Evaluating a Persistent Soft Fault Model on Preconditioned Iterative Methods

Evan Coleman[1,2], **Masha Sosonkina**[2]
[1]Naval Surface Warfare Center - Dahlgren Division, Dahlgren, VA, USA
[2]Modeling, Simulation, and Visualization Engineering Department, Old Dominion University, Norfolk, VA, USA

Abstract— *The impact of soft fault errors on the GMRES iterative method with flexible preconditioning (called FGMRES) is explored. In particular, a new method for simulating soft fault errors is implemented directly in FGMRES, and the effect of error magnitude and timing is evaluated for the FGMRES convergence in solving of an elliptical PDE problem on a regular grid. Two types of preconditioners are explored, featuring an incomplete LU factorization and an algebraic recursive multilevel solver ARMS. The experiments have confirmed an intuition that, in general, injecting perturbation-based faults at the matrix-vector operation stage had a greater impact on the convergence rate than doing so at the preconditioning application stage, resulting in more cases when the iterative solver failed. In addition, several cases of better convergence under faults in preconditioning operation were observed and analyzed.*

Keywords: iterative solvers, preconditioning, fault model, flexible GMRES, pARMS

1. Introduction

Fault tolerance methods are devised to increase both reliability and resiliency of high-performance computing (HPC) applications on exascale platforms, in which the mean time to failure (MTTF) is projected to decrease dramatically due to the sheer size of the computing platform [4]. There are many reports (e.g., [1], [4], [16]) that discuss the expected increase in the number of faults experienced by HPC environments. This is expected to be a more prevalent problem as HPC environments continue to evolve towards larger systems. As the landscape of HPC continues to grow into one where experiencing faults during computations is increasingly commonplace, the software used in HPC applications needs to continue to change alongside it in order to provide an increased measure of resilience against the increased number of faults experienced. Typically, faults are divided into two categories: hard faults and soft faults (see, e.g., [6], [10]). Hard faults come from negative effects on the physical hardware components of the system and cause program interruption. As hardware components themselves continue to evolve and grow both smaller and faster, they (generally) become more prone to error, and the algorithms and software packages that are used in HPC environments need to be able to respond to sudden and unexpected changes in both the quantity and quality of the physical resources that may be available for use. The other category of faults, soft faults, are the focus of this work. This category of failures captures all faults that a program might experience that do not immediately interrupt program execution. Most often, these faults refer to some form of data corruption that is occurring either directly inside of, or as a result of, the algorithm that is being executed. It is possible for a program to detect the presence of a soft fault while it is still executing. In order to properly investigate the impact of soft errors, one needs to select a fault model that fully encapsulates all of the potential impacts of a soft fault, implement the selected fault model into the algorithm to be investigated, and conduct the necessary experiments to determine the potential impact of a fault occurring during the selected algorithm. Typically, soft faults have been modeled by a bit flip. This study focuses on utilizing an arguably more general approach towards the modeling of soft faults, and subsequently evaluating it in the case study of the Flexible GMRES (FGMRES) [14] iterative solver. Faults were injected as small perturbations to results of certain mathematical operations using a modified version of fault injection found in [6] and [8].

The rest of the paper is organized as follows: in Section 2, a brief overview of related studies is provided, in Section 3, details concerning the fault model that is used throughout this work are given, in Section 4, experimental results are provided, and in Section 5, a quick summary is presented along with possible directions for future work.

2. Related Work

Traditionally, when performing experiments to analyze the potential impact of soft faults upon a computing environment, researchers have relied primarily upon the injection of bit flips into a particular portion of the routine [3], [9]. In contrast, in the work by Elliot, Hoemmen, and Mueller [8], [6], faults are modeled in a more general sense. These studies choose to generalize the simulation of soft faults to producing an incorrect solution to one of key computational parts, such as the application of the preconditioner inside of an iterative solver. This approach generalizes the simulation of soft faults by disregarding the actual source of the fault

and allowing the fault injector to create as large or as small a fault as necessary for the experiment. In the experiments conducted in [8], [6], [7] faults are typically defined as either a scaling of the contribution of the result of the preconditioner application for the Message Passing Interface (MPI) process in which a fault was injected, or a permutation of the components of the vector result of the preconditioner application for the MPI process in which a fault was injected.

In the taxonomy of faults given in [6], [10] soft faults are divided into the categories of transient, sticky, and persistent. Transient faults are defined as faults that occur only once, sticky faults indicate a fault that recurs for some period of time but where computation eventually returns to a fault-free state, and persistent faults arise when the fault is permanent. Whether the studies discussed above model faults using bit flips or adopt a more numerical analysis style approach, much of the previous work on the impact of silent data corruption (SDC) has to do with modeling transient errors. The goal of this effort is to present a fault model that can accurately predict the impact of persistent soft faults. Examples of scenarios that could cause a persistent fault are a stuck bit in memory, or a hardware malfunction – such as the Intel Pentium FDIV bug – or the incorrect copy of data from one location to another. [6], [5], [10] The model presented here is general enough that it can be adapted to simulate the impact of any persistent error, including those caused by hardware malfunction.

Traditional analysis of potential persistent type errors has rested more in the hardware domain than in the algorithmic domain, with analysis of both processor based faults [11], [2] and memory based faults [15]. The impact of persistent faults on iterative methods does not seem to have explored to a great extent. The work presented here follows an idea from [8], [6], [7] of disregarding the source of the error in the simulated fault. In other words, an analytical approach is taken as opposed to flipping random bits inside some pertinent data structures. On the other hand, here the simulated soft faults persist once injected as opposed to work in [8], [6] where the faults are transient in nature.

3. Fault Model

The approach chosen was to perturb the vector result of key computations for the single MPI process in which a fault was injected. This perturbation-based fault model is an adaptation of the fault model presented in [6], [8], [7]. The fault model presented in [6], [8], and [7] focuses exclusively on modeling transient faults; the fault model presented here attempts to modify that approach to extend it to persistent soft faults. In an attempt to accurately model the impact of a persistent soft fault, a small randomized perturbation is injected on each iteration after the fault is modeled to occur. Adopting a characterization from [7], faults are divided based upon their impact to the l^2-norm of the vector they are injected into. The default version of the fault model presented here relies on the generation of small random numbers that are added to each element of the data structure where the fault is injected. However, the fault model was also adapted to allow for the possibility of either moving each element of the vector to be perturbed further away from, or closer towards, zero. In this way, the fault model offers some level of control over whether the l^2-norm of the vector the fault is injected into increases or decreases. This allows observations about the effect of perturbing the l^2-norm on the general convergence of the total algorithm. Since FGMRES works towards reducing the l^2-norm of the residual vector, modifying the l^2-norm of any of the vectors used to construct the residual vector may affect its convergence rate by affecting the l^2-norm of the residual in a ripple-like effect [7], [14]. Hence, one of three outcomes may occur: The iterative solver will converge at about the same rate as without perturbation, or converge but will take significantly longer to reach convergence, or fail to converge entirely [8]. However, it is also possible that the injected fault will actually cause the iterative solver to converge in fewer iterations than without perturbations.

It is important to design the fault model in such a way that it encapsulates the worst-case behavior that one is trying to protect against. By modeling faults as a random perturbation, a controlled amount of noise is added to the result of key operation inside of the algorithm. The amount of this noise is parameterized throughout the different experiments. However, the random nature of the faults limits the knowledge of specific details regarding the fault that was injected. As persistent faults are one of three types of soft faults accounted for in the taxonomy of soft faults presented in [6], [10], designing a fault model to accurately measure the impact of these faults is an important endeavor. The nature of the fault model presented here, in that every iteration inside of the iterative solver is perturbed after the fault initially occurs, provides a way to quantify the impacts of a potential persistent soft fault.

3.1 FGMRES

The FGMRES algorithm, as described in [14], is provided in Algorithm 1. FGMRES is similar in its nature to the standard GMRES with the notable exception of allowing the preconditioner to change in each iteration by storing the result of each preconditioning operation (cf. matrix Z_m in line Line 10). FGMRES was selected in this study because it is a robust, popular iterative solver which is proven to converged under variable preconditioning, possibly resulting from a perturbation in the preconditioning operation. Here, such a perturbation is due to injected faults. In particular, faults were injected at two distinct points inside of the FGMRES algorithm; Line 1, termed here as the *outer matvec* operation, and Line 3, which the application of the preconditioner. In this study, the effect of injecting faults exclusively into one of these two locations as well as into both locations

simultaneously was considered. Also, the GMRES restart parameter (*m* in Algorithm 1) was taken to be 20.

Input: A linear system $Ax = b$ and an initial guess at the solution, x_0
Output: An approximate solution x_m for some $m \geq 0$
1 $r_0 = b - Ax_0,\ \beta = ||r_0||_2,\ v_1 = r_0/\beta$
2 **for** $j = 1, 2, \ldots, m$ **do**
3 $\quad z_j = M_j^{-1} v_j$
4 $\quad w = Az_j$
5 $\quad$ **for** *i = 1, 2, ..., j* **do**
6 $\quad\quad h_{i,j} = w \cdot v_i$
7 $\quad\quad w = w - h_{i,j} v_i$
8 $\quad$ **end**
9 $\quad h_{j+1,j} = ||w||_2,\ v_{j+1} = w/h_{j+1,j}$
10 $\quad Z_m = [z_1, \ldots, z_m],\ \bar{H}_m = h_{i,j\,1 \leq i \leq j+1; 1 \leq j \leq m}$
11 **end**
12 $y_m = argmin_y ||\bar{H}_m y - \beta e_1||_2,\ x_m = x_0 + Z_m y_m$
13 **if** *Convergence was reached* **then return** x_m
14 **else** set $x_0 \leftarrow x_m$, **GoTo** Line 1

Algorithm 1: FGMRES as given in [14]

3.1.1 Preconditioner

A traditional linear system is given by $Ax = b$, however a transformed *preconditioned system* is given by $M^{-1}Ax = M^{-1}b$, when preconditioning is applied from the left, and $AM^{-1}y = b$ with $x = M^{-1}y$, when preconditioning is applied from the right. The matrix M is a nonsingular approximation to A, and is called the *preconditioner*. Incomplete LU factorization methods (ILUs) are an effective class of preconditioning techniques for solving linear systems. They define the preconditioner as $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are approximations of the L and U factors of the standard triangular LU decomposition of A. The incomplete factorization may be computed from the Gaussian Elimination (GE) algorithm, by discarding some entries in the L and U factors. If the m independent unknowns are numbered first, and the other $n - m$ unknowns last, the coefficient matrix of the system is permuted in the 2×2 block structure. In multi-elimination methods, a reduced system is recursively constructed from the permuted system by performing a block LU factorization of PAP^T of the form

$$PAP^T = \begin{pmatrix} D & F \\ E & C \end{pmatrix} = \begin{pmatrix} L & 0 \\ G & I_{n-m} \end{pmatrix} \times \begin{pmatrix} U & W \\ 0 & A_1 \end{pmatrix}$$

where D is a diagonal matrix, L and U are the triangular factors of the LU factorization of D, $A_1 = C - ED^{-1}F$ is the Schur complement with respect to C, I_{n-m} is the identity matrix of dimension $n - m$, and then denote by $G = EU^{-1}$ and $W = L^{-1}F$. The reduction process can be applied another time to the reduced system with A_1, and recursively to each consecutively reduced system until the Schur complement is small enough to be solved with a standard method. The factorization of PAP^T above defines a general framework which accommodates for different methods. The ARMS preconditioner uses block independent sets to discover sets of independent unknowns and computes them by using the greedy algorithm. In the ARMS implementation used here, the incomplete triangular factors $\bar{L}$, $\bar{U}$ of D are computed by one sweep of ILUT. In the second loop, an approximation $\bar{G}$ to $E\bar{U}^{-1}$ and an approximate Schur complement matrix $\bar{A}_1$ are derived. This holds at each reduction level. At the last level, another sweep of ILUT is applied to the (last) reduced system.

3.1.2 Fault Detection and Resilience in FGMRES

Fault detection inside of FGMRES can be achieved in many different ways. Upon each restart of FGMRES, the norm of the residual is computed, and in a fault-free environment these norms should be monotonically decreasing. A cheap fault detector could be implemented to check this, and it would be an intuitive way to attempt to detect faults that occur during the outer sparse matrix-vector multiply. It will be shown experimentally that if the fault that is injected into the outer sparse matrix-vector multiply does not increase the norm of the initial residual, than it has a significantly less negative effect on the convergence of FGMRES.

One of the key observations made in [10] was that since the preconditioner is allowed to change on every iteration in the FGMRES algorithm, faults that occur during the precondioning operation (Line 3 in Algorithm 1) can be modeled as different preconditioners. As such, if a fault were to occur anywhere inside of the preconditioning operation it can be modeled by injecting a fault into the result of the preconditioning operation (z_j in Algorithm 1). The perturbation-based fault model proposed in this paper allows the size of the fault to be controlled by offering direct control on the size of the perturbation that is injected.

It will be shown experimentally that FGMRES is capable of proceeding through many faults occuring in the preconditioning operation by accepting the faulty output as a different preconditioner. This natural adaptive response in the FGMRES algorithm to faults that occur during preconditioning should also cause faults that occur during the outer sparse matrix-vector multiply to have more of an impact on the convergence of FGMRES. This was also able to be shown experimentally, and results are provided in Section 4.

4. Experimental Results

The test problem that was used comes directly from the pARMS library [12], and represents an elliptic 2D partial differential equation,

$$-\Delta u + 100\frac{\partial}{\partial x}(e^{xy}u) + 100\frac{\partial}{\partial y}(e^{-xy}u) - 10u = f$$

Parameter	Acceptable Values
Global Preconditioner	Block Jacobi
Local Preconditioner	ILUT, ARMS
Tolerance Required for Convergence	10^{-6}
Starting Iteration at which Fault Appears	≥ 5
Order of Perturbation	$10^{-6}, \ldots, 10^{-4}$
Effect on l^2-norm	Any, Decrease, Increase

Table 1: Input parameters the value of which varied in the experiments.

on a square region with Dirichlet boundary conditions, discretized with a five-point centered finite-difference scheme on a $n_x \times n_y$ grid, excluding boundary points. The mesh is mapped to a virtual $p_x \times p_y$ grid of processors, such that a subrectangle of $r_x = n_x/p_x$ points in the x direction and $r_y = n_y/p_y$ points in the y direction is mapped to a processor. The size of the problem was varied and controlled by changing the size of the mesh that was used in the creation of the domain. The mesh sizes that were considered ranged from $n_x = n_y = 100$ to $n_x = n_y = 500$, and these mesh sizes were run on numbers of processors that varied from four ($p_x = p_y = 2$) to 100 ($p_x = p_y = 10$). In order to compare experiments run with different parameters the resulting number of iterations was compared to the number of iterations in the same unperturbed run and depicted as the percentage in the plots throughout this section.

In all of the experiments that were conducted, multiple sets of runs were executed for each set of parameters (i.e. perturbation size, iteration fault was first injected) and their effect on the convergence of FGMRES was combined into an average with all other runs with the same parameters before being analyzed. The parameters that were varied in these experiments are detailed in Table 1.

Since the fault model presented here is based on a series of random perturbations, multiple runs/solves were conducted for each set of parameters, and the results were averaged and depicted in the plots of this section. In all of the experiments, a maximum number of iterations was instituted and, if a run did not converge within this preset number of iterations, then it was terminated, and determined to have failed. The focus is on investigating the effects on two of the local preconditioners available within pARMS: Incomplete LU with dual nonzero dropping strategy [14] (referred to as ILUT), and the ARMS preconditioner [13].

The experiments conducted for this study were run on two distinct hardware environments. The first test environment was a node with Intel Core i7 processor having four physical cores at 2.50 GHz each and 16 GB of main memory. The second was the Hopper supercomputer, which is a compute resource of the National Energy Research Scientific Center (NERSC). Hopper has a total of 153,216 compute cores, 212 Terabytes of memory and nodes are connected with a custom high-bandwidth, low-latency network provided by Cray. Up to five compute nodes of Hopper were utilized. The problem size was scaled appropriately for each environment, by adjusting the size of the square mesh per subdomain; namely, 200 and 500 points for the Intel Core i7 and Hopper, respectively.

The results shown in all the figures of Sections 4.1 to 4.3 come from runs on the Intel Core i7 platform using four MPI ranks, one per core. The results from the runs with larger problem sizes performed on Hopper showed similar convergence tendencies under perturbation-based faults considered here. Note that, in all the plots, the x-axis represents the fraction (as %) of the execution when a fault begins and the y-axis shows the increase (or decrease) in the number of iterations with respect to non-perturbed case. For example, a data point with x coordinate of 50% shows an effect from the fault injected *halfway through the number of iterations that would be required by a fault-free run*. This effect is quantified by the y coordinate of the point, such that, if $y = +100\%$, e.g., then the run corresponding to this data point required twice as many iterations to converge than that did in a fault free case.

4.1 Matvec Perturbations

The first set of experiments affected only the outer matvec operation in the FGMRES algorithm (see Line 1). The following results are provided the instance where the sign of the perturbation (and hence, the magnitude of the l^2-norm) was not controlled by the fault model. Figure 1 shows the effects of faults with various perturbation sizes in outer matvec when the ARMS (top) or ILUT (bottom) local preconditioner is used. Only perturbation sizes no larger than 5×10^{-5} are shown since for larger values the solver failed to converge. Comparing the results in Fig. 1 (top) and (bottom), a similar convergence behavior may be observed. However, the faults corresponding to smaller perturbations ($10^{-6}, \ldots, 5 \times 10^{-5}$) have a slightly greater negative effect on the runs with the ILUT preconditioner than on those with ARMS. When examining effects of very small perturbations (on the order of convergence tolerance, which is 10^{-6} here), it was found that they had either no effect at all on the convergence rate or slightly decreased the total number of iterations. This beneficial effect was noted regardless of when during the run the fault has started, and it appears more often with the ARMS preconditioner than with ILUT.

Next, results for the case where the sign of the perturbation was matched with the sign of the existing vector component in order to ensure that the l^2-norm of the perturbed operation result decreased (Fig. 2). In order to match the sign appropriately, the fault model checks the sign of the original vector component before applying the fault.

It is interesting to observe in Fig. 2 the increased rate of successful convergence for a much larger range of fault magnitudes. A larger spectrum of perturbation sizes resulted in successful convergence and, hence, is represented in Fig. 2. Comparing the effects of injecting faults that vary the l^2-

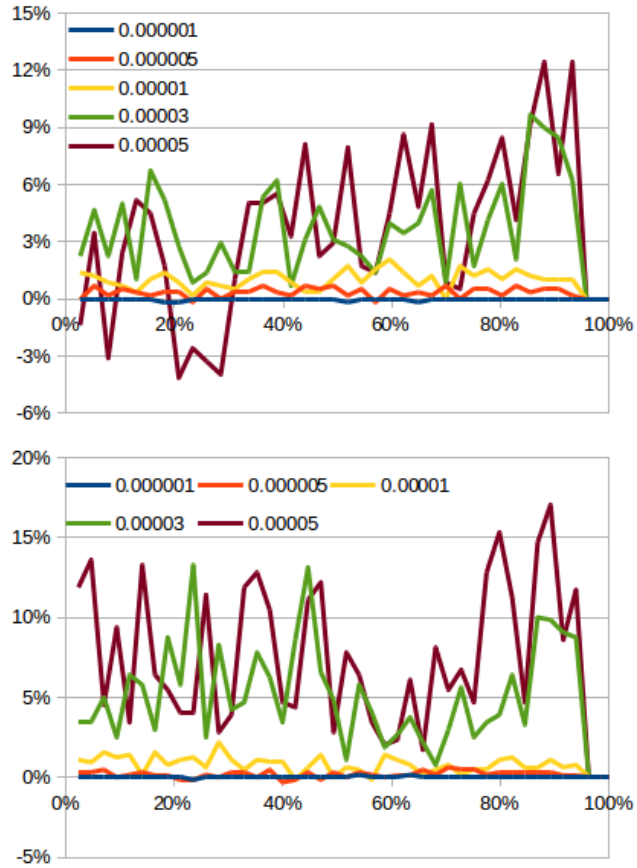

Fig. 1: Outer matvec perturbation faults with varied l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

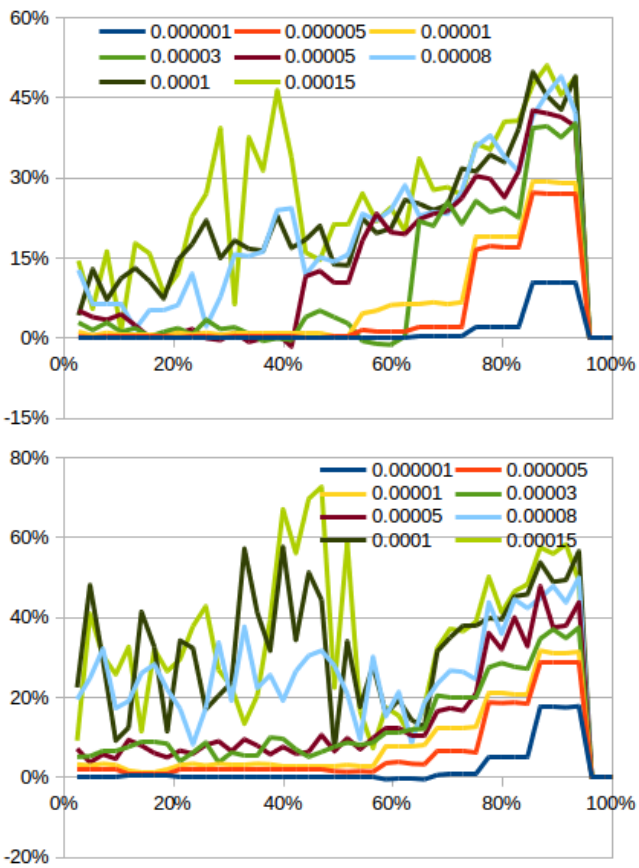

Fig. 2: Outer matvec perturbation faults with decreasing l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

norm (Fig. 1) to those that shrink the l^2-norm (Fig. 2), there is also a decrease in the negative effect that a fault of the same magnitude has upon the FGMRES algorithm. In general, the performance of the two preconditioners is fairly similar in the case when faults are incurred in the outer matvec operation. For instance, as expected, there is a tendency for the fault to have more of an impact on the convergence if the fault commences later in the execution; smaller perturbations show little effect while larger perturbations produce a much higher variations of convergence results. Perturbed executions resulting in fewer iterations than non-perturbed ones appear to arise with about equal frequency between scenarios using either the ARMS or the ILUT preconditioner. These results are seen for all the fault sizes—although much more commonly for faults of size $\leq 10^{-4}$—and are observed most when faults occur before the run reaches approximately 60% of completion of a fault-free run.

4.2 Preconditioner Perturbations

Results (Fig. 3) are presented for each of the two preconditioners, ARMS and ILUT, and exclusively for the version of the perturbation-based soft fault model that decreases the l^2-norm of the vector that it is applied to. Comparing the results with the ILUT preconditioner to those with ARMS, it again appears that the runs with the latter suffer less of a negative effect than those with the former for the faults of an equivalent size. Next, when examining results with the ARMS preconditioner in Fig. 3(top), it is clear that injecting a perturbation-based fault into the result of the application of the preconditioner (from Line 3 in Algorithm 1) has less of an effect on a FGMRES solve using the ARMS preconditioner than that from injecting a similar fault into the result of the outer matvec iteration (Fig. 2(top)). Even a magnitude of fault (e.g., 10^{-4}) that may cause stagnation when injected into the outer matvec operation, causes only a 40–50% increase in the total number of iterations here and only has a large impact if injected throughout the

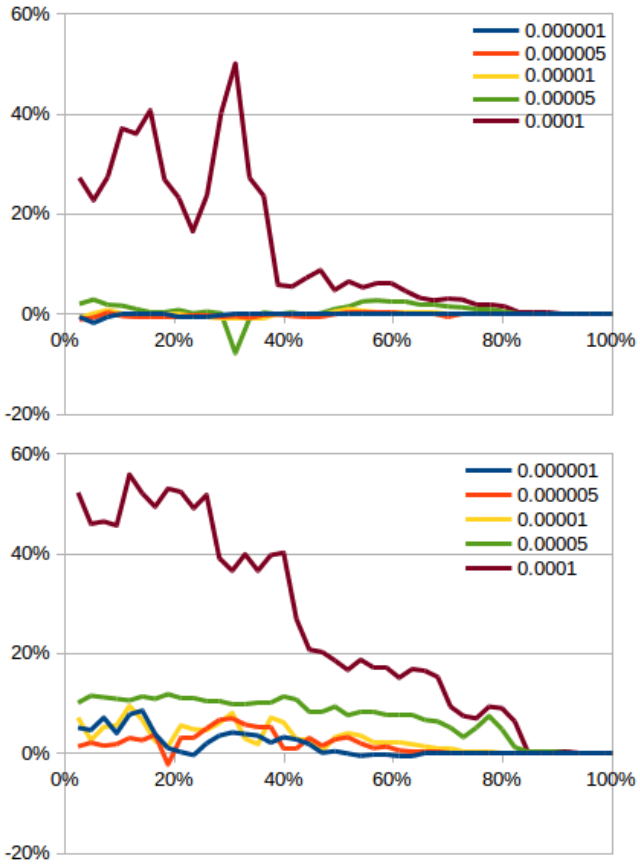

Fig. 3: Preconditioner perturbation faults with decreasing l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

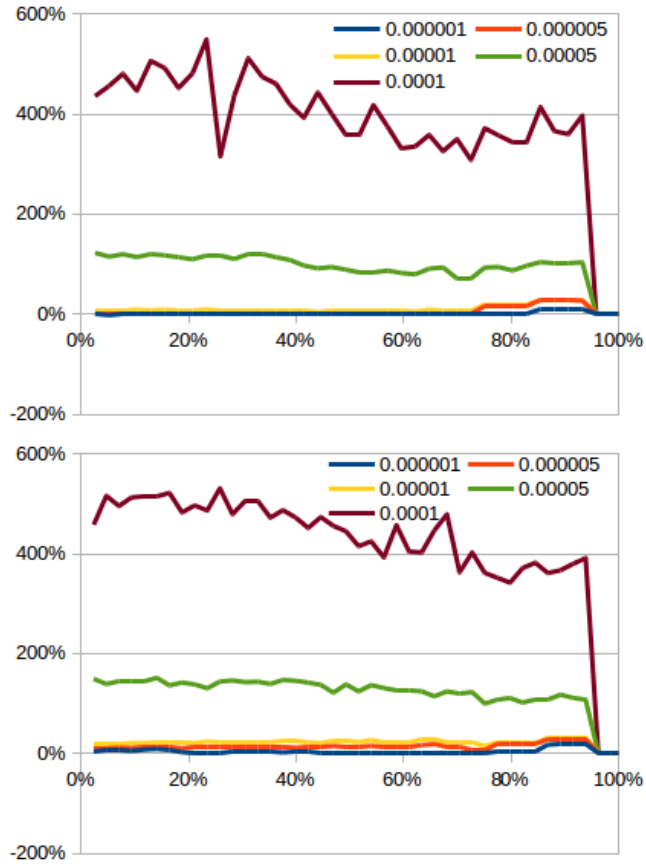

Fig. 4: Outer matvec and preconditioner application faults with decreasing l^2-norm for the ARMS preconditioner (top) and ILUT preconditioner (bottom). On y- and x-axis, % of extra iterations and of fault-commencing iteration, respectively, compared to the number of iterations in the fault-free run.

majority of the run. Similar observations may be made for the ILUT preconditioning in Fig. 3(bottom): less of a negative effect is evident when perturbation-based faults appear in this preconditioning operation than in the outer matvec (cf. Fig. 2(bottom)). In general, FGMRES, being able to converge with a preconditioner that changes at each iteration, does not negatively react to perconditioner changes due to faults in the course of linear system solution.

4.3 Matvec and Preconditioner Perturbations

The graphs in Fig. 4 show the effect of injecting a fault into the two fault sites considered simultaneously (i.e., at the same FGMRES iteration), the outer matvec *and* preconditioner application, such that the l^2-norm decreases. In Fig. 4, notice that, for large faults (starting at 10^{-4}), the increase in the number of iterations required to converge was very high—between 400-600% at times. This increase is also much higher than that for either matvec- or preconditioner-only incurred faults producing the highest increases of $\sim$60% and $\sim$55%, respectively, for the same perturbation value of 10^{-4}. Conversely, for perturbation sizes of 10^{-5} and smaller, the effect on convergence appears similar to that of either matvec faults. This suggests that the ability of FGMRES to accept faulty preconditioners is inhibited by the coexistence of a matvec fault. Also, there were fewer cases where the number of iterations to converge decreased due to faults.

All of the experiments were also performed with a variant of this perturbation-based fault model that increased the l^2-norm of the operation result. In all instances, smaller fault sizes caused FGMRES to fail to converge compared with the other l^2-norm variants of the fault model, and, for the cases in which the iterative solver converged, many more iterations were required. Due to space considerations, the experiments with the increasing l^2-norm are not detailed in this paper.

Fault Size	Fault Location	Starting Its	l^2-norm Effect	PC	Improvement
5×10^{-5}	matvec	0% - 30%	Varied	ARMS	2% - 4%
10^{-6}	matvec	(anywhere)	Varied	ARMS	0% - 1%
$10^{-6} - 10^{-5}$	matvec	(anywhere)	Varied	ILUT	0% - 1%
(any size)	matvec	0% - 60%	Decreasing	ARMS, ILUT	0% - 1%
$\leq 5 \times 10^{-5}$	PC	(anywhere)	Decreasing	ARMS	0% - 5%
$\leq 5 \times 10^{-6}$	PC	(anywhere)	Decreasing	ILUT	0% - 2%

Table 2: Summary of Beneficial Results- Note: Its (Iteration), PC (Preconditioner)

5. Summary and Future Work

This paper showcased experiments designed to exhibit a persistent fault model with faults affecting bounds within an iterative solver, which may be monitored and play a role in the solver reaction to faults. Specifically, effects on the l^2-norm of the fault-perturbed vector were explored and it was found that persistent faults may be treated similarly to episodic faults in quantifying their effects except that the application possibly needs to adjust to continuing operation "under failure". An investigation of such adaptations is left as a future work. In particular, persistent faults that shrink the l^2-norm have less of a negative effect upon the convergence of the iterative solver. It was also found that injecting faults into the outer matvec operation, in general, had a greater impact upon the FGMRES convergence than doing so for the preconditioner application—including causing more cases in which the iterative solver failed—which was observed for both the ARMS and ILUT preconditioners. It appears that runs using ARMS preconditioner are more naturally resilient to the injection of persistent perturbation-based faults than runs using the ILUT preconditioner; regardless of which of the two fault sites is chosen. In addition, a small fault injection resulted in several runs that converged in up to 5% fewer iterations than would be typically required. Table 2 summarizes beneficial outcomes from the results presented in this paper. In the future, it is planned to use this summary as a guide in devising algorithm based fault tolerance procedures for the flexible GMRES.

5.1 Acknowledgments

This work was supported in part by the Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476, by the National Science Foundation grant 1516096, by the U.S. Department of Energy (DOE), Office of Advanced Scientific Computing Research, through the Ames Laboratory, operated by Iowa State University under contract No. DE-AC02-07CH11358, and by the U.S. Department of Defense High Performance Computing Modernization Program, through a HASI grant. This work used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] K Asanovic, R Bodik, BC Catanzaro, JJ Gebis, P Husbands, K Keutzer, DA Patterson, WL Plishker, J Shalf, SW Williams, et al. The landscape of parallel computing research: A view from berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[2] FA Bower, DJ Sorin, and S Ozev. Online diagnosis of hard faults in microprocessors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(2):8, 2007.

[3] G Bronevetsky and B de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 155–164. ACM, 2008.

[4] F Cappello, A Geist, W Gropp, S Kale, B Kramer, and M Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.

[5] A Edelman. The mathematics of the Pentium division bug. *SIAM review*, 39(1):54–67, 1997.

[6] J Elliott, M Hoemmen, and F Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1193–1202. IEEE, 2014.

[7] J Elliott, M Hoemmen, and F Mueller. Tolerating Silent Data Corruption in Opaque Preconditioners. *arXiv preprint arXiv:1404.5552*, 2014.

[8] J Elliott, M Hoemmen, and F Mueller. A numerical soft fault model for iterative linear solvers. In *Proceedings of the 24nd International Symposium on High-Performance Parallel and Distributed Computing*, 2015.

[9] J Elliott, F Mueller, M Stoyanov, and C Webster. Quantifying the impact of single bit flips on floating point arithmetic. *preprint*, 2013.

[10] M Hoemmen and MA 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*, volume 3, page 9. Citeseer, 2011.

[11] ML Li, P Ramachandran, SK Sahoo, SV Adve, VS Adve, and Y Zhou. Trace-based microarchitecture-level diagnosis of permanent hardware faults. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 22–31. IEEE, 2008.

[12] Z Li, Y Saad, and M Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical linear algebra with applications*, 10(5-6):485–509, 2003.

[13] Y Saad and B Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. *Numerical linear algebra with applications*, 9(5):359–378, 2002.

[14] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[15] B Schroeder, E Pinheiro, and WD Weber. DRAM errors in the wild: a large-scale field study. In *ACM SIGMETRICS Performance Evaluation Review*, volume 37, pages 193–204. ACM, 2009.

[16] M Snir, RW Wisniewski, JA Abraham, SV Adve, S Bagchi, P Balaji, J Belak, P Bose, F Cappello, B Carlson, et al. Addressing failures in exascale computing. *International Journal of High Performance Computing Applications*, page 1094342014522573, 2014.