To Protect or Not To Protect: Probability-Aware Selective Protection for Sparse Iterative Solvers

Daniel Ryley Johnson*, Hongyang Sun*, Joshua Dennis Booth†, Padma Raghavan‡
* University of Kansas, Lawrence, KS, USA
† University of Alabama in Huntsville, Huntsville, AL, USA
‡ Vanderbilt University, Nashville, TN, USA

***Abstract*—With the increasing scale of high-performance computing (HPC) systems, transient bit-flip errors are now more likely than ever, posing a threat to long-running scientific applications. A substantial portion of these applications involve simulation of partial differential equations (PDEs), modeling physical processes over discretized spatial and temporal domains, with some requiring solving sparse linear systems of equations. While these applications are often paired with system-level application-agnostic resilience techniques, such as checkpointing and replication, using these techniques imposes significant overhead. In this work, we present a probability-aware framework that produces low-overhead selective protection schemes for the widely used Preconditioned Conjugate Gradient (PCG) method, whose performance can heavily degrade due to error propagation through the sparse matrix-vector multiplication (SpMV) operation. Through the use of a straightforward mathematical model and an optimized machine learning model, our selective protection schemes incorporate error probability to protect only certain crucial operations. An experimental evaluation using 15 matrices from the SuiteSparse Matrix Collection demonstrates that our protection schemes effectively reduce resilience overheads, outperforming two baseline and two existing protection schemes across all error probabilities.**

***Index Terms*—Fault tolerance, soft errors, selective protection, iterative solvers, preconditioned conjugate gradient.**

I. Introduction

When a computing system is impacted by a transient bit-flip without any apparent failure, applications may behave erroneously, resulting in incorrect calculations or excessively long runtimes. This phenomenon is called a *soft error*. Soft errors have been attributed to various sources, including radioactive sources such as cosmic rays and packaging materials [3], as well as electrical sources like voltage fluctuations [21] and simple hardware defects.

Large high-performance computing (HPC) systems are particularly susceptible to soft errors due to the sheer amount of electronic components they contain. For this reason, numerous resilience techniques are employed in practice, both at the system level and at the application level. System-level techniques such as checkpointing and replication [4], [5], [14], [17], [25] are widely adopted due to their simplicity and application-agnostic nature. Given their significant overhead, application-specific techniques, or algorithm-based fault tolerance (ABFT) [10], [22], are often employed to mitigate this overhead.

In this work, we introduce a resilience technique for the widely used Preconditioned Conjugate Gradient (PCG) method. PCG solves a linear system $Ax = b$ where A is an $N \times N$ symmetric positive definite (SPD) matrix, b is an $N \times 1$ known vector, and x is the $N \times 1$ solution vector. Previous works have studied the impact of soft errors on PCG's performance [7], [26], [30], which can heavily deteriorate as errors propagate through the sparse matrix-vector multiplication (SpMV) operation. However, certain errors may have no impact on PCG's performance at all. As a result, fully protecting the SpMV operation can lead to suboptimal resilience overhead.

The resilience framework introduced in this work produces probability-aware selective protection schemes designed to protect only crucial operations within SpMV, thereby mitigating performance degradation while minimizing resilience overhead. Our schemes are guided by two static features and one parameter, making decisions with a straightforward mathematical model supplemented by a machine learning model. The operations that we opt to protect are replicated at the system level, and if a soft error is detected between the original and duplicated computations, the previous iteration is repeated.

An experimental evaluation across 15 matrices from the SuiteSparse Matrix Collection demonstrates that our probability-aware selective protection schemes can significantly reduce resilience overheads, frequently surpassing or matching the performance of both baseline and established protection schemes over all error probabilities.

The main contributions of this work are as follows:

- We formulate a probability-aware mathematical model for the selective protection of operations in the key SpMV operation of PCG.
- We develop machine learning models capable of predicting the slowdown resulting from a soft error in PCG from two statically known features, making the mathematical model practical in real-world applications.
- We conduct an experimental evaluation to demonstrate the effectiveness of our approach as a low-overhead system-level resilience technique.

The rest of this paper is structured as follows. Section II provides the necessary background on sparse linear systems, the PCG algorithm, and soft errors. In Section III, key equations are introduced to formulate the mathematical model of our selective protection scheme. Section IV explains how machine learning is employed to predict a key quantity for our mathematical model. Sections V and VI detail and

present the results of an experimental evaluation of our selective protection scheme. Section VII reviews related work on resilience techniques to protect sparse iterative solvers and surveys results on selective reliability for scientific applications. Finally, Section VIII concludes the paper and discusses potential avenues for future work.

II. Background

A. Sparse Linear Systems

Large-scale scientific applications often require the solving of sparse linear systems of equations. One set of such applications are physical simulations that implicitly solve partial differential equations (PDEs). One specific example from this set is solving the heat equation [19] on an $N \times 1$ rod. Fully representing the dependencies of the $N \times 1$ cells on each other would require an $N \times N$ matrix. Given a sufficiently large N, meaning each cell likely does not affect the majority of other cells, this $N \times N$ matrix would then be sparse.

Sparse matrices are commonly stored in sparse matrix storage formats such as coordinate list (COO), Compressed Sparse Row (CSR), Compressed Sparse Column (CSC), and more advanced formats to exploit the fact that a significant amount of matrix entries are zero [24]. These formats reduce the space complexity from $O(N^2)$ to $O(nnz)$, where nnz is the number of non-zero elements in the matrix.

B. Preconditioned Conjugate Gradient

While the standard CG method can theoretically solve the sparse system of linear equations in N iterations, the addition of a preconditioner tends to reduce the required number of iterations significantly. In particular, a preconditioner M is utilized such that $A \approx M$ (i.e., $M^{-1}Ax = M^{-1}b$ is solved). To keep computational cost low (i.e., reduce the cost of finding M and computing M^{-1}), normally a block diagonal preconditioner (e.g., block Jacobi) or incomplete Cholesky factorization (i.e., $A \approx LL^T$ where L is a sparse lower triangular matrix) is used. Concretely, PCG solves a linear system of the form $Ax = b$, where A is an $N \times N$ sparse and symmetric positive definite (SPD) matrix, b is a dense vector, and x is the solution vector. Pseudocode for PCG is shown in Algorithm 1. The algorithm's inputs are as follows: the coefficient matrix A, the preconditioner matrix M, the right-hand side vector b, the initial guess of the solution vector x_0, the threshold that defines solution convergence tol, and the maximum number of iterations allowed before the algorithm is aborted $maxit$.

The main loop of PCG, much like other iterative optimization techniques, focuses on updating the solution vector and finding the next search direction. Specifically, at any iteration i, PCG updates the solution vector x_i to x_{i+1} using the previous search direction p_i and generates a new search direction p_{i+1} that is A-orthogonal to each previous search direction, i.e., $\forall j \in \{0, 1, \ldots, i\}$, $p_{i+1}^T A p_j = 0$. Lastly, if the residual of the solution r_i is below the threshold tol, a satisfactory solution has been found.

Algorithm 1: Preconditioned Conjugate Gradient (PCG)

Input: $A, M, b, x_0, tol, maxit$
begin
  $r_0 \leftarrow b - Ax_0$; // Initial residual
  $z_0 \leftarrow M^{-1}r_0$; // Preconditioning
  $p_0 \leftarrow z_0$;
  $i \leftarrow 0$;
  while $i < maxit$ and $\|r_i\|/\|b\| > tol$ **do**
    $q_i \leftarrow Ap_i$;
    $v_i \leftarrow r_i^T z_i$;
    $\alpha \leftarrow v_i/(p_i^T q_i)$;
    $x_{i+1} \leftarrow x_i + \alpha p_i$; // Improve approximation
    $r_{i+1} \leftarrow r_i - \alpha q_i$; // Update residual
    $z_{i+1} \leftarrow M^{-1}r_{i+1}$; // Preconditioning
    $v_{i+1} \leftarrow r_{i+1}^T z_{i+1}$;
    $\beta \leftarrow v_{i+1}/v_i$;
    $p_{i+1} \leftarrow z_{i+1} + \beta p_i$; // New search direction
    $i \leftarrow i + 1$;
  end
end

The complexity of PCG is dominated by the sparse matrix-vector multiplication of Ap_i or the computation of M^{-1}. Sparse matrix-vector multiplication (SpMV) requires $O(nnz)$ time, while the computation cost of M^{-1} would be the $O(N)$ if a block diagonal preconditioner is used or $O(nnz)$ (i.e, the application of sparse triangular solve) if an incomplete Cholesky is used.

C. Soft Errors

As computing systems grow in scale, the threat of bit-flip errors due to environmental factors grows in tandem. The implementation of error-correcting codes (ECC) and similar mechanisms can drastically decrease the frequency of bit-flip errors by up to 10,000-fold [2], but they do not offer a complete solution. Consequently, software solutions are often employed to supplement these hardware-based approaches.

These errors can be divided into two categories: hard errors and soft errors. Hard errors result in crashes of running processes or entire systems, whereas soft errors modify application data in a way that could lead to incorrect calculations or excessively long runtimes without causing a failure. Hard errors are relatively straightforward to identify and rectify through system checkpointing [14], [25], where the entire system or application state is periodically dumped to disk and restored in the event of a crash. Soft errors, which are the main focus of this work, are not as easily detected and corrected, often requiring the replication of processes or the duplication of operations [5], [17]. Checkpointing and replication both impose significant overhead, thus any decrease in their frequency is highly beneficial.

III. Problem Formulation

Shantharam et al. [30] proved that if a soft error in PCG propagates into the p vector of the SpMV operation in PCG, it will ultimately impact all elements in p given a sufficient number of iterations. For this reason, we consider a problem where an error is introduced at some iteration i into a random

element j of the $N \times 1$ vector p. This error placement can be represented by a two-tuple (i, j), where $i \in \{0, 1, \ldots, I_o\}$ and $j \in \{0, 1, \ldots, N-1\}$. Here, I_o denotes the number of iterations for PCG to converge in an error-free run. From this point onward, we refer to this two-tuple as a *fault site*.

The presence of a soft error at a fault site (i, j) will typically still make PCG converge to the correct solution (which happened to all of our test cases). However, it may require substantially more iterations to reach convergence. We quantify this using *slowdown*, defined as the ratio of the number of iterations to converge with the error ($I_e(i, j)$) to the number of iterations to converge without errors (I_o), i.e.,

$$slowdown(i, j) = \frac{I_e(i, j)}{I_o} \ . \tag{1}$$

Figure 1 shows the mean slowdowns over 1000 PCG error runs with random fault sites for 15 matrices from the SuiteSparse Matrix Collection [12]. Figures 2a and 2b provide the corresponding number of iterations without error (I_o) and the mean number of iterations with errors (I_e) for these matrices.

Protecting, synonymous with replicating, an operation also introduces some cost. To relate the cost of errors with the cost of protection, we introduce the following model:

$$C_{protection}(i, j) = 1 \ , \tag{2}$$

$$C_{error}(i, j, p_e) = \frac{p_e}{N I_o}(I_e(i, j) - I_o)N \ . \tag{3}$$

In particular, we establish the cost of protecting any fault site (i, j) as one unit (Equation (2)), reflecting the need to replicate one dot product operation in SpMV. Subsequently, the cost associated with an error occurrence is determined by the total number of dot product operations performed during the additional iterations of PCG. However, errors should never be considered guaranteed. Given the probability of an error occurring in PCG, denoted as p_e, the expected cost of an error occurring at a particular fault site (i, j) is shown in Equation (3), calculated as the cost of the error $((I_e(i, j) - I_o)N)$ multiplied by the probability of the error occurring at said fault site $\frac{p_e}{N I_o}$. Here, we assume that each of the $N I_o$ potential fault sites has the same probability of having the error. It logically follows that protection should be done when the expected cost of an error occurring surpasses the cost of protection, i.e., $C_{error}(i, j, p_e) > C_{protection}(i, j)$. This trivial inequality leads to a decision criterion for when to protect any fault site as derived below:

$$\begin{aligned} & \frac{p_e}{N I_o}(I_e(i, j) - I_o)N > 1 \\ \Rightarrow \quad & \frac{I_e(i, j) - I_o}{I_o} > \frac{1}{p_e} \\ \Rightarrow \quad & \frac{I_e(i, j)}{I_o} - 1 > \frac{1}{p_e} \\ \Rightarrow \quad & slowdown(i, j) > 1 + \frac{1}{p_e} \ . \end{aligned} \tag{4}$$

Figure 3 shows a selective protection scheme constructed from this model for one matrix **ct20stif** from the SuiteSparse Matrix Collection [12] with $p_e = 0.1$.

If the slowdown for a given fault site were a known constant prior to runtime, this model could be readily applied. In the subsequent sections, we present and evaluate a machine learning-based approach aimed at predicting slowdown based on two statically known features.

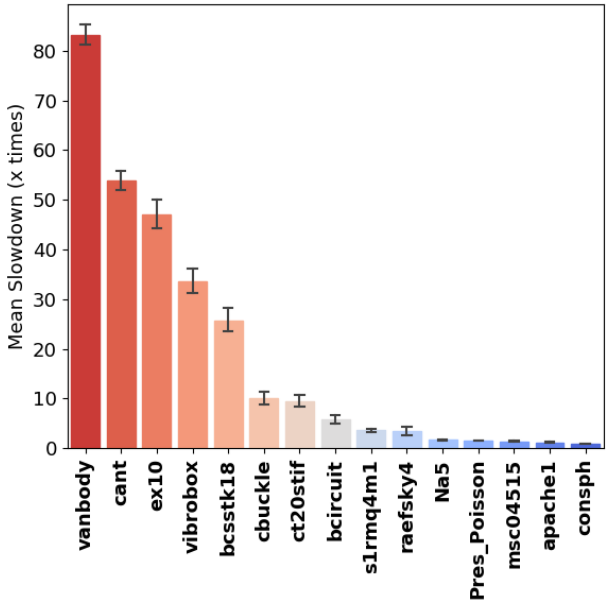


Fig. 1: Mean slowdowns over 1000 PCG error runs on 15 matrices from the SuiteSparse Matrix Collection [12]. Error bars indicate a 95% confidence interval of the mean.

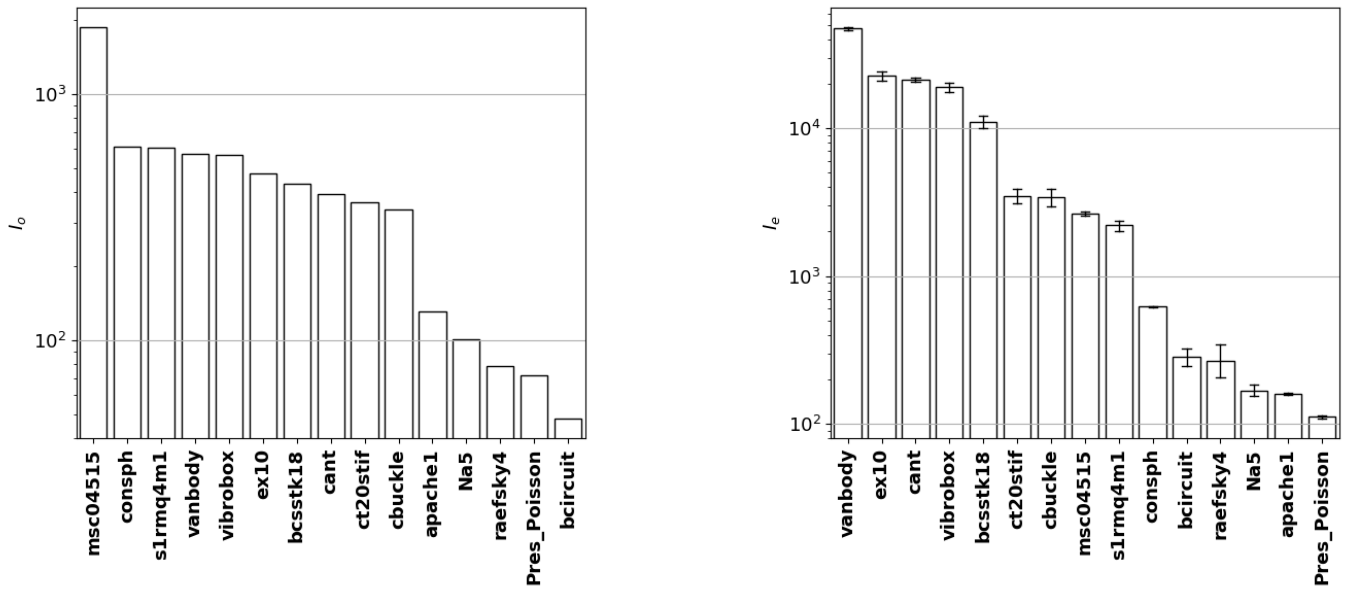
IV. Prediction of Soft Error Impacts

With the objective of estimating slowdown using information known prior to execution, machine learning is a natural direction to pursue. The top row of Figure 5 illustrates the correlation between two key features and the slowdown on four representative matrices: the PCG iteration number, i, and the 2-norm of the j-th row of A, $\|A_j\|_2$. The formal definition of $\|A_j\|_2$ is as follows:

$$\|A_j\|_2 = \sqrt{\sum_{k=1}^{N} A_{j,k}^2}. \tag{5}$$

Based on our observations, a higher $\|A_j\|_2$ typically results in a higher slowdown. On the other hand, i has a more complex correlation with slowdown, often peaking at clusters of iterations. The data shown was taken from 1,000 PCG runs with errors inserted at one random fault site for each run. Both features are static, thus any model trained on them can be used to predict the slowdowns offline, incurring no additional overhead during runtime.

To develop machine learning models capable of capturing this correlation, a Bayesian hyperparameter search [33] is conducted over a set of five standard regression models. This set consists of polynomial regression, random decision forest, K-nearest neighbors, gradient-boosted decision trees (XGBoost),



(a) (b)

Fig. 2: I_o and I_e values on 15 matrices from the SuiteSparse Matrix Collection [12]. (a) Number of iterations without error (I_o); (b) Mean number of iterations with errors (I_e) over 1000 PCG error runs. Error bars indicate a 95% confidence interval of the mean.

and support vector machines. This Bayesian hyperparameter search employs 3-fold cross-validation on a dataset comprising 2,000 experimental PCG runs with an injected error, optimizing the validation r^2 of each model. Once the hyperparameter search is completed, five optimized models are produced to choose from.

However, while regression performance is significant, the primary focus lies in making the correct decision regarding whether to protect or not to protect. Therefore, classification metrics should be considered when determining which of the five optimized models should be employed.

This model selection should prioritize the model that excels in classifying whether the slowdown will be greater or less than $1 + \frac{1}{p_e}$, as shown in Equation (4). It is important to note that varying p_e will impact model performance. Therefore, we evaluate classification scores across a range of 100 different p_e values from 0 to 1 (with a 0.01 increment). To prevent bias towards any specific p_e, a model should be selected based on some average classification score across all p_e's. In practice, if some specific range of p_e is preferred, an average of only that particular range could be taken.

We focus on two specific classification metrics, accuracy and F_β score. Accuracy is presented for ease of human interpretation, but it is not effective in assessing model performance in scenarios where classes are imbalanced, which is common in this setting. For a more effective assessment of model performance, we utilize the F_β score, which is defined as follows:

$$F_\beta = (1 + \beta^2) \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \ . \tag{6}$$

Specifically, we choose the F_β score with $\beta = 2$. This choice is motivated by the goal of optimizing both precision and recall, while also taking into account the imbalance in misclassification cost. If a false positive occurs, indicating that a fault site was protected when it did not encounter an error, a cost of one is incurred. Conversely, if a false negative occurs, indicating that a fault site was not protected when it did encounter an error, we are likely to incur a significantly larger cost. For these reasons, we prioritize the optimization of recall over precision, making $F_{\beta=2}$ a natural choice. We also explored other choices of $\beta > 2$, which do not significantly affect the model choice and performance. Upon making this model choice, PCG can be effectively protected, as will be validated in the following sections. Figure 4 gives a complete outline of our protection scheme production framework.

V. Evaluation

To assess our selective protection schemes against other existing schemes, it is crucial to employ a metric that encourages error prevention while minimizing the total number of protections. A suitable metric meeting these criteria is the computational overhead incurred when an error strikes a particular fault site (i, j), which is defined as follows:

$$overhead(i,j) = \frac{\sum_{k=1}^{I_e(i,j)} \left(N + n_k\right) - I_o N}{I_o N} \ , \tag{7}$$

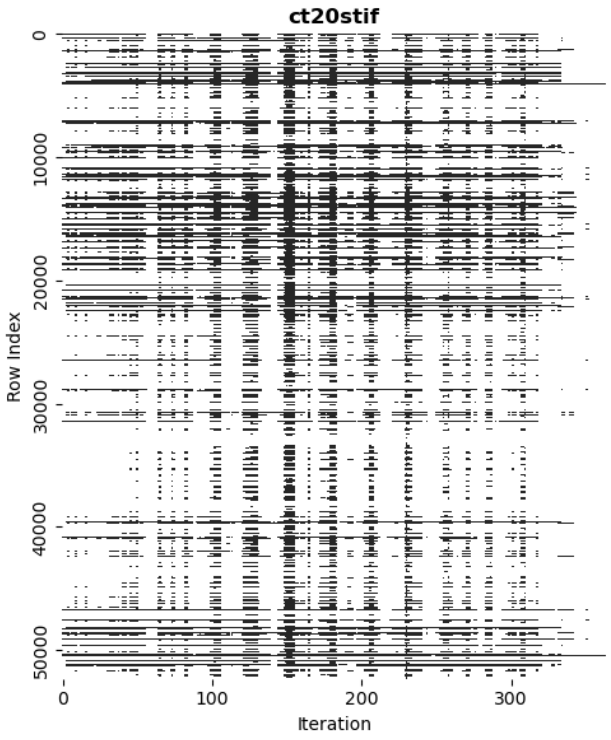


Fig. 3: A selective protection scheme for the matrix **ct20stif** with $p_e = 0.1$. Black cells indicate protected fault sites.

where n_k denotes the total number of protections at iteration k. This formula essentially compares two quantities, $\sum_{k=1}^{I_e(i,j)} \big(N + n_k\big)$ and $I_o N$. The first quantity $\sum_{k=1}^{I_e(i,j)} \big(N + n_k\big)$ represents the complete cost of an error-prone run including the cost of protections at each iteration, and the second quantity $I_o N$ is simply the cost of an error-free run without any protection. Overhead can be better understood through the following key scenarios:

- Full protection: $overhead(i,j) = 1$. Protecting $n_k = N$ elements at each iteration k simplifies Equation (7) to $\frac{2I_o N - I_o N}{I_o N}$, resulting in an overhead of one, regardless of error presence. Note that no slowdown will be incurred in this scenario even if an error is present since all elements are protected (i.e., $I_e(i,j) = I_o$).
- No protection with no error present: $overhead(i,j) = 0$. Protecting no elements at each iteration without an error occurring simplifies Equation (7) to $\frac{I_o N - I_o N}{I_o N}$, resulting in an overhead of zero.
- No protection with error present: $overhead(i,j) > 1$ is likely. Protecting no elements at each iteration in the presence of an error occurring at fault site (i,j) often leads to $\sum_{k=1}^{I_e(i,j)} \big(N + 0\big)$ being much greater than $I_o N$, resulting in an overhead greater than one. It is also possible to have $overhead(i,j) \leq 1$ in this scenario if $I_e(i,j)$ does not increase by too much relative to I_o.
- Selective protection with error present:
  - $overhead(i,j) \leq 1$ if the fault site is protected. Protecting between 0 and N elements at each iteration while correctly protecting the fault site leads to $\sum_{k=1}^{I_e(i,j)} \big(N + n_k\big) - I_o N = \sum_{k=1}^{I_o} \big(N + n_k\big) - I_o N = \sum_{k=1}^{I_o} n_k$ being less than $I_o N$, resulting in an overhead between zero and one.
  - $overhead(i,j) > 1$ is likely if the fault site is not protected. Protecting between 0 and N elements at each iteration while not correctly protecting the fault site will often lead to $\sum_{k=1}^{I_e(i,j)} \big(N + n_k\big)$ being much greater than $I_o N$, resulting in an overhead greater than one. It is also possible to have $overhead(i,j) \leq 1$ in this scenario if $I_e(i,j)$ and/or n_k are relatively small.

Utilizing this key metric, we offer a comprehensive evaluation of our selective protection schemes compared to two baseline schemes as well as two previously established protection schemes:

- No protection: A baseline scheme where no protection is employed.
- Full protection: A baseline scheme that protects all potential fault sites.
- Best Random %: Among all potential fault sites, randomly protect P percent of them. Integer values of P ranging from 1 to 99 are tested, and the P yielding the best performance is selected. This protection scheme was presented in [32].
- Best 2-Norm %: For every potential fault site, protect if $\|A_j\|_2$ is in the top P percentile of 2-norms in A. Integer values of P ranging from 1 to 99 are tested, and the P yielding the best performance is selected. This protection scheme was presented in [38].

In our evaluation, we select 15 matrices from the SuiteSparse Matrix Collection [12]. These matrices were chosen at random with the caveat that they must be SPD due to that being a prerequisite for PCG. Table I presents these matrices alongside their row/column count (N), number of non-zero elements (nnz), and element density (nnz/N^2).

For each selected matrix, we execute the PCG algorithm using a zero fill sparse incomplete Cholesky Factorization from Matlab (i.e, `icho`) as the preconditioner, $tol = 10^{-6}$, $maxiter = 100 \cdot I_o$, $b = A \cdot \vec{\mathbf{1}}_N$ where $\vec{\mathbf{1}}_N$ is a vector of length N containing only ones, and the initial guess $x_0 = \vec{\mathbf{0}}_N$ where $\vec{\mathbf{0}}_N$ is a vector of length N containing only zeros. Experimental setups similar to this have been used in previous works (e.g. [7], [26], [38], [39]). The error-free iteration count I_o for each matrix in this setup is also shown in Table I. A random fault site is sampled from a uniform distribution such that in the i-th iteration of PCG, the j-th element of vector p (Line 7 of Algorithm 1) has an error injected in it. Instead of inducing a physical bit-flip in the data operated on by SpMV, a straightforward value augmentation is done. Specifically, the value of the j-th element in vector p is increased by the maximum value in p, i.e., $p_j = p_j + \max_k p_k$. Following the injection of this error, the corresponding element may or may not be appropriately protected, after which the $overhead$ of the run is recorded. This process is repeated 1,000 times,

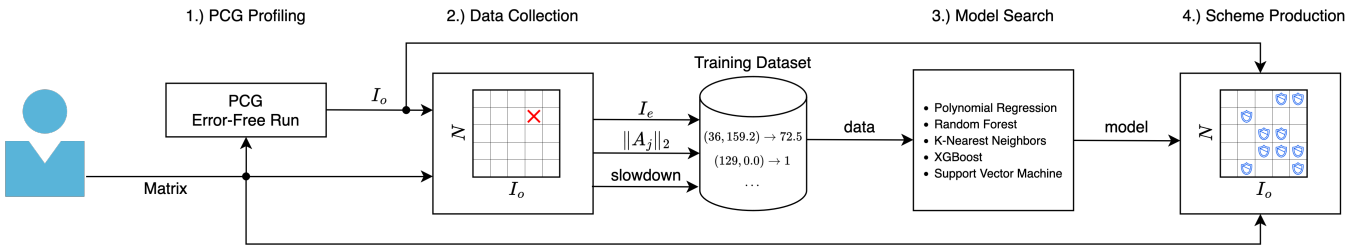


Fig. 4: An overview of the selective protection scheme production framework.

forming the runs where an error is present in the evaluation.

To account for the probability of an error occurring, p_e, there must also be $1000/p_e - 1000$ runs where an error does not occur, thus a total of $1000/p_e$ runs. Due to the deterministic nature of PCG and our machine learning models, all non-error PCG runs will produce identical overhead results. Therefore, we duplicate the overhead of one non-error PCG run $1000/p_e - 1000$ times for each considered p_e. Once all evaluation runs are completed, the average overhead is computed over all recorded overhead values. The probability of an error, p_e, is varied between 0 and 1 in our evaluation, with an increment of 0.01.

Table I: 15 matrices selected from the SuiteSparse Matrix Collection [12] for performance evaluation.

Id	*Matrix*	*N*	*nnz*	*Density*	I_o
1	ex10	2410	54840	0.94%	479
2	msc04515	4515	97707	0.48%	1879
3	s1rmq4m1	5489	262411	0.87%	605
4	Na5	5832	305630	0.9%	101
5	bcsstk18	11948	149090	0.1%	432
6	vibrobox	12328	301700	0.2%	565
7	cbuckle	13681	676515	0.36%	341
8	Pres_Poisson	14822	715804	0.33%	72
9	raefsky4	19779	1316789	0.34%	79
10	vanbody	47072	1751178	0.079%	491
11	ct20stif	52329	2600295	0.095%	364
12	cant	62451	4007383	0.1%	394
13	bcircuit	68902	375558	0.0079%	48
14	apache1	80800	542184	0.0083%	131
15	consph	83334	6010480	0.087%	611

VI. Results

In this section, we present performance metrics evaluating the quality of our predictions of soft error impacts, alongside a comparison of our produced selective protection schemes against both baseline approaches and existing schemes from previous works.

A. Prediction of Soft Error Impacts

The bottom row of Figure 5 presents the regression results for four models over four matrices based on 1,000 PCG runs, achieving high r^2 values over a diverse set of correlations. The mean r^2 of our optimized models across all 15 matrices is 0.84, indicating strong overall regression fits.

As mentioned previously, classification results are more indicative of good performance due to our setting ultimately being a decision problem. Figure 6 shows accuracies and $F_{\beta=2}$ scores of all five models produced from the Bayesian hyperparameter search over the same four matrices. Per our methodologies, the model with the highest average $F_{\beta=2}$ score over all error probabilities is taken. One can see that the best models often surpass the 0.9 mark, indicating a good balance of avoiding false negatives and false positives. Notably, this metric is still not a direct measurement of protection scheme performance, which is given below.

B. Protection Scheme Performance

To evaluate our produced protection schemes, Figure 7 shows the mean overheads and the corresponding 95% confidence intervals of three protection schemes on four representative matrices, varying p_e. The "Best Random %" scheme is omitted in the interest of visual clarity and the lack of competitiveness (its results are summarized for selected p_e's along with those of other schemes in Table II). The "Full Protection" scheme is also omitted due to its inherent 100% overhead with zero uncertainty.

It is evident that the performance of the "No Protection" scheme can fluctuate significantly. This approach typically performs exceptionally well when the error probability p_e is extremely small but tends to perform disastrously when p_e is substantial, particularly when the average slowdown associated with an error occurrence for a given matrix is significant. We classify matrices with high average slowdowns when errors occur as *dangerous*, as exemplified by the **vanbody** matrix. Conversely, matrices with low average slowdowns are deemed *non-dangerous*, as exemplified by the **Na5** matrix. In the case of non-dangerous matrices, the "No Protection" scheme can perform effectively even at high values of p_e.

The "Best 2-Norm %" scheme maintains overheads below 100% for every value of p_e. While this scheme consistently outperforms "Full Protection", it does not account for p_e, leading to higher overheads for low values of p_e.

"Our Scheme" refers to the decisions made from Equation (4) where the slowdown is predicted by the optimized model produced from Section IV. This scheme consistently approximates the performance of "No Protection" at low p_e values, being the only scheme to do so. When p_e is significant, "Our Scheme" often outperforms "Best 2-Norm %" with

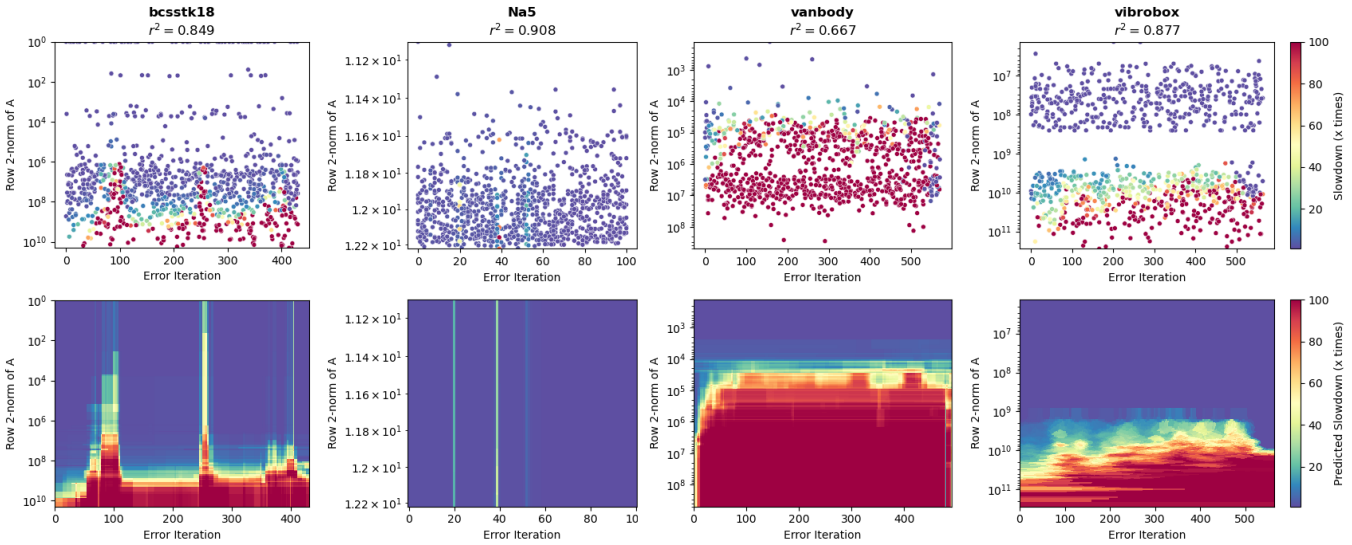


Fig. 5: Correlations between the error iteration number i, the 2-norm of the j-th row of A, and the resulting slowdown, along with regression fits r^2 over 1000 PCG error runs on four representative matrices.

diminishing margins as p_e increases. **Na5** is the only matrix where this margin increases. Across 8 out of 15 matrices, "Our Scheme" exhibits a 95% confidence interval lower than that of "Best 2-Norm %" at all values of p_e. In matrices where this is not the case, the confidence interval of "Our Scheme" overlaps with that of "Best 2-Norm %" for less than 50% of p_e values, almost always being the lower end of the overlap.

Table II presents the mean overheads of four protection schemes on all 15 matrices, with p_e fixed at 0.1, 0.5, and 0.9, reflecting low, medium, and high probabilities of error, respectively. The "Full Protection" scheme is again omitted due to its inherent 100% overhead with zero uncertainty. At $p_e = 0.1$, "Our Scheme" outperforms "No Protection" by an average of 145.8% with a maximum difference of 723.8%, "Best Random %" by an average of 36.96% with a maximum difference of 88.50%, and "Best 2-Norm %" on average by 14.93% with a maximum difference of 52.09%. At $p_e = 0.5$, "Our Scheme" outperforms "No Protection" by an average of 846.8% with a maximum difference of 4,008%, "Best Random %" by an average of 36.85% with a maximum difference of 88.17%, and "Best 2-Norm %" on average by 5.747% with a maximum difference of 17.97%. At $p_e = 0.9$, "Our Scheme" outperforms "No Protection" by an average of 1,556% with a maximum difference of 7,294%, "Best Random %" by an average of 43.92% with a maximum difference of 92.33%, and "Best 2-Norm %" on average by 4.600% with a maximum difference of 26.13%.

These results demonstrate that our schemes consistently match or outperform other existing protection schemes, regardless of whether the matrices are dangerous or non-dangerous. It has been shown that our schemes more accurately determine which fault sites are crucial to protect and which are not, resulting in lower resilience overheads on average. Furthermore, our schemes are capable of achieving the previously unparalleled performance of "No Protection" at very low levels of p_e, while surpassing existing schemes as p_e increases.

VII. Related Work

This section reviews related work on resilience techniques to protect sparse iterative solvers and surveys results on selective reliability for scientific applications.

A. Resilence for Sparse Iterative Solvers

Due to the importance of sparse iterative solvers (e.g., CG, PCG, GMRES) in scientific computing [18], [19], various resilience methods have been proposed to protect them against hard failures and soft errors. Checkpointing, replication, and recomputation are general techniques to protect and/or recover scientific applications from faults [20]. For example, Dichev and Nikolopoulos [13] applied replication to detect soft errors for the PCG solver. Sloan et al. [32] used partial recomputation by performing a binary search to locate the errors. Benoit et al. [4] combined replication (duplication and triplication) and checkpointing to detect and correct soft errors.

Besides general-purpose approaches, many application-specific resilience techniques have also been proposed. Algorithm-based fault tolerance (ABFT) is one such technique [10], [23], which was originally designed for dense linear algebra but has been successfully applied to protect computations that involve sparse matrices. Bronevestky and de Supinski [7] applied ABFT-based matrix encoding to protect the CG solver. Shantharam et al. [31] develop an ABFT-encoded scheme to protect PCG while exploring the symmetric positive definite and diagonally dominant properties of the sparse matrices. Schöll et al. [29] applied a similar technique but used a blocked checksum approach for PCG. While most existing work only applied a roll-back method to recover applications upon fault detection, Tao et al. [39] combined roll-back and roll-forward

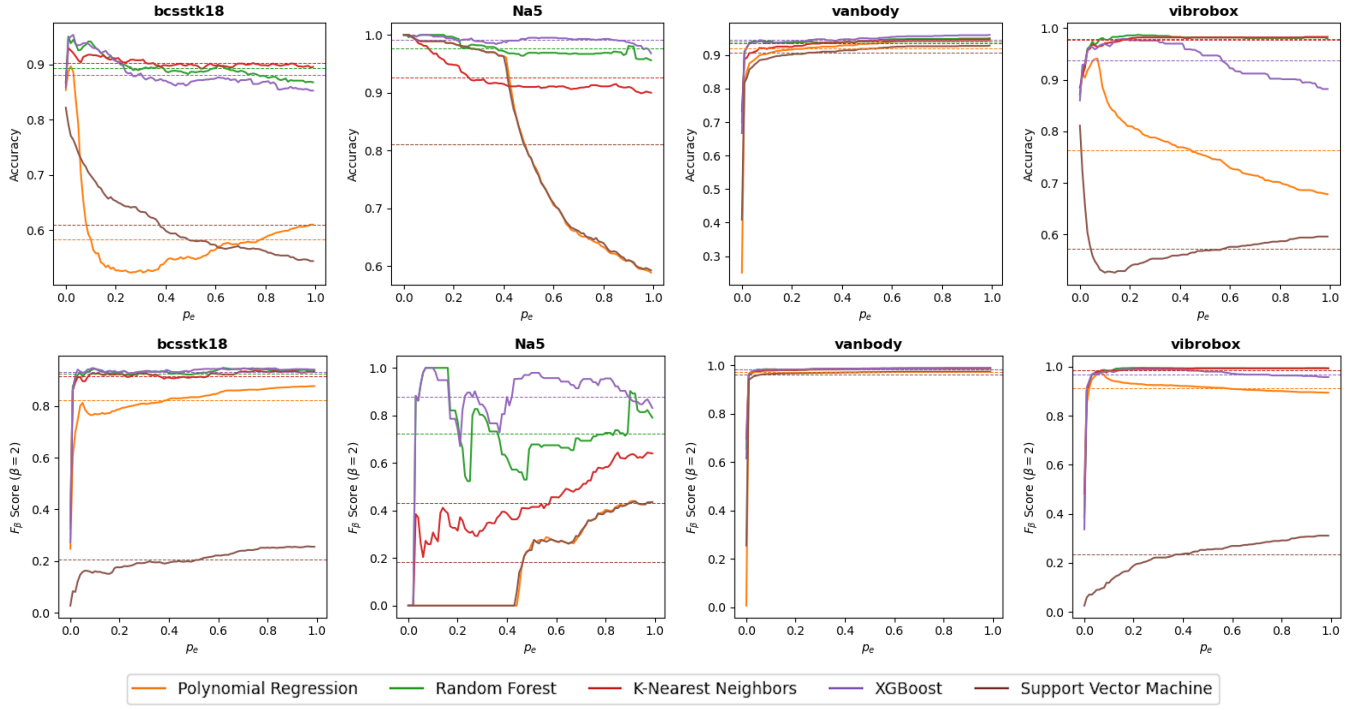


Fig. 6: Accuracy (top row) and $F_{\beta=2}$ score (bottom row) of each model over 1000 PCG error runs on four representative matrices, while varying p_e between 0 and 1. Dashed lines indicate an average over all p_e values.

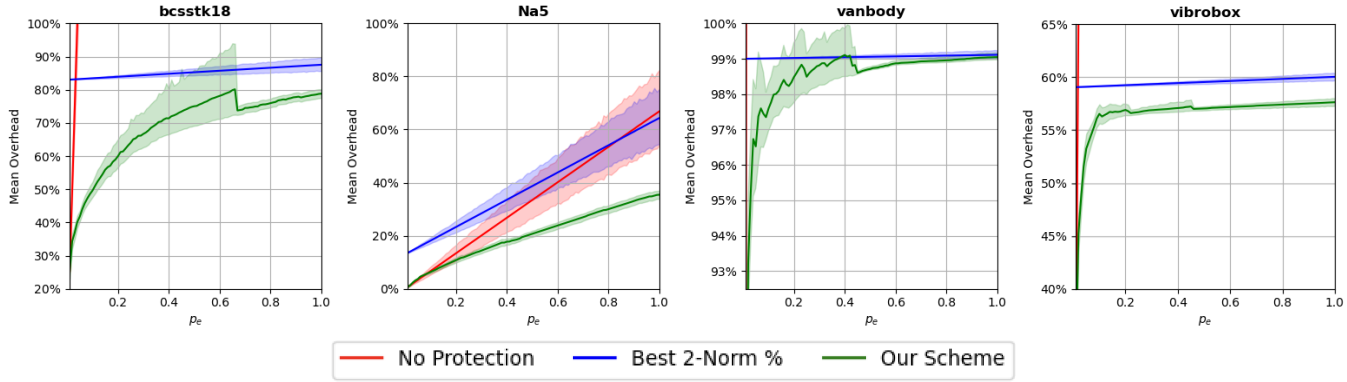


Fig. 7: Mean overhead (in %) of three protection schemes over 1000 PCG error runs and $\frac{1000}{p_e} - 1000$ non-error runs on four representative matrices, while varying p_e between 0 and 1. Shaded regions indicate the 95% confidence interval of the mean.

methods to provide a more complete protection scheme for the PCG solver. A similar technique was also developed in [16] to protect CG with both error detection and correction capabilities. Another application-specific technique to protect sparse iterative solvers is to explore their numerical and convergence properties to make them resilient to faults. Chen [9] performed error detection with a periodic verification of the orthogonal and residual properties of the Krylov subspace, and used checkpoint/restart if errors were detected. Sao and Vuduc [27] proposed a self-stabilizing approach by periodically checking and restoring the orthogonality properties of iterative solvers. Schöll et al. [28] also combined orthogonality restoration and periodic checkpointing to achieve both roll-back and roll-forward recoveries for PCG. Agullo et al. [1] combined the residual gap criterion and a bound on the α value to detect soft errors. They showed the robustness of this method even under finite precision computation.

All application-specific techniques above require modifying the underlying numerical libraries of the solvers, which is an intrusive approach that is often not easy and sometimes impossible to implement (when accessing the numerical libraries is limited). In this case, general-purpose techniques remain the most viable resilience approach. This is the approach we assumed in this paper by duplicating the computation to detect soft errors and re-running an iteration if errors are detected.

Table II: Mean overhead (in %) of four protection schemes over 1000 PCG error runs and $\frac{1000}{p_e} - 1000$ non-error runs on 15 matrices from the SuiteSparse Matrix Collection [12], where $p_e \in \{0.1, 0.5, 0.9\}$. Note that the numbers in each column are aligned by decimal point.

	p_e = 0.1				p_e = 0.5				p_e = 0.9			
Matrix	No Protection	Best Random %	Best 2-Norm %	Our Scheme	No Protection	Best Random %	Best 2-Norm%	Our Scheme	No Protection	Best Random %	Best 2-Norm%	Our Scheme
apache1	2.133	3.131	13.27	2.133	10.67	11.51	18.32	10.46	19.20	20.08	23.38	18.15
bcircuit	49.03	98.72	54.07	23.81	245.2	105.5	54.18	47.66	441.3	101.0	54.30	52.90
bcsstk18	247.2	103.2	83.45	49.62	1236	119.1	85.27	75.02	2225	133.4	87.08	77.80
cant	529.8	102.4	99.55	92.65	2649	126.2	99.57	94.78	4768	142.1	99.60	96.42
cbuckle	90.60	99.71	34.85	27.83	453.0	115.2	42.05	40.81	815.5	133.2	49.25	48.09
consph	0.1376	1.137	1.137	0.1376	0.6882	1.683	1.681	0.6882	1.239	2.218	2.225	1.239
ct20stif	85.57	98.74	98.09	46.01	427.9	113.3	98.47	85.02	770.2	110.3	98.85	92.53
ex10	461.2	103.0	66.63	53.99	2306	133.7	73.22	69.76	4151	127.5	79.81	77.84
msc04515	4.054	5.009	19.63	4.054	20.27	20.97	22.14	16.19	36.49	36.86	24.65	21.12
Na5	6.687	26.96	18.15	6.339	33.44	44.68	38.68	20.71	60.19	63.11	59.22	33.09
Pres_Poisson	5.468	8.311	13.02	5.468	27.34	29.46	29.10	24.94	49.22	51.14	45.18	41.64
raefsky4	23.84	93.28	5.984	4.783	119.2	97.00	9.917	8.833	214.6	99.99	13.85	13.03
s1rmq4mi	26.38	98.67	51.20	22.30	131.9	101.6	51.52	45.90	237.5	106.0	51.84	48.90
vanbody	821.4	104.0	99.01	97.59	4107	119.0	99.06	98.71	7393	191.3	99.10	99.01
vibrobox	326.2	101.5	59.15	56.54	1631	110.3	59.55	57.05	2936	119.9	59.94	57.53

B. Selective Reliability

While some parts of an application are important and should be protected at all costs, some other parts (i.e., non-critical data) can be loosely protected or or even not protected with little impact on the results. This is referred to as selective reliability, which has been considered by some authors to reduce the resilience overhead for certain applications.

Bridges et al. [6] and Elliott et al. [15] applied a "sandbox" model that provides isolation of unreliable parts of an application to contain the impact of faults. They applied this model to the GMRES solver and designed a fault-tolerant version of it, called FT-GMRES, that uses selective reliability to its inner and outer loops. Casas et al. [8] considered the Algebraic Multi Grid (AMG) solver, which is vulnerable to pointer corruptions. They designed a resilience technique that selectively protects the most critical pointers as well as the operations that access them. In a series of works, Subasi et al. [34]–[37] applied partial redundancy to different scientific applications (e.g., sparse LU, Cholesky, FFT, Stream) to mitigate both fail-stop and silent errors. By observing that different tasks/phases in an application exhibit different vulnerabilities and may have different reliability requirements, they designed techniques to selectively replicate certain tasks in the application, either through programmer-directed annotations [34] or by automatic runtime heuristics [35]–[37], which determine which tasks are critical and therefore should be replicated.

Sun et al. [38] applied selective protection to the PCG solver to reduce the resilience overhead against soft errors. They presented a static scheme that uses only the row-2-norm of the sparse matrix to decide which elements should be protected based on an analytical model. The closest work to ours is done by Chen et al. [11], which considered a similar problem while utilizing machine learning to predict the impact of soft errors. A dynamic scheme was proposed to select the elements to be protected if the impact crosses a threshold. Both of these works, however, did not take the error probability into account. In this paper, we presented a probability-aware selective protection scheme with a versatile machine-learning model that improves upon the previous results.

VIII. Conclusion and Future Work

A. Summary

In this work, we have introduced a probability-aware framework for producing low-overhead selective protection schemes for the widely used PCG algorithm. Our schemes are informed by two statically known features and one probability parameter. Using this information, a straightforward mathematical model guided by an optimized machine learning model is proposed, capable of making highly accurate protection decisions. Experimental results derived from an evaluation using matrices from a subset of the SuiteSparse Matrix Collection demonstrate that our schemes effectively reduce the resilience overheads, outperforming both baseline and existing protection schemes across all error probabilities.

The code and results of this work are available at https://github.com/DanielRJohnson/Dynamic-Selective-Protection.

B. Future Work

One key area for future research is the creation of a unified slowdown prediction model that performs well across all matrices. To achieve this, the disparate correlations between the iteration number and the row 2-norm with the slowdown would need to be unified through the use of other matrix features. Several features could be explored in this process, such as the number of non-zero elements in A (nnz), the density of A (nnz/N^2), the average non-zero element value ($\frac{1}{nnz}\sum_{i=1}^{N}\sum_{j=1}^{N} A_{i,j}$), or even an embedding of the sparsity pattern of A. Producing a unified model would also necessitate closely matching or surpassing the performance of the individual models, which could prove challenging.

Another area worthy of exploration is delving deeper into extremely small values of p_e. Because our choice of $maxiter$ is $100 \cdot I_o$, the minimum possible p_e before our approach regresses into "No Protection" is $1/99$, simply derived directly from Equation (4). While this could be beneficial for systems with an exceedingly low probability of errors, it makes the data

collection process extremely long. With $maxiter = 100 \cdot I_o$, certain highly dangerous matrices like **vanbody** and **cant** already require days to gather a substantial amount of data. Clever techniques will need to be developed to improve the efficiency of the data collection process without compromising the model performance for small values of p_e.

Lastly, our current framework requires perfect knowledge of p_e in order to make protect or no protect decisions. Although error probabilities can be estimated through system failure logs and application profiling, the estimate may not be accurate. How to make robust protection decisions with uncertainty of p_e is another area worthy of investigation.

ACKNOWLEDGEMENT

This work is supported in part by the U.S. National Science Foundation grants #2135309, #2135310, and the institutional fund of the University of Kansas.

REFERENCES

[1] E. Agullo, S. Cools, E. F. Yetkin, L. Giraud, N. Schenkels, and W. Vanroose. On soft errors in the conjugate gradient method: Sensitivity and robust numerical detection. *SIAM Journal on Scientific Computing*, 42(6):C335–C358, 2020.

[2] R. Baumann. Soft errors in advanced computer systems. *IEEE design & test of computers*, 22(3):258–266, 2005.

[3] R. C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and materials reliability*, 5(3):305–316, 2005.

[4] A. Benoit, A. Cavelan, F. Cappello, P. Raghavan, Y. Robert, and H. Sun. Coping with silent and fail-stop errors at scale by combining replication and checkpointing. *J. Parallel Distrib. Comput.*, 122:209–225, 2018.

[5] M. Bougeret, H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni. Using group replication for resilience on exascale systems. *Int. J. High Perform. Comput. Appl.*, 28(2):210–224, May 2014.

[6] P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen. Fault-tolerant linear solvers via selective reliablity. *Sandia National Laboratories*, 2012.

[7] G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *ICS*, pages 155–164, 2008.

[8] M. Casas, B. R. de Supinski, G. Bronevetsky, and M. Schulz. Fault resilience of the algebraic multi-grid solver. In *ICS*, page 91–100, 2012.

[9] Z. Chen. Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *PPoPP*, 2013.

[10] Z. Chen and J. Dongarra. Algorithm-based fault tolerance for fail-stop failures. *IEEE Transactions on Parallel and Distributed Systems*, 19(12):1628–1641, Dec 2008.

[11] Z. Chen, T. Verrecchia, H. Sun, J. D. Booth, and P. Raghavan. Dynamic selective protection of sparse iterative solvers via ML prediction of soft error impacts. In *FTXS*, 2023.

[12] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), 2011.

[13] K. Dichev and D. S. Nikolopoulos. TwinPCG: Dual thread redundancy with forward recovery for preconditioned conjugate gradient methods. In *CLUSTER*, 2016.

[14] J. Dongarra, T. Herault, and Y. Robert. *Fault tolerance techniques for high-performance computing*. Springer, 2015.

[15] J. Elliott, M. Hoemmen, and F. Mueller. Evaluating the impact of SDC on the GMRES iterative solver. In *IPDPS*, 2014.

[16] M. Fasi, Y. Robert, and B. Uçar. Combining backward and forward recovery to cope with silent errors in iterative solvers. In *PDSEC*, 2015.

[17] K. Ferreira, J. Stearley, J. H. Laros, III, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the viability of process replication reliability for exascale systems. In *SC*, pages 44:1–44:12, 2011.

[18] G. Golub and J. M. Ortega. *Scientific Computing: An Introduction with Parallel Computing*. Academic Press, 1993.

[19] M. T. Heath. *Scientific computing: an introductory survey, revised second edition*. SIAM, 2018.

[20] T. Hérault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*, Computer Communications and Networks. Springer Verlag, 2015.

[21] S. Krishnamohan and N. R. Mahapatra. A highly-efficient technique for reducing soft errors in static cmos circuits. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings.*, pages 126–131. IEEE, 2004.

[22] Kuang-Hua Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, C-33(6):518–528, June 1984.

[23] Kuang-Hua Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, C-33(6):518–528, 1984.

[24] D. Langr and P. Tvrdik. Evaluation criteria for sparse matrix storage formats. *IEEE Transactions on parallel and distributed systems*, 27(2):428–440, 2015.

[25] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Nov 2010.

[26] B. Ozcelik Mutlu, G. Kestor, J. Manzano, O. Unsal, S. Chatterjee, and S. Krishnamoorthy. Characterization of the impact of soft errors on iterative methods. In *HiPC*, pages 203–214, 2018.

[27] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *ScalA*, 2013.

[28] A. Schöll, C. Braun, M. A. Kochte, and H. Wunderlich. Low-overhead fault-tolerance for the preconditioned conjugate gradient solver. In *DFTS*, 2015.

[29] A. Schöll, C. Braun, M. A. Kochte, and H. Wunderlich. Efficient algorithm-based fault tolerance for sparse matrix operations. In *DSN*, 2016.

[30] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Characterizing the impact of soft errors on iterative methods in scientific computing. In *ICS*, 2011.

[31] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *ICS*, 2012.

[32] J. Sloan, R. Kumar, and G. Bronevetsky. An algorithmic approach to error localization and partial recomputation for low-overhead fault tolerance. In *DSN*, 2013.

[33] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[34] O. Subasi, J. A. Moreno, O. S. Unsal, J. Labarta, and A. Cristal. Programmer-directed partial redundancy for resilient HPC. In *Computing Frontiers*, 2015.

[35] O. Subasi, O. S. Unsal, and S. Krishnamoorthy. Automatic risk-based selective redundancy for fault-tolerant task-parallel HPC applications. In *ESPM2@SC*, 2017.

[36] O. Subasi, G. Yalcin, F. Zyulkyarov, O. S. Unsal, and J. Labarta. A runtime heuristic to selectively replicate tasks for application-specific reliability targets. In *IEEE CLUSTER*, 2016.

[37] O. Subasi, G. Yalcin, F. Zyulkyarov, O. S. Unsal, and J. Labarta. Designing and modelling selective replication for fault-tolerant HPC applications. In *CCGRID*, 2017.

[38] H. Sun, A. Gainaru, M. Shantharam, and P. Raghavan. Selective protection for sparse iterative solvers to reduce the resilience overhead. In *SBAC-PAD*, 2020.

[39] D. Tao, S. L. Song, S. Krishnamoorthy, P. Wu, X. Liang, E. Z. Zhang, D. Kerbyson, and Z. Chen. New-sum: A novel online ABFT scheme for general iterative methods. In *HPDC*, 2016.