A Comparison of Soft-Fault Error Models in the Parallel Preconditioned Flexible GMRES*

Evan Coleman [1,2] (ecole028@odu.edu), Aygul Jamal[3] (jamal@lri.fr), Marc Baboulin[3] (baboulin@lri.fr), Amal Khabou[3] (khabou@lri.fr), and Masha Sosonkina [2] (msosonki@odu.edu)

[1] Naval Surface Warfare Center - Dahlgren Division, Dahlgren, VA, United States
[2] Old Dominion University, Norfolk, VA, United States
[3] Université Paris-Sud, Université Paris-Saclay; 91405 Orsay, France

Abstract.
The effect of two soft fault error models on the convergence of the parallel flexible GMRES (FGMRES) iterative method solving an elliptical PDE problem on a regular grid is evaluated. We consider two types of preconditioners: an incomplete LU factorization with dual threshold (ILUT), and an algebraic recursive multilevel solver (ARMS) combined with random butterfly transformation (RBT). The experiments quantify the difference between two soft fault error models considered in this study and compare their potential impact on the convergence.

Keywords: fault tolerance, soft fault models, FGMRES, parallel iterative linear solvers, preconditioners, ARMS, ILUT, RBT randomization

1 Introduction

The prevalence of faults is expected to increase as high-performance computing platforms continue to grow [1,6] and the mean time between failures (MTBF) continues to decrease, which calls for the design of fault-tolerant computational algorithms that are robust, in the sense of being able to cope with errors. Mostly, faults are divided into two categories: hard faults and soft faults [4,9]. Hard faults are usually due to negative effects on the physical components of the system; their key characteristic is that they cause program interruption. Thus, they are

*This work was supported in part by the Air Force Office of Scientific Research under the AFOSR award FA9550-12-1-0476 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, and the ILIR/IAR program at NSWC Dahlgren. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 and of Old Dominion University operating the Turing High Performance Computing Cluster.

difficult to deal with from an algorithmic standpoint. Conversely soft faults do not immediately cause program interruption, although such an interruption may occur. Another key feature of soft faults is that they can be detected during the program execution. Typically, soft faults allude to some data corruption. The occurrence of soft faults is commonly modeled by injection of bit flips into the algorithm data structures [5,12]. Recent research efforts (see, e.g., [7,8,9,10,11]) have focused on modeling the impact of soft faults with a numerical approach that quantifies the potential impact by generating an appropriately sized faults. It is important to develop and study such models because they provide a means of simulating the data corruption caused by faults, and thus, enable to develop fault resilient algorithms without making any assumptions concerning how a fault may manifest on either current or future hardware. Note that the data corruption caused by the simulation of a fault is not expected to mirror exactly the data corruption that would be caused by the error (e.g., bit flip), but that the impact on the algorithm should be the same. In the case of parallel iterative methods this impact may be judged by the resulting extra iterations. Note also that these numerical soft fault models allow one to model the worst-case behavior by adjusting internal parameters. Moreover, stochastically sampling a particular type of error, such as a bit flip, will tend to reveal an average-case behavior. See [13] for a more detailed description of the numerical approach to simulating faults. This paper aims at adapting two existing numerical soft fault models to study a particular class of soft faults, referred to as *sticky faults*. In the classification of soft faults that is presented in [4,9], soft faults are divided into three categories based upon how they affect the 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 refer to permanent faults.

An example of a sticky fault, that is provided in [4], 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. 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. Whether researchers choose 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 presented in this paper 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 one is capable of modeling the potential impact of a sticky fault. Specifically, the main contributions of this work include (1) an extension to the fault model presented by Elliot *et al.* in [9,10,11], (2) a modification of the fault model proposed by Coleman *et al.* in [7], and (3) an analysis of the differences between the two models.

The remaining sections of the paper are organized as follows: in Section 2, we give some background information for both the FGMRES algorithm and the preconditioners used for the experiments, in Section 3, we detail the two fault models adapted here, in Section 4, we present experiment results, and in Section 5, we conclude.

2 Background

2.1 Preconditioners

A *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 effective preconditioning techniques for solving linear systems. In this case, the matrix M has the form $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are approximations to the L and U factors of the standard LU decomposition of A. The incomplete factorization may be computed using the Gaussian elimination algorithm, by discarding some entries in the L and U factors. In the ILUT preconditioner used in the experiments, a dual non-zero threshold (τ, ρ) is used: all computed values that are smaller than $\tau||a_i||_2$ are dropped, where $||a_i||_2$ is the norm of a given row of the matrix A, and only the largest ρ elements of each row are kept. Note that throughout the paper, the Euclidean norm is used.

For 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 a 2×2 block structure.

In multi-elimination methods [16, p. 392], a reduced system is recursively constructed from the permuted system performing a block LU factorization of PAP^T as follows

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

where P is a permutation matrix, D is a diagonal matrix (or block-diagonal if we consider sets of independent unknowns), 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 $n - m$, $G = EU^{-1}$ and $W = L^{-1}F$. The reduction process can be applied another time to 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 can accommodate for different methods. The Algebraic Recursive Multilevel Solver (ARMS) preconditioner [17] uses block independent sets to discover sets of independent unknowns and computes them by using a 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 non-zero 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.

In this study, we also use an implementation of ARMS called ARMS_RBT [3] where the last Schur complement system is small enough to be converted into a dense matrix and randomized using Random Butterfly Transformations [2] to avoid pivoting in the Gaussian elimination. Then the resulting system is solved via a routine that performs Gaussian elimination with no pivoting, followed by two triangular solves. The ARMS_RBT version has shown satisfactory numerical behavior [3] and can potentially benefit from GPU computing [14]. It appeared also in the experiments conducted in our study that the convergence results with ARMS and ARMS_RBT have been quite similar.

In the remainder of this paper, ARMS_RBT will be simply referred to as "ARMS". In our experiments, we will use the preconditioners ILUT and ARMS_RBT. This choice is motivated by the fact that ultimately, we plan to study soft errors in the pARMS solver [15].

2.2 Flexible GMRES

The right-preconditioned FGMRES algorithm, as described in [16, p. 273] is provided in Algorithm 1. FGMRES is similar in its nature to the standard GMRES with the exception of allowing the preconditioner to change at each iteration by storing the result of each preconditioning operation (cf. matrix Z_m in line 10). In this study, we select FGMRES (instead of GMRES) because it is a robust solver which is proven to converge under variable preconditioning, including converging in situations where the variability comes as a result of some anomaly in the preconditioning operation [7], the anomaly being here the fault injected via the soft error fault models. In our experiments, the faults are injected at two distinct locations inside the FGMRES algorithm: line 1, called here the *outer matvec* operation, and line 3, which is the application of the preconditioner M. These two locations were chosen since they are two of the most computationally demanding operations in the algorithm.

3 Fault Models

As noted earlier, the two main sticky fault models used in this study are: first an adapted version of the model presented in [11], referred to as "Numerical Soft Fault Model" (NSFM) due to its origins in seeking a numerical estimation of a fault, and second an adapted version of the model given in [7], which will be referred to as the "Perturbation Based Soft Fault Model" (PBSFM) due to its modeling of faults as small random perturbations.

Numerical Soft Fault Model. The approach detailed in [11] generalizes the simulation of soft faults by disregarding the actual source of the fault and allowing the fault injector to vary the size of errors. In the experiments conducted in [9,10,11], faults are typically defined as either: (1) a scaling of the contribution of the result of the preconditioner application for the Message Passing Interface (MPI)

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** go to line 1

Algorithm 1: Flexible GMRES algorithm.

process in which a fault was injected, (2) a permutation of the components of the vector result of the preconditioner application for the MPI process in which a fault was injected, or (3) a combination of these two effects. We denote as α the scaling factor used, as x and $\hat{x}$ the vector with and without faults, respectively: $\alpha = 1$: $||x||_2 = ||\hat{x}||_2$, $0 \leq \alpha < 1$: $||x||_2 > ||\hat{x}||_2$ and $\alpha > 1$: $||x||_2 < ||\hat{x}||_2$.

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 at every iteration in which a fault is simulated. The analysis that was performed in [9,10,11] details the impact of the NSFM model 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 [11] and shows that regardless of where the bit flip occurs, the NSFM will perform in a similar way to the worst case scenario induced by a traditional bit flip. Analysis showing the impact of a bit flip based on where in the storage of a floating point number it occurs is given in [12].

Perturbation Based Soft Fault Model. The approach in [7] selects a single MPI process and injects a small random perturbation into each vector element. If

Table 1: Effect of the two fault models on random vectors of several sizes

`Size`	10^1	10^2	10^3	10^4	10^5	10^6
`NSFM`	2.2223	5.1826	17.1997	53.8458	172.3676	543.9308
`PBSFM`	0.0009	0.0029	0.0091	0.0289	0.0913	0.2887

the vector to be perturbed is x and the size of the perturbation based fault is ϵ then to inject a fault, one should generate a random number in the range $r_\epsilon \in (-\epsilon, \epsilon)$ and set $x_i = x_i + r_\epsilon$ for all i. The vector with the fault injected $\hat{x}$, is thus perturbed away from the original vector x. Since the FGMRES algorithm works at minimizing the norm of the residual, and this can be directly affected

by the norm of certain steps inside the FGMRES algorithm, we present three variants of 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 $r_\epsilon \in (-\epsilon, 0)$ and if $x_i < 0$ then $r_\epsilon \in (0, \epsilon)$. Here, $||x||_2 \geq ||\hat{x}||_2$.
3. If $x_i \leq 0$ then $r_\epsilon \in (-\epsilon, 0)$ and if $x_i > 0$ then $r_\epsilon \in (0, \epsilon)$. Here, $||x||_2 \leq ||\hat{x}||_2$.

Using these three variants allows the PBSFM to monitor changes over the norm of the vector where a fault is injected (similarly to the NSFM with the α coefficient) and therefore an additional level of control on how a fault may affect the convergence of the FGMRES algorithm.

To show the potential difference between a given vector x and its instance $\hat{x}$ where a fault is injected, we report in Table 1 the norm of the distance between x and $\hat{x}$ computed for each fault model (see columns `NSFM` with $\alpha = 1.0$ and `PBSFM` $\epsilon = 5 \times 10^{-4}$) when considering $10,000$ random vectors x of varying sizes (column `Size`) generated using MATLAB.

Additionally, the NSFM allows for slightly more exact statements to be made concerning the effect of the injected fault on the norm, as the norm will not be affected by the shuffling of elements and the scaling factor causes a predictable effect; even if it is not applied to all subdomains. However, the *size* of the fault—measured as a difference between fault-free and faulty runs—in general, depends only on the problem size in the case of the NSFM. On the other hand, when using the PBSFM, this fault *size* is easier to control.

4 Experiments

The test problem that was considered here comes from the pARMS library [15], 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 with $r_x = r_y = 200$ and a "large" problem variant with $r_x = r_y = 400$. Both of these two problem sizes were run on a $p_x = p_y = 20$ grid of 400 processors in total. This leads to problem sizes of $16,000,000$ and $64,000,000$, respectively ($n = p_x \times p_y \times r_x \times r_y$). The right hand side was chosen as $b = Ax$ with $x = (1, 1, \ldots, 1)^T$. The initial guess was then set to $(0, 0, \ldots, 0)^T$. This problem was selected in order to provide a comparison to existing work [7,15].

The experiments have been carried out on the computing platform Edison located at NERSC. It is a Cray XC30 with $134,064$ cores and 357 TB memory across a total of 5586 nodes. Each node has two sockets, with a 12-core Intel

"Ivy Bridge" processor at 2.4 GHz per socket. All the experiments in this paper were conducted on a subset of 400 cores.

4.1 Experiment Description

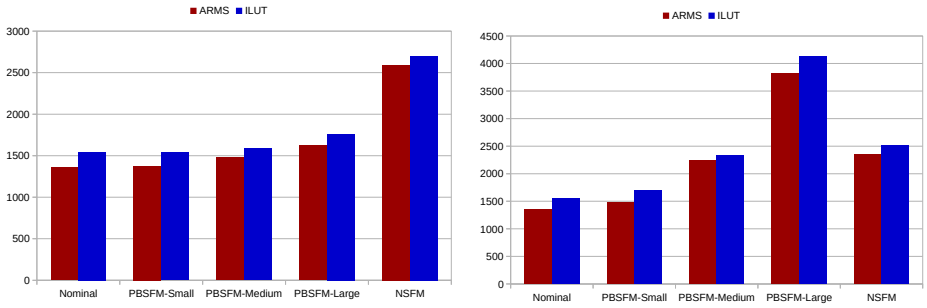
For both the small and large problem, the performed tests included a fault-free run, a series of runs using the NSFM model and a series of runs using the PBSFM model. For the NSFM, the variable that will have the largest impact upon the fault injected is the scaling factor α while for the PBSFM the largest contributor to the impact of the fault is the size of the perturbation ϵ. Sticky faults were conservatively defined to be present during the first 1000 iterations of the iterative solver execution. For the fault-free test, the small problem converged in roughly 1500 iterations, and the large problem in approximately 3500 iterations. For these experiments, three values of both α and ϵ were used: $\alpha = 1/2, 1, 2$, and $\epsilon = 10^{-3}, 5 \cdot 10^{-4}, 10^{-4}$.

The NSFM runs using $\alpha = 1/2$, $\alpha = 1$, and $\alpha = 2$, were compared to the three variants of PBSFM that decreases the norm, that leaves the norm approximately the same (referred to as "neutral" in the remainder), and that increases the norm, respectively (see Section 3). For both fault models, the three values were chosen such that the largest error was close to the largest possible error that allowed convergence for the given problem and fault definition, and the two smaller values were scaled down by reasonable amounts. Note that all the runs of FGMRES were performed multiple times and the average was taken.

4.2 Results

The plots are only presented for the neutral norm variants of the fault models in Figs. 1 and 2. This involves the variants of the PBSFM model where the norm remains approximately the same, and the version of the NSFM where the scaling factor α is set to 1. Each figure shows five different fault methods: a nominal (fault-free) run, a PBSFM run with a "small" fault (10^{-4}), a PBSFM run with a "medium" fault ($5 \cdot 10^{-4}$), a PBSFM with a "large" fault (10^{-3}), and a NSFM run with $\alpha = 1$. Fig. 1a depicts the effects of the various soft faults injected into the outer matrix vector operation of the FGMRES algorithm when solving the small problem. In this figure, it is apparent that, for both the ARMS and ILUT preconditioners, the NSFM has a more negative effect on the convergence of the FGMRES algorithm than the PBSFM. For instance, compared to the fault-free runs, the NSFM runs needed more than 1000 additional iterations to converge for both preconditioners while the additional number of iterations is at most around 150 for the different PBSFM variants. Fig. 1b shows the results when the faults are injected into the vector resulting from the preconditioner application. We observe that the size of errors in PBSFM has more impact when injected in the preconditioner than in the outer matvec operation and that, for large error sizes (10^{-3}), PBSFM requires even more iterations than NSFM.

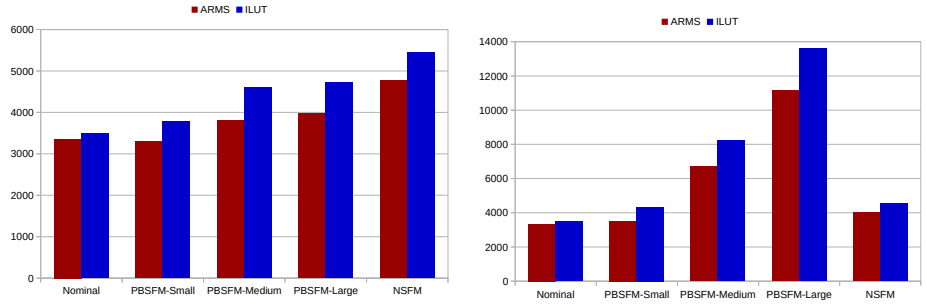
Fig. 2a displays the number of iterations to convergence when injecting faults into the outer matrix vector operation for the large problem. As in Fig. 1a, the



(a) matvec (b) precond

Fig. 1: Soft fault model comparison for the small problem for faults injected at the outer matvec operation (a), and at the application of the preconditioner (b).

results in Fig. 2a show a steady increase in the delay in the convergence of FGMRES from the nominal case to the PBSFM cases (ordered by the increasingly sized faults), then to the NSFM case. The plots in Fig. 2b depict the injection of faults into the result of the preconditioning operation for the large problem. Similarly to Fig. 1b, we observe for PBSFM a higher sensitivity to the size of errors for perturbations of the preconditioner than of the outer matvec operation. Note also that medium and large fault sizes associated with PBSFM cause a larger delay in the convergence than the corresponding NSFM run. For this specific case, the same observation holds for all the three norm variants (cf. Table 2).



(a) matvec (b) precond

Fig. 2: Soft fault model comparison for the large problem for faults injected at the outer matvec operation (a), and at the application of the preconditioner (b).

Complete results, including different PBSFM variants, are provided in Table 2 for the small (`SP`) and large (`LP`) problems with the neutral, decrease, and increase norm variants in rows represented by signs $=$, $-$, and $+$, respectively. The † symbol indicates that the corresponding solver does not converge. We recall that for NSFM, the cases $=$, $-$, and $+$ correspond to $\alpha = 1$, $1/2$, and 2, respectively.

		$\|\|\ \|\|_2$	Nominal		PBSFM-Small		PBSFM-Medium		PBSFM-Large		NSFM	
			SP	LP	SP	LP	SP	LP	SP	LP	SP	LP
ILUT	matvec	=	1542	3496	1380	3300	1477	3797	1624	3969	2590	4768
		−	1542	3496	2236	3807	2318	4170	2352	4380	2565	4660
		+	1542	3496	2241	3603	2326	4140	2358	4386	2637	4788
	precond	=	1542	3496	1487	3523	2243	6703	3811	11156	2355	4022
		−	1542	3496	1499	3280	2155	5163	2782	7639	2324	4093
		+	1542	3496	1499	3518	2168	5162	2780	7735	†	†
ARMS	matvec	=	1359	3357	1538	3790	1585	4594	1764	4727	2698	5456
		−	1359	3357	2323	4199	2426	4810	2459	7639	2697	5375
		+	1359	3357	2339	3825	2423	4655	2459	5059	2646	5426
	precond	=	1359	3357	1700	4349	2336	8221	4125	13607	2518	4550
		−	1359	3357	1706	4010	2201	6063	2925	9492	2570	4493
		+	1359	3357	1657	3989	2205	6061	2927	9005	†	†

Table 2: Full results for the small and large problems with the neutral, decrease, and increase norm.

For a fault-free case, the preconditioning is not variable and we simply run GMRES (see Section 2.2), which converged in fewer iterations when using the ARMS preconditioner compared to the ILUT preconditioner, as already observed in [3]. This remained true when faults were injected. For the large problem, the impact of faults injected is more pronounced for PBSFM, and for ILUT rather than ARMS.

Our experiments showed that the NSFM has a more 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 did. This is in part due to the fact that the NSFM moves the vector where a fault is injected much further from its original location than the PBSFM does (see, e.g., Table 1). In summary, for recurring faults specifically, the PBSFM offers a greater level of fine-tuned control over the fault impacts. However, the size of the fault in the PBSFM does not seem to have as large impact on the convergence of FGMRES in the runs that attempted to manipulate the norm.

5 Conclusion

We compared two soft fault error models, termed as "numerical" (NSFM) and "perturbation based" (PBSFM) that we adapted to simulate sticky faults in the FGMRES algorithm preconditioned with ARMS or ILUT. We injected errors of different sizes when performing the outer matrix-vector operation or when applying the preconditioner. For both models, faults in the preconditioner application lead to slower convergence as compared with errors in the outer matrix-vector multiplication. The experiments also showed that, even in the presence of faults, FGMRES converges in fewer iterations when using the ARMS preconditioner than when using ILUT. These observations indicate that it is advantageous to apply the most robust preconditioner in the environments prone to soft faults.

In the future, we plan to consider a wider range of preconditioners and compare their resilience to soft faults and robustness.

References

1. K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, et al. The landscape of parallel computing research: A view from Berkeley. Technical report, UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.
2. M. Baboulin, J. Dongarra, J. Herrmann, and S. Tomov. Accelerating linear system solutions using randomization techniques. *ACM Trans. Math. Softw.*, 39(2):8:1–8:13, February 2013.
3. M. Baboulin, A. Jamal, and M. Sosonkina. Using random butterfly transformations in parallel Schur complement-based preconditioning. In *2015 Federated Conference on Computer Science and Information Systems*, pages 649–654, 2015.
4. P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen. Fault-tolerant linear solvers via selective reliability. *arXiv preprint arXiv:1206.1390*, 2012.
5. G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proc. of the 22nd annual international conference on Supercomputing*, pages 155–164. ACM, 2008.
6. 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.
7. E. Coleman and M. Sosonkina. Evaluating a Persistent Soft Fault Model on Preconditioned Iterative Methods. In *Proc. of the 22nd annual International Conference on Parallel and Distributed Processing Techniques and Applications*, 2016.
8. E. Coleman, M. Sosonkina, and E. Chow. Fault Tolerant Variants of the Fine-Grained Parallel Incomplete LU Factorization. In *Proc. of the 2017 Spring Simulation Multiconference*. Society for Computer Simulation International, 2017.
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 Int'l*, pages 1193–1202. IEEE, 2014.
10. J. Elliott, M. Hoemmen, and F. Mueller. Tolerating silent data corruption in opaque preconditioners. *arXiv:1404.5552*, 2014.
11. J. Elliott, M. Hoemmen, and F. Mueller. A Numerical Soft Fault Model for Iterative Linear Solvers. In *Proc. of the 24nd Int'l Symposium on High-Performance Parallel and Distributed Computing*, 2015.
12. J. Elliott, F. Mueller, M. Stoyanov, and C. Webster. Quantifying the impact of single bit flips on floating point arithmetic. *preprint*, 2013.
13. James Elliott, Mark Hoemmen, and Frank Mueller. Resilience in numerical methods: a position on fault models and methodologies. *arXiv:1401.3013*, 2014.
14. A. Jamal, M. Baboulin, A. Khabou, and M. Sosonkina. A hybrid CPU/GPU approach for the parallel algebraic recursive multilevel solver parms. In *18th Int'l Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2016, Timisoara, Romania, September 24-27, 2016*, pages 411–416, 2016.
15. 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.
16. Y. Saad. *Iterative methods for sparse linear systems*. Siam, 2003.
17. 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.