2, a significant bit error from inner solver increases the number of outer solver iterations 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.

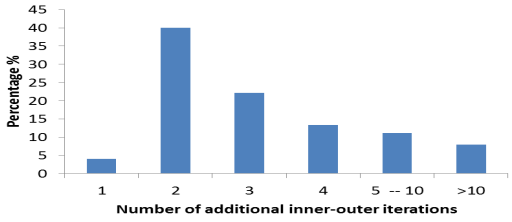

Fig. 2: Distribution of additional inner-outer solver iterations incurred by significant inner solver errors.

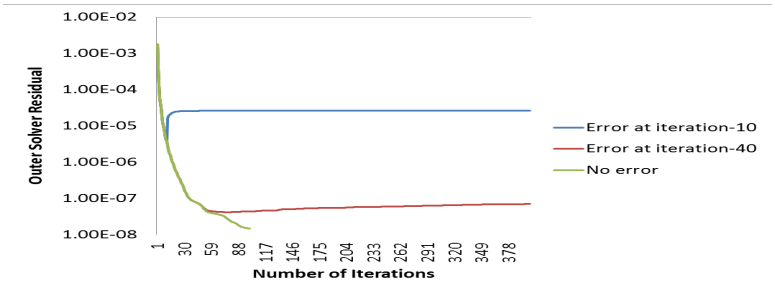

Fig. 3: An example of outer solver error impact on residual; a single error may cause convergence failure.

3.2 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 in the basis vectors or Hessenberg matrix H, the residual may increase or stay constant for several iterations (see Fig. 3). Even a single bit-flip may lead to divergence no matter at which iteration the bit-flip occurs.

4 Error Checking and Recovery in an Inner-Outer Solver

4.1 Inner solver: 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. The first is recomputing the inner solver [2], incurring high overhead since the inner solver is 90% of the computation. Despite that, recomputation 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.

4.2 Inner solver: inside

For higher error rates, it is necessary to handle the errors inside the inner solver. For example, we can use GVR-enabled GMRES or ABFT based CG solver [3]. The crossover for preference happens when the overhead of error recovery within the solver is less than ignoring and 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, and the overhead of ignoring error is Φ. Note that re-computing still has the error probability of P, so the expected re-computing overhead is

$$E(r, P) = \sum_{k=1}^{\infty} P^{k-1}(1-P)k = 1 + \frac{P}{1-P} \tag{1}$$

When $E(r, P) > \Phi$, the recomputing of whole inner solver becomes non beneficial.

4.3 Outer solver

An ideal error checking and recovery mechanisms for outer solver would have low overhead, high accuracy, high coverage, and iteration isolation. Iteration isolation means to prevent error propagation to the next iteration – critical for performance overhead. Achieving all three at once is difficult, so we study three different approaches.

Approach1: simple residual based checking If the residual increases signal error (the residual of GMRES should decrease monotonically [1]). The strategy has only 0.2% overhead and 100% accuracy, but the error may propagates across multiple iterations causing multiple basis vectors to be corrupted. Recomputation of the whole inner-outer solver iteration multiple times may produce high recovery overhead. Also it is difficult to identify which iteration to rollback.

Approach2: double modular redundancy (DMR) checking [8]. Execute the outer solver twice and compare. Reloading the result from previous iteration before the second execution, can detect memory errors in the original execution. Much higher overhead (2%-10%), but excellent coverage, accuracy, and iteration isolation.

Approach3: algorithm-based checking. Here we check theoretical conditions during steps 4-13 such as the orthogonality relationship of basis vectors $v_{i+1}^T v_i = 0$ [3] and Hessenberg matrix norm bound $|h_{i,j}| < ||A||_F$ [9]. This approach has higher overhead than the residual based check and lower overhead than DMR based checking. For example, our studies show calculating $v_{i+1}^T v_i = 0$ incurs around 0.5% overhead However, this fine-grained approach only provides partial coverage.

The best checking approach depends on error rate and location. Low error rates warrant coarse-grained methods (first approach), moderate error rates (second approach), and at high error rates, aggressive fine-grained approaches to containment and recovery are required. We validate these tradeoffs in our experiments in Section 5.

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 Inner solver

To explore a range of error rates, we vary error probability in the inner solver result vector z_j. The recomputing 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 a our resilience method. We 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 recomputing based recovery outperforms original FGMRES without error checking for low error rates. Without restarting, the recomputing-based recovery becomes non-viable when the error probability is higher than 60% – consistent with Equation 1 – as each significant error introduces around 200% overhead, i.e., $\Theta = 2$. So when probability of significant error is $> 50\%$, $E(r, P) > \Theta$ and recomputing the whole inner solver is not beneficial.

With restarting, recomputing is infeasible for error rates higher than 40%; even lower than without restarting. Restarting can relieve the error accumulation, especially for the errors occurring at the end of the restart cycle.

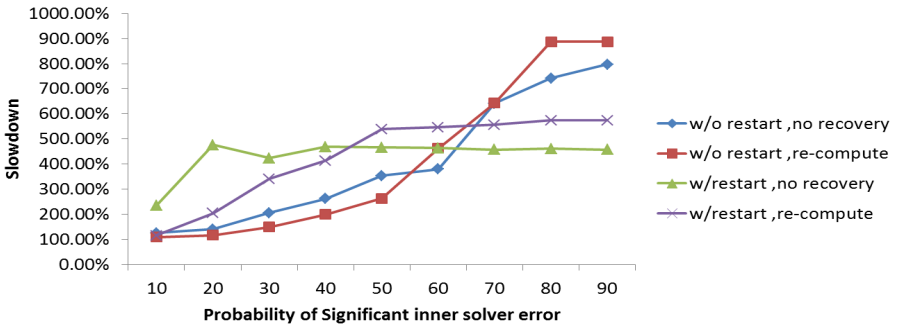

Fig. 4: Execution slowdown - recomputing based recovery, various approaches.

5.2 Outer solver

In the first set of experiments, we inject a single error in outer solver and study the performance of Approach-1 and Approach-2 (residual based checking and restarting), using FGMRES without/with restarting as baseline). The original FGMRES without restarting does not converge (see Fig. 3). Approach-1 can guarantee the convergence because the error is removed after restarting with 7.1% performance overhead in average. However, Approach-1 has the disadvantage of detection latency which may introduce higher overhead. In one extreme case the error is detected 70 iterations after the injection incurring 34.2% of the performance overhead.

Original FGMRES with restarting can also remove the single error. However, without detection, the error remains until the end of restarting cycle incurring high overhead. For example, matrix $mhd4800b$ with restarting length $m = 200$ incurs 260.7% overhead. Approach-1 can detect errors much earlier with only 2% of performance overhead in average. A simple strategy for original FGMRES is to reduce restarting length and

remove errors earlier. However, short restarting length can lead to stagnation of convergence since the subspace is too small. For example, FGMRES with $m = 25$ has 438.1% longer execution time than the case with $m = 200$. Therefore Approach-1 with long restarting length outperforms the original FGMRES with short restarting length. In addition, Approach-1 also outperforms Approach-2 for single error. Approach-2 generally introduces 11.3% of overhead, which is much higher because every outer iteration is forced to execute twice.

In additional experiments, we inject multiple errors. In Table 1, we present the execution overhead versus recovery strategy under various error probability for the outer solver. For higher error rates, both original FGMRES with restarting and Approach1 are not viable as errors may occur in each restarting cycle even after restarting. In constrast, Approach-2 successfully addresses the high error probability with relatively small overhead because it can isolate the errors in each iteration. Approach-2 and Approach-3 shows no significant improvement over Approach-2 because the outer solver execution time is short and Approach3 has limited error check coverage.

Table 1: Execution overhead of error checking/recovery strategies under different error probability of outer solver.

Error Probability	FGMRES w/ restarting	Approach1	Approach2	Approach2 + Approach3
10%	1682.5%	909.5%	16.4%	14.8%
20%	divergence	3616.6%	17.6%	15.8%
30%	divergence	divergence	19.8%	17.1%
40%	divergence	divergence	23.4%	20.7%
50%	divergence	divergence	27.5%	24.1%
60%	divergence	divergence	34.4%	30.4%
70%	divergence	divergence	46.5%	42.2%
80%	divergence	divergence	70.3%	53.4%
90%	divergence	divergence	140.5%	77.2%

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 [11]. In [3], Chen developed theoretical conditions based error checking for Krylov subspace iterative methods. In [4], Bronevetsky analyzed soft error vulnerability for linear solvers. In [10], fault tolerant PCG solver is presented for sparse linear systems. Du presented encoding strategy for LU factorization based dense liner systems [17]. Unlike these works, this study is focusing on inner-outer solver.

The studies on fault tolerant inner-outer solver are limited. In [18], 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 Conclusion

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.

Acknowledgement

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, as well as by Sandia Laboratories Contract 1309936.

References

1. Y. Saad. Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics. Second Edition. April 30, 2003.
2. Mark Hoemmen and Michael A. Heroux. Fault-tolerant iterative methods via selective reliability. Sandia National Laboratories Technical Report, June 2011.
3. Z. Chen. Online-ABFT : An Online Algorithm Based Fault Tolerance Scheme for Soft Error Detection in Iterative Methods. Proc. of PPoPP, 2013.
4. G. Bronevetsky and B. de Supinski, Soft error vulnerability of iterative linear algebra methods, Proc. of ICS, 2008.
5. Global View Resillienc Project(GVR). http://gvr.cs.uchicago.edu.
6. M. Heroux and et. al., An Overview of Trilinos, Sandia National Laboratories Technical Report, SAND2003-2927, 2003.
7. T. Davis, University of Florida Sparse Matrix Collection, NA Digest, 97(23), June 1997.
8. J. Lidman, D. Quinlan, C. Liao and S. McKee, ROSE::FTTransform-A Source-to-Source Translation Framework for Exascale Fault-Tolerance Research, FTXS workshop, 2011.
9. J. Elliott, M. Hoemmen, and F. Mueller, Evaluating the Impact of SDC on the GMRES Iterative Solver, Proc. of IPDPS, 2014.
10. M. Shantharam, S. Srinivasmurthy, and P. Raghavan, Fault tolerant preconditioned conjugate gradient for sparse linear system solution, Proc. of ICS, 2012.
11. K. Huang and J. Abraham. Algorithm-based fault tolerance for matrix operations. IEEE Transactions on Computers, vol. C-33:518C528, 1984.
12. S. Borkar and Andrew A.Chien, The future of microprocessors, Communications of the ACM, 54(5), 67–77, 2011.
13. F. Cappello, A. Geist, W. Gropp, L. Kale, W. Kramer, M. Snir, Towards Exascale Resilience, International Journal of High Performance Computing Applications, 23(4), 374-388, 2009.
14. M. Elnozahy, System Resilience at Extreme Scale: A White Paper, DARPA Resilience Report for ITO, William Harrod, 2009.
15. Peter Kogge, et. al., Exascale Computing Study: Technology Challenges in Achieving an Exascale Systems, DARPA IPTO Study Report for William Harrod, 2008.
16. A. Moody, G. Bronevetsky, K. Mohror, and B. Supinski, Design, modeling, and evaluation of a scalable multi-level checkpointing system, Proc of Supercomputing, 2010.
17. P. Du, P. Luszczek, J. Dongarra, High Performance Dense Linear System Solver with Resilience to Multiple Soft Errors. Proc. of ICCS, 2012.
18. J. Chen, L. Curfman McInnes and H. Zhang, Analysis and Practical Use of Flexible BiCGStab, ANL/MCS-P3039-0912, 2012.