A Comparison and Analysis of Soft-Fault Error Models using FGMRES

Evan Coleman
NSWC - Dahlgren Division, Old Dominion University
ecole028@odu.edu

Masha Sosonkina
Old Dominion University
msosonki@odu.edu

Keywords: fault tolerance, soft fault models, FGMRES, iterative linear solvers, resilient computing

Abstract

A comparison of the impact of soft fault errors - as given by two distinct soft fault error models - on the GMRES iterative method with flexible preconditioning (called FGMRES) is performed. The effect of the two soft fault error models is evaluated on the convergence of FGMRES in solving 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 conducted in this study quantify the difference in soft fault error modeling approaches and provide a means to compare the potential impact of soft fault errors of various types.

1. INTRODUCTION

Research into fault tolerant methods is a natural means to protect against the appearance of faults inside of computing programs. The prevalence of faults is expected to increase as high-performance computing (HPC) platforms continue to move towards larger and larger computing environments ([4, 1]). As the typical HPC environment moves closer to the exascale level of performance, the mean time between failures (MTBF) will continue to decrease dramatically. As the occurrence of faults becomes increasingly more common, the software used in HPC environments will become required to provide some means of fault tolerance in order to deal with the faults that are experienced. When examining the need for fault tolerance, one must look at what type of occurrence is traditionally associated with the word "fault". In most of the research into fault tolerance, faults are typically divided into two distinct categories: hard faults and soft faults ([9, 12]).

Hard faults usually come about due to negative effects on the physical hardware components of the system; the key characteristic of all hard faults is that they cause program interruption. As such, they are difficult to deal with from an algorithmic standpoint. However, 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 notable for the fact that they do not immediately cause program interruption, although program interruption can occur as a result of the damage caused by a soft fault. Another key feature of soft faults, is that they are able to be detected (though the cost for detection may be prohibitively high) during program execution. Most of the time, soft faults allude to some type of data corruption.

The study of soft faults is the focus of this paper. Soft faults are typically divided into various categories based on how long their effect is felt by the resident program. For a long time, the impact of soft faults was measured by the injection of bit flips into the data structures that are used by the algorithm in question. However, recent research efforts (e.g., [8, 9, 10, 6]) have focused on modeling the impact of soft faults with a slightly more generalized numerical approach that quantifies the potential impact of a bit flip – which is dependent on where the bit flip occurs – and generating an appropriately sized fault using a more numerically-based scheme. The experiments conducted in this paper seek to adapt two existing, generalized, numerical soft fault models to study a particular class of soft faults faults ("sticky" faults) that has not been examined extensively in the past.

The remaining sections of the paper are organized as follows: in section 2., the motivation for both the adapted fault models and the study itself is presented, and a brief overview of related studies is provided as well as general motivation for the study of soft faults, in section 3., background information for both the FGMRES algorithm and the preconditioners used in this experiment is provided, in section 4., details concerning each of the fault models that is used throughout this work are given, in section 5., experiment results from each of the fault models are provided, along with an analysis of the similarities and differences that appear between the results from each of the two models, in section 6., a quick summary is presented along with a few possible directions for future work. The contributions found in this work include the following:

- An extension to the fault model presented by Elliot, Hoemmen, and Mueller [8, 9, 10] is provided

- A modification to the fault model presented by Coleman and Sosonkina in [6] is detailed
- A comparison and analysis of the differences between each of the adapted models is given, focusing on how each of the two fault models predicts that a sticky fault will effect an iterative linear solver – specifically, the iterative solver, FGMRES.

2. MOTIVATION AND RELATED WORK

As the easiest instance of a soft fault to understand occurs in the form of a bit flip, researchers have typically relied upon the direct injection of bit flips into their routines in order to simulate the occurrence of a soft fault [3, 11]. On the other hand, in the work by Elliot, Hoemmen, and Mueller [8, 9, 10], faults are modeled in a more general sense. In this approach, during the injection of a fault into the result of a specific important operation inside of the algorithm – in the case of an iterative solver (i.e. FGMRES) this could be a sparse matrix-vector multiply or in the application of the preconditioner – instead of flipping a bit inside of the resultant data structure, a study ([11]) was conducted to quantify the impact of a single bit flip and this analysis was used in [8] in order to create a numerical soft fault model that injects a fault in a more general sense. This fault injection methodology is described fully in section 4.1.

In the classification of soft faults that is presented in [9, 12], soft faults are divided into the following three categories based upon how they effect (and continue to effect) program execution: 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. Note that a fault in any of these three classifications should be detectable during program execution, and as such, it should be possible to ameliorate the negative impact caused by the fault.

Scenarios that could cause a persistent fault include a stuck bit in memory, or the Intel Pentium FDIV bug [9, 7]. 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 [13, 2] and memory based faults [19]. An example of a situation that can create a sticky fault, that is provided in [12], is the incorrect copy of data from one location to another. The incorrect bit pattern present in the faulty copy of the data will remain incorrect for an indefinite amount of time, but will be corrected if and when the data is copied over again – assuming that the later copy routine executes correctly. It is also important to note that in the case of a sticky fault, the fault can be corrected by means of a direct action. Transient errors are typically caused by solitary bit flips, which may be caused by different issues (e.g. radiation, hardware malfunction, data cache set incorrectly, etc).

Whether research chooses to model faults using bit flips or adopt a more numerical approach, much of the previous work on the impact of silent data corruption (SDC) has to do with the modeling of transient errors. The goal of the study that is detailed here is to adapt both a numerical soft fault model for transient soft faults and a perturbation based soft fault model for persistent soft faults so that each of the two models is capable of modeling the potential impact of a sticky fault.

3. BACKGROUND

In this section, a brief background of both the GMRES algorithm with flexible preconditioning (FGMRES) is given, and then an overview of the two preconditioners that were focused on in this study is provided.

3.1. FGMRES

The FGMRES algorithm, as described in [15], 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 11). FGMRES was selected in this study because it is a robust, popular iterative solver which is proven to converge under variable preconditioning - including converging in situations where the variable preconditioning comes as a result of some sort of perturbation or anomaly in the preconditioning operation. In this study specifically, such a perturbation is due to injected faults via one of the two fault models that is used in the experiments.

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

Algorithm 1: FGMRES as given in [15]

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 4, which is the application of the preconditioner.

3.2. Preconditioners

A transformed *preconditioned system* writes the general linear system of equations, $Ax = b$, in the form $M^{-1}Ax = M^{-1}b$, 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 write in the form $M = \bar{L}\bar{U}$, 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. In the ILUT preconditioner used in these experiments, a dual non-zero threshold (τ, ρ) is used where all computed values that are smaller than $\tau||a_i||_2$ are dropped - where $||a_i||_2$ is the l^2-norm of a given row of the matrix A, and only the largest ρ elements of each row are kept.

In a given linear system, if m of the independent unknowns are numbered first, and the other $n-m$ unknowns last, the coefficient matrix of the system is permuted in the resulting 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, and $A_1 = C - ED^{-1}F$ is the Schur complement with respect to C, I_{n-m} is the identity matrix of dimension $n-m$, denoted 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 above defines a general framework which accommodates for different methods. The Algebraic Recursive Multilevel Solver (ARMS) preconditioner [18] 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 ILU using dual nonzero thresholds (ILUT) [16]. 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.

4. FAULT MODELS

In this section, a description of each of the fault models that are included in the comparison is provided. Particular note is made to distinguish each of the models from one another. As noted earlier, the two main sticky fault models that were utilized in this study were an adapted version of the numerical soft model presented in [8, 9, 10] - termed "Numerical Soft Fault Model" (NSFM) due to the origins of this model in seeking a numerical estimation of a fault (rather than modeling faulty behavior directly), and an adapted version of the model given in [6] - which will be referred to as the "Perturbation Based Soft Fault Model" (PBSFM) due to its modeling of faults as small random perturbations. A fuller description of each of the two soft fault models follows in the next two subsections.

4.1. Numerical Soft Fault Model

The approach given by detailed in [8, 9, 10] 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, 9, 10] 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,
- a permutation of the components of the vector result of the preconditioner application for the MPI process in which a fault was injected,

or a combination of either of these two effects. Note that if α is the scaling factor used, and if x is the original vector and if $\hat{x}$ is the vector with a fault injected into it that we have three scenarios:

1. $\alpha = 1$: $||x||_2 = ||\hat{x}||_2$
2. $0 \leq \alpha < 1$: $||x||_2 > ||\hat{x}||_2$
3. $\alpha > 1$: $||x||_2 < ||\hat{x}||_2$

The adaptation that was made to extend this model to be applicable in a "sticky" sense was to inject a fault into a single MPI process in the exact same manner for every iteration that occurred while the persistent fault was being simulated to have occurred. That is, if a sticky fault was determined to be in effect for the first 100 iterations of a run of the iterative solver, on *each* iteration.

The analysis that was performed in [8, 9, 10] details the impact of the NSFM in the case where it is modeling transient soft faults with various scaling values. The impact of this fault model relative to the impact of a single bit flip is given in [8] and shows that regardless of where the bit flip occurs, the NSFM will perform comparably to the worst case scenario induced by a traditional bit flip. Analysis to show the impact of a bit flip based on where in the storage of a floating point number it occurs is given by [11].

4.2. Perturbation Based Soft Fault Model

The approach proposed in [6] is similar in spirit to the NSFM proposed above. It selects a single MPI process and injects a small random perturbation into each element of the vector. That is, if the vector to be perturbed is x and the size of the perturbation based fault is ε than to inject a fault, then generate a random number in the range $r_\varepsilon \in (-\varepsilon, \varepsilon)$ and set $x_i = x_i + r_\varepsilon$ for all i. The resultant vector, call it $\hat{x}$, is thus perturbed away from the original vector x.

Since the FGMRES algorithm works at minimizing the l^2 norm of the residual, and this can be directly effected by the l^2 norm of certain steps inside of the FGMRES algorithm, there are three variants to the PBSFM:

1. The sign of x_i is not taken into account. In this variant, $||x||_2 \approx ||\hat{x}||_2$.
2. If $x_i \geq 0$ then $r_\varepsilon \in (-\varepsilon, 0)$ and if $x_i < 0$ then $r_\varepsilon \in (0, \varepsilon)$. Here, $||x||_2 \geq ||\hat{x}||_2$.
3. If $x_i \leq 0$ then $r_\varepsilon \in (-\varepsilon, 0)$ and if $x_i > 0$ then $r_\varepsilon \in (0, \varepsilon)$. Here, $||x||_2 \leq ||\hat{x}||_2$.

Using these three variants allows the PBSFM to possess some level over the l^2 norm of the vector that it is injecting a fault into, and therefore an added level of control on how a fault may affect the convergence of the FGMRES algorithm.

4.3. Comparison of Soft Fault Models

In looking to see which of the two fault models induces a "larger" fault, then in general it will be the case that the NSFM will create a larger difference between a given data structure with a fault injected and the same data structure in a fault free environment.

Examining this for the test problem (section 5.1.) used in these experiments, the result of the outer matvec operation is a zero vector initially and as FGMRES progresses closer to the solution, this vector will begin to approach the original right hand side of the equation, b. In this problem, the entries in the final iterates of Ax_i before convergence will have entries, b_i, where $-0.01 \leq b_i \leq 0.01$ forms a loose bound on all entries of $b = Ax_i$. To show the potential difference in magnitude between a given vector b and a vector $\hat{b}$ representing the vector b with a fault injected, 10000 random vectors were generated in MATLAB for vectors b of varying sizes (to represent varying problem sizes) and the l^2 norm of $|b - \hat{b}|$ was calculated for each of the two fault models. These results are shown in table 1.

Additionally, the NSFM allows slightly more exact statements to be made concerning the effect of the injected fault on the l^2-norm, as the l^2-norm will be the exact same for all but the effected subdomains, where the l^2-norm of that section is controlled explicitly. However, the size of the fault – measured as a difference from a fault free run – is in general dependent only on problem size in the case of the NSFM. On the other hand, statements concerning the l^2-norm are inherently less exact when the PBSFM is used, as the l^2-norm of the faulty subdomain is not precisely controlled, but the difference from a fault free run - i.e. the "size" of the fault - is easier to control by way of simply adjusting the bounds on the perturbation that is used.

Vector Size	$\|\|b-\hat{b}\|\|^2$ - NSFM	$\|\|b-\hat{b}\|\|^2$ - PBSFM
10	2.2223	9.0351e-04
100	5.1826	0.0029
1,000	17.1997	0.0091
10,000	53.8458	0.0289
100,000	172.3676	0.0913
1,000,000	543.9308	0.2887

Table 1. Difference in the effect of each of the fault models on random vectors with values similar to those found in the result of the outer matvec operation. Note: The scaling factor in the NSFM was set to 1.0 and the fault size in the PBSFM was set to 5×10^{-4}. Columns 2 and 3 represent average differences over 10,000 runs.

In general, one of a limited number of outcomes is most likely to occur when a fault occurs during the execution of an iterative solver ([8, 3]).

- The solver will converge in the approximately the same number of iterations, with an error in the final solution.
- The solver will converge in the approximately the same number of iterations, with no error in the final solution.
- The solver will converge in more iterations than in a fault free run; with or without an error in the final solution.
- The progress of the solver towards the solution will stagnate, and it will fail to converge.

5. RESULTS AND ANALYSIS

In this section, the results of the experiments that were conducted are given; the results are presented as a comparison of the effects of a sticky soft fault as modeled by both the PBSFM and the NSFM. First, a brief summary of both the test problem and the test environment that were utilized in this experiment is given.

5.1. Test Problem

The test problem that was used comes directly from the pARMS library [14], and represents the discretization of the following elliptic 2D partial differential equation,

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

on a square region with Dirichlet boundary conditions, using a five-point centered finite-difference scheme on an $n_x \times n_y$ 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 corresponded to a "small" problem of $n_x = n_y = 200$ and a "large" problem variant with $n_x = n_y = 400$. Both of these two problem sizes were run on a $p_x = p_y = 20$ grid of 400 total processors. This leads to problem sizes of,

- Small: $n = p_x \times p_y \times n_x \times n_y = 64,000,000$
- Large: $n = p_x \times p_y \times n_x \times n_y = 16,000,000$

As pointed out in section 4.3. the NSFM creates larger faults in some sense for larger problem sizes, whereas the size of the fault injected by the PBSFM scales much more evenly with problem size.

5.2. Test Environment

The test environment that was used was the Turing Cluster found at Old Dominion University. The Turing Cluster features a grand total of 22,880 processor cores and 3,060 GB of memory spread across a grand total of 163 nodes. For the purposes of these experiments, all of the experiments were conducted on a subset that contained 400 cores.

5.3. Experiment Description

All of the trials were run on both the small and large problem sizes detailed above in section 5.1. in three sets of scenarios: a fault-free run, a series of runs using the NSFM and a series of runs using the PBSFM. For the NSFM, the variable that will have the largest impact upon the fault injected is the scaling factor, α, while for the PBSFM the largest contributor to the impact of the fault is the size of the perturbation that is added, ε. For these experiments, three values of both α and ε were used: $\alpha = 1/2, 1, 2$, and $\varepsilon = 1e-3, 5e-4, 1e-4$.

All three variants of the PBSFM were utilized. To compare with the runs of the NSFM using $\alpha = 1/2$, the variant of the PBSFM that minimizes l^2 norm was used, to compare with the runs of the NSFM using $\alpha = 1$ the version of the PBSFM that leaves the l^2 norm approximately the same was used, and to compare with the version of the NSFM using $\alpha = 2$ the form of the PBSFM that maximizes the l^2 norm was used.

As sticky soft faults are defined as being present for a certain duration and then being naturally corrected at some point during program execution, sticky faults were defined to be present during the first 1000 iterations of the iterative solvers execution. The number of iterations required for convergence in a fault-free environment is a factor of many variables the small problem used here converged in roughly 1500 iterations in a fault-free environment, and the large problem converged in approximately 3500 iterations for the set of parameters that were used in this study. Determining more exact bounds for the start and stop times for the injection of a sticky fault based on analysis of potential causes of sticky faults is a direction for future work. Lastly, as both fault models that were included in these experiments have an element of randomness all runs of FGMRES were performed multiple times and an average of the results that were found was taken.

5.4. Results

Throughout this section, the effects on the ARMS preconditioner will be provided in blue, while the effects on the ILUT preconditioner will be given in red. Due to space considerations, plots are only presented for the neutral l^2-norm variants of the fault models in figs. 1 to 4, while extended, full results are provided in tables 2 and 3 for all other experiments. Each figure shows five different fault methods: a nominal (fault-free) run, a PBSFM run with a "small" fault (1e-4), a PBSFM run with a "medium" fault (5e-4), and a PBSFM with a "large" fault (1e-3).

The first plots that are shown, in fig. 1, depict the effects of the various soft fault model combinations and the effects they have on the small version of the problem in the case of the neutral (with respect to the l^2-norm) variants of the model. To be specific, this involves the variants of the PBSFM where the faults are centered about 0, and the version of the NSFM where the scaling factor, α, is set to 1.

In these images it is apparent that, for the neutral variants of the soft fault models, for both the ARMS and ILUT preconditioners, the NSFM has a more negative effect on the convergence of the FGMRES algorithm than the PBSFM - at least for all of the parameter combinations that were considered.

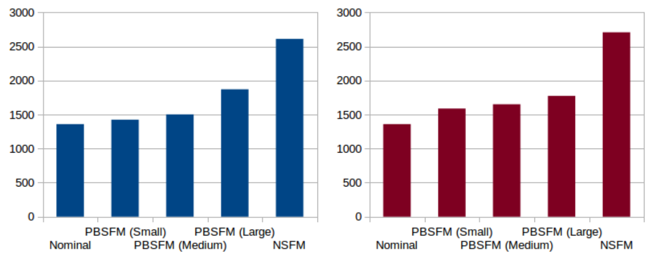

Figure 1. Soft fault comparison on total number of iterations for the small problem for faults injected at the outer matvec operation. ARMS preconditioner on the left (blue), ILUT preconditioner on the right (red). Fault methods are displayed along the x-axis and total iterations required for convergence are represented by the y-axis.

Next, in fig. 2, the results for both soft fault models are shown for the small problem in the case where the faults are injected into the second fault location, the result of the application of the preconditioner. Again, the plots that are shown for this set of results represent the neutral l^2-norm variants.

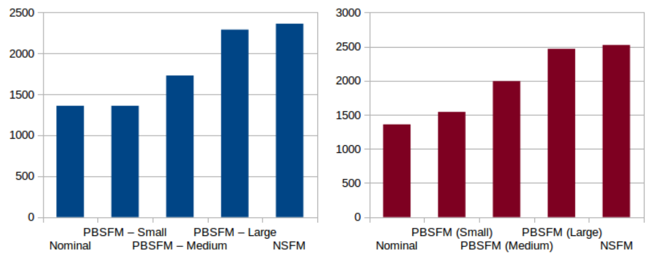

Figure 2. Soft fault comparison on total number of iterations for the small problem for faults injected at the application of the preconditioner. ARMS preconditioner on the left (blue), ILUT preconditioner on the right (red). Fault methods are displayed along the x-axis and total iterations required for convergence are represented by the y-axis.

All of the remaining results for the experiments that were conducted on the "small" problem are provided in table 2. Note that the results in table 2 are again color coded so that the blue lines represent the results associated with the ARMS preconditioner, and the red lines represent the results corresponding to the ILUT preconditioner.

l^2	PBSFM (S)	PBSFM (M)	PBSFM (L)	NSFM
=	1424	1501	1870	2610
–	2238	2241	2232	2541
+	1605	1611	1616	2596
=	1359	1728	2289	2361
–	2314	2258	2291	2315
+	2173	2360	2180	2380
=	1587	1651	1773	2707
–	2346	2342	2232	2737
+	1870	1859	1875	2749
=	1542	1993	2466	2522
–	2366	2372	2403	2539
+	2312	2400	2298	2570

Table 2. Full results for the small problem for faults injected into the outer matvec operation (above the line) and the preconditioning operation (below the line) with the ARMS preconditioner (blue) and ILUT preconditioner (red). "=" represents the neutral l^2-norm variant of all models, "–" represents the model variants that decrease l^2-norm, and "+" represents the versions of the model that increase l^2-norm.

In the next set of images, shown in fig. 3, results are given for the simulated injection of faults into the outer matvec operation on the large problem size. As before, the plots reflect the versions of the fault models that are designed to not effect the l^2 norm. As in fig. 1, the results shown in fig. 3 show a steady increase in the delay in the convergence of the FGMRES iterative solver from the nominal case, to the PBSFM (ordered by the increasingly sized faults), to the faults simulated by the NSFM.

l^2	PBSFM (S)	PBSFM (M)	PBSFM (L)	NSFM
=	3246	3820	3974	4787
–	3583	3580	3642	4755
+	4066	4057	4055	4764
=	3096	4014	4163	3921
–	4207	4159	3939	3957
+	4374	4079	4367	4053
=	3756	4610	4739	5526
–	4232	4139	4828	5500
+	4836	4827	4828	5502
=	3452	4781	4707	4561
–	4833	4549	4733	4554
+	4990	4538	5007	4540

Table 3. Full results for the large problem for faults injected into the outer matvec operation (above the line) and the preconditioning operation (below the line) with the ARMS preconditioner (blue) and ILUT preconditioner (red). "=" represents the neutral l^2-norm variant of all models, "–" represents the model variants that decrease l^2-norm, and "+" represents the versions of the model that increase l^2-norm.

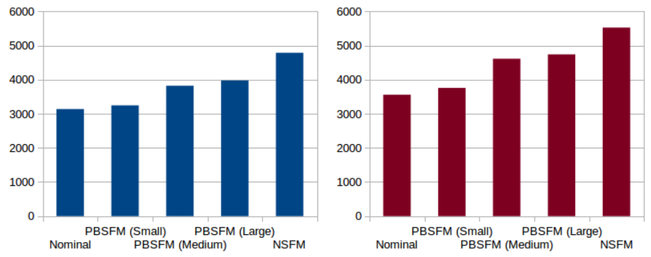

Figure 3. Soft fault comparison on total number of iterations for the large problem for faults injected at the outer matvec operation. ARMS preconditioner on the left (blue), ILUT preconditioner on the right (red). Fault methods are displayed along the x-axis and total iterations required for convergence are represented by the y-axis.

In the last set of plots, shown in fig. 4, results are given for the injection of faults into the result of the preconditioning operation for the large problem size using the neutral l^2-norm variants of both of the soft fault models. These results show the one instance where the "large" fault size associated with the PBSFM ($1e-3$) causes a larger delay in the convergence of the FGMRES solver than the corresponding runs of the NSFM. This effect was seen for all l^2-norm variants of the PBSFM, although only the neutral variant is shown in fig. 4. The details of the results for all of the other experiments that were run on the large problem are given in table 3. As in table 2, the results are colored so that the runs with the ARMS preconditioner are marked with blue and the runs with the

ILUT preconditioner are denoted with red in table 3.

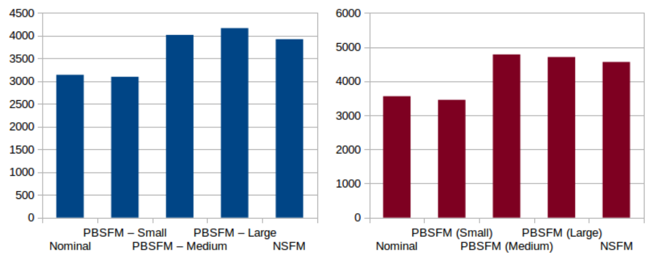

Figure 4. Soft fault comparison on total number of iterations for the large problem for faults injected at the application of the preconditioner. ARMS preconditioner on the left (blue), ILUT preconditioner on the right (red). Fault methods are displayed along the x-axis and total iterations required for convergence are represented by the y-axis.

5.5. Comparison and Analysis

In a fault-free environment the use of the ARMS preconditioner caused the FGMRES algorithm to converge in fewer iterations than the use of the ILUT preconditioner did. This remained true when faults were injected into the application of the preconditioner, but the injection of faults into the outer matvec operation caused FGMRES to converge in roughly the same number of iterations whether it was preconditioned with ILUT or with ARMS. This suggests that for faults occurring at the outer matvec operation, the advantage of the ARMS preconditioner is not as present as it is elsewhere.

Next, faults injected into the outer matvec operation had a larger impact than identical faults injected into the result of the application of the preconditioner. This was seen in runs using both the ILUT and the ARMS preconditioners. Similar results were seen in [6]. In addition, the impact of the faults injected by each of the two soft fault models studied on the l^2-norm seems to be more pronounced in the PBSFM; although, this is clearly adjustable through the use of the parameters available to both soft fault models and using larger values for α in the NSFM may provide a better comparison.

When comparing the two fault models presented here directly, it is evident that the NSFM has a larger negative impact on the convergence of the iterative FGMRES than the PBSFM in most scenarios. In every instance tested except for preconditioner faults on the larger problem size, the comparable version of the NSFM delayed convergence longer than the PBSFM. This is in part due to the fact that the NSFM moves the vector that it is injecting a fault into much farther from its original location than the PBSFM (see section 4.3.). Also worth noting is that in all instances, the FGMRES algorithm was able to converge successfully to the correct solution despite the presence of a sticky fault for the first 1,000 iterations of execution.

For recurring faults specifically, the PBSFM offers a greater level of fine-tuned control over the impact of the fault, as the size of the fault that is injected on each iteration. However, the size of the fault in the PBSFM does not seem to have as large of an impact on the convergence of FGMRES on the runs that attempted to manipulate the l^2-norm. Also, due in part to the results in section 4.3., the expectation was that the NSFM would delay convergence of an iterative solver significantly more than the PBSFM (for the particular model input parameters chosen); while this is true for the case of faults injected into the outer matvec operation, it appears that the PBSFM causes more of a problem with convergence for FGMRES than the NSFM for faults injected into the result of the application of the preconditioner.

6. SUMMARY AND FUTURE WORK

In this paper, the results of a comparison between two preliminary sticky soft fault error models has been given and analyzed. It is hoped that analyses such as this can contribute to the formation of algorithm based fault tolerance (ABFT) techniques to combat the effects of all possible of variants of soft faults in the future. In order to formulate effective ABFT techniques, the first step would be to develop a reliable fault detection mechanism; however, the two fault models explored in this study each present their unique respective challenges in fault detection, and a fault detection designed towards one of the fault models may not necessarily work when applied to faults generated by the other. Using the data from each series of runs to generate the most generalized fault detection technique possible would have the farthest reaching impact and is a direction for future work.

Additionally, it would be beneficial to better quantify the potential impact of both sticky and persistent faults that originate in a real-world environment. Doing so could help make more precise the simulation of both sticky and persistent soft faults, aiding in soft fault simulation and the development of ABFT techniques. It would also be helpful to examine the impact of both of these modified fault models on multiple subdomains; they both define faults to effect a single subdomain (i.e. a single MPI process) at a time, but it is possible that a real world fault could effect multiple subdomains and the impact of this potential event could be explored. For the transient case, this has been explored in [8, 9, 10]. It may also prove helpful to consider a wider range of scaling factors for the NSFM, as well as a wider range of fault sizes for the PBSFM in order to cover a larger spectrum of potential impacts due to the presence of faults.

6.1. Acknowledgements

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 grants 0904782, 1047772, 1516096, by the U.S. Department of Energy, 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 research was also supported in part by the Turing High Performance Computing cluster at Old Dominion University.

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] E Chow and Y Saad. Experimental study of ILU preconditioners for indefinite matrices. *Journal of Computational and Applied Mathematics*, 86(2):387–414, 1997.

[6] E Coleman and M Sosonkina. Evaluating a perturbation-based soft fault model on preconditioned iterative methods. *Independent Study Report - Old Dominion University*, 2016.

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

[8] J Elliott, M Hoemmen, and F Mueller. A numerical soft fault model for iterative linear solvers.

[9] 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.

[10] J Elliott, M Hoemmen, and F Mueller. Tolerating silent data corruption in opaque preconditioners. *arXiv preprint arXiv:1404.5552*, 2014.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] Y Saad. ILUT: A dual threshold incomplete lu factorization. *Numerical linear algebra with applications*, 1(4):387–402, 1994.

[17] Y Saad and MH Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[18] 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.

[19] 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.