Dynamic Selective Protection of Sparse Iterative Solvers via ML Prediction of Soft Error Impacts

Zizhao Chen
University of Kansas
Lawrence, KS, USA
zchen@ku.edu

Thomas Verrecchia
ENSTA Paris
Palaiseau, France
thomas.verrecchia@ensta-paris.fr

Hongyang Sun
University of Kansas
Lawrence, KS, USA
hongyang.sun@ku.edu

Joshua D. Booth
University of Alabama in Huntsville
Huntsville, AL, USA
joshua.booth@uah.edu

Padma Raghavan
Vanderbilt University
Nashville, TN, USA
padma.raghavan@vanderbilt.edu

ABSTRACT

Soft errors occur frequently on large computing platforms due to the increasing scale and complexity of HPC systems. Various resilience techniques (e.g., checkpointing, ABFT, and replication) have been proposed to protect scientific applications from soft errors at different levels. Among them, system-level replication often involves duplicating or even triplicating the entire computation, thus resulting in high resilience overhead. This paper proposes dynamic selective protection for sparse iterative solvers, in particular for the Preconditioned Conjugate Gradient (PCG) solver, at the system level to reduce the resilience overhead. For this method, we leverage machine learning (ML) to predict the impact of soft errors that strike different elements of a key computation (i.e., sparse matrix-vector multiplication) at different iterations of the solver. Based on the result of the prediction, we design a dynamic strategy to selectively protect those elements that would result in a large performance degradation if struck by soft errors. An experimental evaluation demonstrates that our dynamic protection strategy is able to reduce the resilience overhead compared to existing algorithms.

KEYWORDS

Fault tolerance, soft errors, dynamic selective protection, iterative solvers, preconditioned conjugate gradient.

ACM Reference Format:
Zizhao Chen, Thomas Verrecchia, Hongyang Sun, Joshua D. Booth, and Padma Raghavan. 2023. Dynamic Selective Protection of Sparse Iterative Solvers via ML Prediction of Soft Error Impacts . In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3624062.3624117

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11.
https://doi.org/10.1145/3624062.3624117

1 INTRODUCTION

Sparse iterative solvers, such as the Preconditioned Conjugate Gradient (PCG) solver, represent an important class of numerical methods in scientific computing for solving sparse systems of linear equations. When these iterative solving techniques are used at scale (i.e., on large linear systems), the number of iterations and clock cycles will be high. As such, these methods are exposed to both hard failures and soft errors, which occur at higher rates due to the large system scale and increasing hardware complexity. Thus, fault tolerance or resilience techniques have been widely recognized as a critical component to ensure the effective use of HPC systems [1, 10].

Numerous resilience solutions have been proposed over the years to protect iterative solvers from faults, ranging from system-level techniques using checkpointing and replication [5] to application-specific techniques such as algorithm-based fault tolerance (ABFT) [2, 8, 12]. In this work, we focus on system-level replication as a general-purpose approach to transparently detect and mitigate soft errors for the PCG solver. Traditional replication strategies often duplicate or even triplicate the entire computation, which incurs high resilience overheads. However, not all elements of the computation are equally susceptible to soft errors. While errors striking certain elements could cause significant performance degradations, errors striking other elements will have little impact on the solver's convergence [7, 8]. This creates an opportunity for reducing the resilience overheads by selectively protecting only the critical elements.

This work proposes this type of low-overhead resilience algorithm by using dynamic selective protection. We first perform a characterization of the impacts of soft errors on the convergence of PCG by injecting a sample of errors at different elements and iterations in a key computation of the solver, i.e., sparse matrix-vector multiplication (SpMV). This sampling allows us to train several classic ML models (i.e., Logistic Regression, Support Vector Machine, Random Forest, K-nearest neighbors [4]) that predict if the slowdown incurred by a soft error striking a particular location will exceed a critical threshold. This model allows for the identification of corresponding elements in the SpMV that will be protected during the execution. Our preliminary results evaluated using two matrices from the SuiteSparse Matrix Collection [3] show that the Random Forest classifier tends to offer the best prediction performance on the testing data, and the resulting dynamic protection

scheme provides a promising resilience solution while reducing the overhead compared to two baseline schemes and two static protection schemes.

2 BACKGROUND AND MOTIVATION

2.1 The PCG Solver

PCG [6] is one of the most widely used algorithms for solving a sparse linear system: $Ax = b$, where A is an $N \times N$ symmetric positive definite sparse matrix, and x and b are $N \times 1$ dense vectors. Algorithm 1 shows the pseudocode. It starts with an initial guess of the solution vector x_0, and at each iteration i, computes an updated solution vector x_i. The algorithm terminates when the solution converges based on a pre-defined residual threshold (tol) or when the specified maximum number of iterations ($maxit$) is reached.

Algorithm 1: Preconditioned Conjugate Gradient (PCG)

```
   Input: A, M, b, x_0, tol, maxit
1  begin
2      r_0 ← b − A x_0                          // Initial residual
3      z_0 ← M^{-1} r_0                         // Preconditioning
4      p_0 ← z_0
5      i ← 0
6      while i < maxit and ||r_i||/||b|| > tol do
7          q_i ← A p_i
8          v_i ← r_i^T z_i
9          α ← v_i / (p_i^T q_i)
10         x_{i+1} ← x_i + α p_i                // Improve approximation
11         r_{i+1} ← r_i − α q_i                // Update residual
12         z_{i+1} ← M^{-1} r_{i+1}             // Preconditioning
13         v_{i+1} ← r_{i+1}^T z_{i+1}
14         β ← v_{i+1} / v_i
15         p_{i+1} ← z_{i+1} + β p_i            // New search direction
16         i ← i + 1
17     end
18 end
```

Among the computations performed in each iteration of PCG, the SpMV operation (Line 7) is the most compute-intensive, taking $O(nnz)$ time, where nnz represents the number of nonzeros in A. SpMV is also the computation that is the most prone to soft errors [7, 11]. Thus, we inject soft errors in SpMV, in particular in the vector p, to study the impacts of such errors on the convergence.

2.2 Impact of Soft Errors

We inject errors at different elements of the p vector (Line 7 of Algorithm 1) and at different iterations of the computation. The impact is measured in terms of the *slowdown*, as defined below:

$$slowdown = I_e / I_o \,, \tag{1}$$

where I_e and I_o denote the number of iterations for the algorithm to converge with and without errors, respectively.

Five sets of experiments are conducted at five different iterations: 2, $0.25I_o$, $0.5I_o$, $0.75I_o$, and I_o. In each set, 100 experiments are run, where each run has an error injected at the given iteration in a random location of the p vector. The resulting slowdowns (sorted in ascending order) for each considered iteration are shown in Figures 1(a)(c) for the sparse matrices *"cbuckle"* and *"bcsstk18"* [3]. For both matrices, if the slowdown of an experiment exceeds 100,

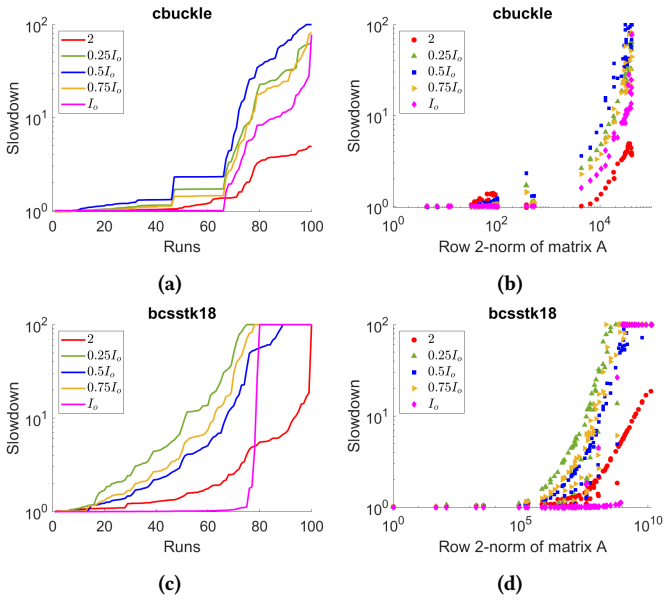


Figure 1: Impact of soft errors for two matrices: *"cbuckle"* (top) and *"bcsstk18"* (bottom). Figures (a)(c) show the slowdowns caused by errors injected in 100 random locations of p vector at five different iterations. Figures (b)(d) show the correlations between the slowdowns and the row 2-norms of the matrices.

we terminate the execution and simply report the slowdown for that experiment as 100.

We observe that not all errors are equal: while some lead to significant slowdowns in the convergence of the algorithm, many others have small or even no impact on the convergence. The impact of an error depends greatly on the location of its occurrence (i.e., which iteration and which element of the p vector). To reduce the resilience overhead, it is therefore desirable to protect only those elements in the locations that will lead to large slowdowns.

The question is how to identify those locations with large slowdowns. Figures 1(a)(c) show that, for the two matrices, errors occurring in the middle iterations tend to result in larger slowdowns than those occurring at the start or end of the computation. At a given iteration, it has been shown [7, 11] that the slowdown caused by an error in the i-th element is strongly correlated with the 2-norm of matrix A at the corresponding row, i.e.,

$$\|A_{i*}\|_2 = \sqrt{\sum_{j=1}^{N} A_{i,j}^2} \,. \tag{2}$$

Figures 1(b)(d) show that such a correlation indeed exists at different iterations for both matrices. Overall, the above results suggest that one could use the iteration number and the matrix's row 2-norm to make good predictions on the impact of a soft error at a given location, which we will leverage in the design of our selective protection strategy.

3 DYNAMIC SELECTIVE PROTECTION

In this work, we assume the use of *system-level* replication as a general-purpose technique to transparently protect the application from soft errors.[1] The protection is done by duplicating the computations corresponding to the protected elements. Any soft errors can then be detected by comparing the results of the duplicate computations and the affected iteration can be re-executed.

3.1 Resilience Overhead

We measure the performance of a protection scheme using the resilience *overhead*. Recall that N denotes the dimension of the solution vector, which represents the number of elements that need to be computed at each iteration. Let n_t denote the number of elements that are protected and hence additionally computed at iteration t. The total number of computed elements at iteration t is therefore given by $N + n_t$. The resilience overhead is defined as the total amount of additional computation that is done throughout the entire execution compared to the error-free run, i.e.,

$$overhead = \frac{\sum_{t=1}^{I_e} \left(N + n_t\right) - I_o N}{I_o N} . \tag{3}$$

There exists a trade-off between the amount of per-iteration protection and the expected slowdown. On the one hand, fully protecting all the elements at all iterations (i.e., $n_t = N$) incurs a large per-iteration overhead but will cause no slowdown (i.e., $I_e = I_o$), since any error will be immediately mitigated. Based on Equation (3), this gives an overall resilience overhead of 100%. On the other hand, protecting no elements (i.e., $n_t = 0$) exposes the application to soft errors, which could result in large slowdowns (i.e., $I_e \gg I_o$) and thus high resilience overhead if errors occur in critical locations.

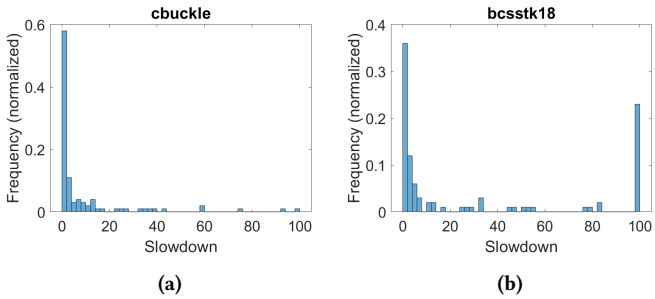
Our goal is to reduce the resilience overhead by selectively protecting those elements at each iteration that will lead to large slowdowns if struck by soft errors. We note that a *static* approach [11] could choose to protect the same number of elements for all iterations, but it may not lead to ideal performance as the impacts of errors happening at different iterations could be different, as shown in Figure 1. In this work, we aim at designing a *dynamic* approach that could protect a varying number of elements at different iterations based on the predicted error impacts.

3.2 ML Prediction of Error Impacts

We apply machine learning (ML) as a way to predict the impact of a soft error, namely, the slowdown, if it occurs in a certain location. Based on the results of Section 2.2, we leverage the following two features in the training of ML models:

- Iteration number;
- Row 2-norm of matrix.

For each matrix we consider, we conduct 300 experiments, and use 200 of them for training and 100 for testing. In each experiment, a soft error is injected at a random iteration chosen uniformly in the range $[1, I_o]$ and at a random element chosen uniformly in the range $[1, N]$.



(a) (b)

Figure 2: The distributions of slowdowns caused by soft errors for the two matrices: *"cbuckle"* (a) and *"bcsstk18"* (b).

As a first attempt, we tried to predict exactly the slowdown induced by a soft error at a particular location by considering it as a *regression* problem. However, the prediction results turned out to be quite poor due to the uneven distribution of the slowdowns, as shown in Figure 2. While many errors tend to result in very small slowdowns (e.g., below 2), or in the case of *"bcsstk18"* relatively large slowdowns (e.g., 100 or higher), the errors covering other slowdown values (e.g., in the median range) are quite scarce, thus leading to bad performance of the regression models.

In fact, since our protection decisions are binary (i.e., either protect an element or not protect an element), we can consider the problem as a *classification* problem by predicting whether the slowdown will exceed a certain threshold, in which case the decision would be to protect the element, and otherwise we would not protect the element. To set the threshold, intuitively, any slowdown that is equal or larger than 2 (i.e., $I_e \geq 2I_o$) implies that the total amount of computation to be done will at least double that of the error-free run, thus inducing an overhead of at least 100% (which can be achieved by the full-protection scheme). Since our objective is to achieve an overhead lower than 100%, we opt to set the threshold at 2, thus having two classes:

- Class 1: $slowdown \geq 2$;
- Class 0: $slowdown < 2$.

We trained four ML models for the above classification problem using the *scikit-learn* library[2]. The following shows four well-known and commonly used classifiers with some important hyperparameters that have been found to work well for our problem using a simple hyperparameter search.

- Logistic Regression (LR): with L2 regularization and an inverse of regularization strength of $C = 1$.
- Support Vector Machine (SVM): with the RBF kernel and an inverse of regularization strength of $C = 1$.
- Random Forest (RF): with 30 estimators (i.e., decision trees) and a maximum tree depth of 200.
- K-Nearest Neighbors (KNN): with the Euclidean distance and $K = 5$ neighbors.

Furthermore, we set a higher weight for Class 1 than for Class 0, with a weight ratio of 2.5:1, for all the ML models except for

[1] We note that some *application-level* resilience techniques for sparse iterative solvers (e.g., by using ABFT [2, 8, 12]) could incur a lower resilience overhead but require tapping into the numerical libraries of the solver. We consider only system-level techniques in this paper.

[2] https://scikit-learn.org/

KNN. This is to increase the prediction accuracy for the errors that would incur a large slowdown (≥2), as the potential cost (in terms of overhead) of failing to protect against those errors is higher compared to mistakenly protecting some additional elements that would incur a small slowdown thus should not be protected.

Table 1 summarizes the prediction results of the four classifiers (in terms of *accuracy*, *recall*, *precision*, and *F1 score*) for the testing datasets of the two matrices ("*cbuckle*" and "*bcsstk18*"). We observe that the performance of the four classifiers is similar for the "*cbuckle*" matrix, which is generally better than the performance for the "*bcsstk18*" matrix. Overall, RF tends to produce consistently good results in terms of the prediction measures for both matrices. Thus, we will use RF as the prediction algorithm for evaluating our dynamic selective protection scheme.

Table 1: Prediction results (accuracy, recall, precision, and F1 score) of four classifiers for two matrices.

	"cbuckle"				*"bcsstk18"*			
	Acc.	Recall	Prec.	F1	Acc.	Recall	Prec.	F1
LR	0.91	0.8	0.97	0.88	0.57	1	0.57	0.73
SVM	0.92	0.8	1	0.89	0.57	1	0.57	0.73
RF	0.93	0.9	0.93	0.91	0.8	0.9	0.78	0.84
KNN	0.94	0.95	0.91	0.93	0.76	0.84	0.76	0.80

3.3 Evaluation Results

Our dynamic ML-based selective protection scheme will protect those elements at each iteration that are predicted to be in Class 1 (i.e., those with a slowdown of at least 2). Its performance in terms of the resilience overhead is shown in Table 2 by averaging over the 100 testing experiments of the two matrices. Also shown in the table is the performance of a few other protection schemes.

Table 2: Average resilience overheads of five protection schemes for two matrices.

	"cbuckle"	*"bcsstk18"*
zero-protection	816%	3143%
full-protection	100%	100%
static-rand (best)	91.4% (89%)	99% (99%)
static-r2n (best)	56.3% (30%)	88.4% (88%)
dynamic-ml	**43.7%**	**87.6%**

First, we can see that our dynamic protection scheme (*dynamic-ml*) has the best performance compared to the other four protection schemes, with a 43.7% overhead for the *"cbuckle"* matrix and a 87.6% overhead for the *"bcsstk18"* matrix.

For the two baseline algorithms, the *zero-protection* scheme protects no elements at all and it performs badly with very high overheads for both matrices, which is due to the severe slowdowns caused by errors in certain critical locations. The *full-protection* scheme also incurs a relatively high overhead of 100% by duplicating the entire computation.

We also evaluated two static selective protection schemes, which protect a fixed percentage of elements at all iterations. Given a protection percentage, *static-rand* [9] randomly selects the corresponding number of elements to protect at each iteration, while *static-r2n* [11] selects the elements that correspond to the largest row 2-norms of the matrix. For both schemes, we experimented with different protection percentages (from 1% to 99%) and chose the one (shown in parenthesis) that leads to the best overhead. The results show that both static schemes are able to achieve a resilience overhead below 100%, with static-r2n offering better performance due to the use of row-2 norms that better correlate with the error impacts. However, these static schemes do not capture the errors' varying behaviors across iterations. Our dynamic scheme overcomes this limitation and achieves lower overhead.

4 CONCLUSION AND FUTURE WORK

In this paper, we conducted an initial study on the use of dynamic selective protection to protect the PCG solver from soft errors. Our preliminary results have shown that the combined use of ML prediction and dynamic element selection offers a promising solution for reducing the resilience overhead at the system level.

We plan to investigate the following directions in our future work: (1) improve the prediction accuracy by exploring more ML algorithms or including additional features (e.g., runtime characteristics of the solution vector); (2) explore alternative thresholds for deciding the protected elements, possibly leveraging analytical models that incorporate error probabilities; (3) further validate our approach by experimenting with more sparse matrices and other iterative solvers (e.g., MINRES, GMRES); (4) train a single ML model that works simultaneously for multiple matrices (e.g., by capturing the sparse matrix structures) rather than one model for each individual matrix (our current approach).

ACKNOWLEDGMENTS

This work is supported in part by the U.S. NSF grant #2135310. Thomas Verrecchia conducted this research while visiting Vanderbilt University.

REFERENCES

[1] Franck Cappello, Geist Al, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. 2014. Toward Exascale Resilience: 2014 Update. *Supercomput. Front. Innov.: Int. J.* 1, 1 (2014), 5–28.
[2] Zizhong Chen. 2013. Online-ABFT: an online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *PPoPP*.
[3] Timothy A. Davis and Yifan Hu. 2011. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.* 38, 1, Article 1 (2011), 25 pages.
[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2001. *The Elements of Statistical Learning.* Springer New York Inc., New York, NY, USA.
[5] Thomas Hérault and Yves Robert (Eds.). 2015. *Fault-Tolerance Techniques for High-Performance Computing.* Springer Verlag.
[6] Y. Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2nd ed.). Society for Industrial and Applied Mathematics, USA.
[7] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. 2011. Characterizing the impact of soft errors on iterative methods in scientific computing. In *ICS*.
[8] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. 2012. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *ICS*.
[9] J. Sloan, R. Kumar, and G. Bronevetsky. 2012. Algorithmic approaches to low overhead fault detection for sparse linear algebra. In *DSN*.
[10] M. Snir and et al. 2014. Addressing Failures in Exascale Computing. *Int. J. High Perform. Comput. Appl.* 28, 2 (2014).
[11] Hongyang Sun, Ana Gainaru, Manu Shantharam, and Padma Raghavan. 2020. Selective Protection for Sparse Iterative Solvers to Reduce the Resilience Overhead. In *SBAC-PAD*.
[12] Dingwen Tao, Shuaiwen Leon Song, Sriram Krishnamoorthy, Panruo Wu, Xin Liang, Eddy Z. Zhang, Darren Kerbyson, and Zizhong Chen. 2016. New-Sum: A Novel Online ABFT Scheme For General Iterative Methods. In *HPDC*.