Detecting Silent Data Corruption in Sparse Matrices using Hardware Performance Counters

Minseop Choi
University of Massachusetts Lowell
Lowell, MA, USA
minseop_choi@student.uml.edu

Orlando Arias
University of Massachusetts Lowell
Lowell, MA, USA
orlando_arias@uml.edu

Seung Woo Son
University of Massachusetts Lowell
Lowell, MA, USA
seungwoo_son@uml.edu

Abstract

High-Performance Computing (HPC) systems frequently execute large-scale sparse matrix computations in scientific and engineering domains. These workloads are susceptible to *Silent Data Corruptions* (SDCs)—undetected faults that can alter results without triggering errors—posing a significant risk to computational integrity. In this work, we show how injected errors in sparse matrices propagate during repeated sparse matrix-vector multiplication (SpMV) executions and evaluate whether hardware performance counter (PMC) patterns can be used to detect such corruptions. We conduct controlled experiments with Gaussian noise injection at varying magnitudes and injection rates, record hardware counter values using the Linux `perf` tool, and train a Decision Tree classifier to distinguish corrupted runs from clean runs. Experiments on four real-world matrices from the SuiteSparse Matrix Collection yield average detection accuracy near 90% (up to 95%) with under 2% runtime overhead. The results confirm that PMC-based classification is a viable approach for lightweight SDC detection.

Keywords

Silent Data Corruption, Sparse Matrices, Error Propagation, Performance Counters, Decision Tree, SpMV

1 Introduction

Sparse matrices are widely used in HPC applications, enabling efficient storage and computation for large-scale problems. Many workloads repeatedly invoke the sparse matrix-vector multiplication (SpMV) kernel, where the accumulation of small numerical errors can influence results over successive iterations.

Silent Data Corruptions (SDCs) [2] are particularly problematic because they occur without warning. SDCs can persist undetected across computational phases. Without detection, corrupted values can propagate and amplify through iterative solvers or repetitive computations, eventually producing significantly incorrect results.

Traditional methods such as Algorithm-Based Fault Tolerance (ABFT) [3, 9, 10] can detect and sometimes correct these faults, but often incur significant computational and memory overhead and require algorithm-specific integration.

This work focuses on observing how a single injected error in sparse matrices propagates during repeated SpMV execution, and on determining whether hardware performance counter (PMC) patterns can be leveraged for detecting SDCs without intrusive changes to the computation when multiple errors are inserted into real data.

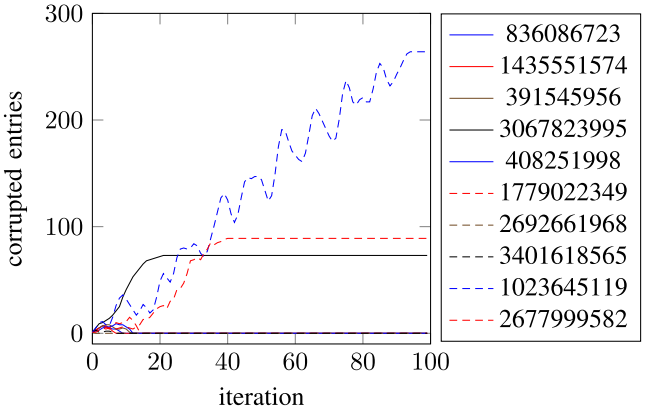

Figure 1: Error propagation on sparse matrices: different initial error locations yield distinct propagation patterns under repeated SpMV.

2 Background

2.1 Error Propagation in CRS Format

We use the Compressed Row Storage (CRS) format, which was designed to encode a sparse matrix and store it in memory [1] while being able to efficiently perform computations with it. Although the CRS format enables efficient SpMV computations, errors can significantly affect the computation results. Corruption in any of these arrays can alter access patterns and results. The proximity of an injected error to nonzero elements directly affects the spread of corruption during repeated SpMV operations.

3 Error Propagation and Detection

The initial position of an injected error in the matrix significantly influenced the rate and extent of propagation during the repeated SpMV algorithm from our previous study [4]. Figure 1 shows how a single error injected into a sparse matrix propagates during SpMV iterations. The seed number lines (e.g., 836086723) show how the number of corrupted entries in a 400 × 400 matrix with a sparsity factor of 0.9975 changes with a single error injected. The location of the error injected with respect to the non-zero values in the matrix has a direct effect on the propagation of the error in the computation. Modern CPUs provide performance monitoring events (PMEs) through the performance monitoring unit (PMU), and typically offer 4-8 performance monitoring counters (PMCs) to record their values [6, 7, 11]. To enable early detection of errors and prevent error propagation, it is essential to investigate the relationship between PMCs values and errors.

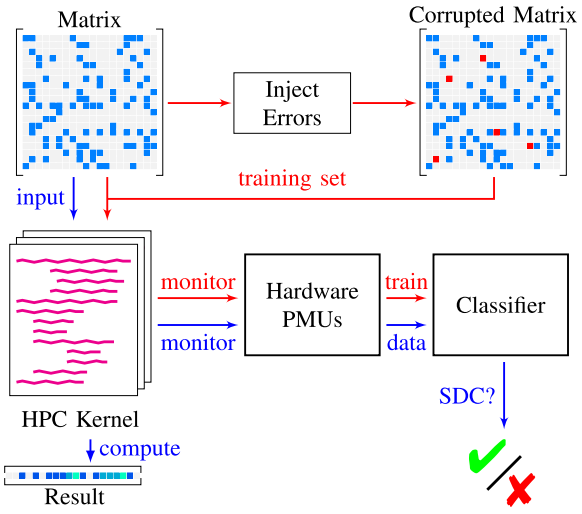

Figure 2: Workflow: training (red arrow in poster) and deployment (blue arrow) phases.

4 Experimental Setup

Platform. Experiments were run on a CloudLab [5] node with two Intel Xeon Silver 4114 CPUs (10 cores each, 2.20 GHz), 192 GiB ECC DDR4-2666 RAM, Ubuntu 22.04.5 LTS, Linux kernel 5.15.0-122.

Datasets. We used four real-world matrices from the SuiteSparse Matrix Collection (See Table 1).

Baseline Measurements. We execute the SpMV once on error-injected matrices, recording hardware performance counter values using the Linux `perf` tool. To ensure sufficient variability and robustness, hardware performance counters were collected 100 times under error-free conditions for each dataset.

5 Methodology

Our workflow is shown in Figure 2:

(1) **Error Injection:** We generated 100 error-injected matrices by varying the error rate e from 0.1 to 1.0 in increments of 0.1 and the injection rate i from 0.01 to 0.10 in increments of 0.01. We perturbed the selected entries with Gaussian noise $\mathcal{N}(\mu, \sigma)$ with mean μ equal to the original value and standard deviation σ to simulate SDC using the error injection module in the ADSP (Anomaly Detection using Sparsity Profile) framework [8].

(2) **Computation:** For each dataset, run SpMV once on each of the 100 error-injected matrices and on each of the 100 error-free runs.

(3) **PMC Collection:** During the training phase (red arrow), all PMCs are used to identify the most important counters using the Linux `perf` tool, and in the deployment phase (blue arrow)), only a few key PMCs are monitored for efficient detection.

(4) **Classification:** Train a Decision Tree classifier with 100 labeled training pairs (event data, label: Error or Non-Error). Use a 70/30 split for training/validation, and apply random oversampling to balance classes.

Table 1: The results of DT classification performance metrics across different dataset.

Dataset	Accuracy	Precision	Recall	F_1 Score
`494_bus.mtx`	0.919	0.919	0.924	0.920
`662_bus.mtx`	0.943	0.950	0.938	0.943
`can_62.mtx`	0.811	0.811	0.822	0.812
`bcspwr03.mtx`	0.892	0.903	0.882	0.890

6 Results

6.1 Detection Performance

Using the full set of 5,670 performance monitoring events listed by perf list, we collected their corresponding counter values as PMC features contain signals of injected errors; when modeled with a Decision Tree, we separated error-injected from error-free runs with accuracies of 0.919, 0.943, 0.811, and 0.892 (<2% overhead), using only 8 selected PMCs in the deployment phase across the four datasets, though the key features varied by dataset.

7 Conclusion and Future Work

We confirmed that PMC data can be used to detect such corruption with high accuracy and minimal overhead. However, the weights of each PME varies by dataset, suggesting the need for further refinement.

Future work will explore additional experiments aimed at stabilizing the importance of each PME across different data conditions (e.g., minimizing time drift when collecting PMC values for error and error-free runs, removing irrelevant PMCs, and verifying that the selected small set of counters maintains stability—measured by reproducibility—across different datasets).

We plan to apply custom `#pragma` directives to instrument HPC workloads, enabling finer-grained PMC data collection and broadcasting for SDC classification on a central node, which is essential for developing a practical, real-time detection system.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 2312982 OAC Core: Improving Data Integrity for HPC Datasets using Sparsity Profile.

References

[1] Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. 2009. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*. 233–244.

[2] Jon Calhoun, Marc Snir, Luke N. Olson, and William D. Gropp. 2017. Towards a More Complete Understanding of SDC Propagation. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. 131–142. doi:10.1145/3078597.3078617

[3] Zizhong Chen. 2013. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. *ACM SIGPLAN Notices* 48, 8 (2013), 167–176.

[4] Minseop Choi, Kyle Chaisson, Orlando Arias, and Seung Woo Son. 2025. Detecting Silent Data Corruption From Hardware Counters. In *2025 IEEE International Conference on Cluster Computing (CLUSTER)*.

[5] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*. 1–14. https://www.flux.utah.edu/paper/duplyakin-atc19

[6] Intel Corporation. [n. d.]. https://perfmon-events.intel.com/.

[7] Intel Corporation 2017. *Intel 64 and IA-32 Architectures Performance Monitoring Events*. Intel Corporation.

[8] Aekyeung Moon, Minjun Kim, Jiaxi Chen, and Seung Woo Son. 2023. Anomaly Detection in Scientific Datasets using Sparse Representation. In *Proceedings of the First Workshop on AI for Systems* (Orlando, FL, USA) *(AI4Sys '23)*. 13–18. doi:10.1145/3588982.3603610

[9] Alexander Schöll, Claus Braun, Michael A Kochte, and Hans-Joachim Wunderlich. 2016. Efficient algorithm-based fault tolerance for sparse matrix operations. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 251–262.

[10] Joseph Sloan, Rakesh Kumar, and Greg Bronevetsky. 2013. An algorithmic approach to error localization and partial recomputation for low-overhead fault tolerance. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 1–12.

[11] Ahmad Yasin. 2014. A Top-Down method for performance analysis and counters architecture. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 35–44. doi:10.1109/ISPASS.2014.6844459