Detecting Silent Data Corruption in Sparse Matrices using Hardware Performance Counters

Minseop Choi Orlando Arias Seung Woo Son

Department of Electrical & Computer Engineering, University of Massachusetts Lowell

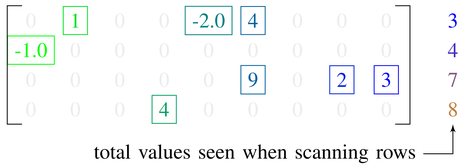
Overview

- High-Performance Computing (HPC) systems perform large-scale numerical computations using sparse matrices in various scientific and engineering applications.
- Silent Data Corruptions (SDCs) are undetected faults that alter results without causing crashes, threatening computational correctness.
- We analyze error propagation in sparse matrices and detect SDCs using Decision Trees (DT) with hardware performance counters (PMCs) collected via the Linux `perf` tool.
- **Proposed Workflow:**
  Error Injection → Computation & Monitoring → PMC Collection (`perf`) → Classifier Evaluation.

Silent Data Corruptions (SDCs)

- Silent Data Corruptions (SDCs) are data errors that occur without being detected by error detection mechanisms, potentially affecting computation correctness.
- SDCs are caused by hardware failures, electromagnetic interference, power fluctuations, and modern low-voltage chip designs make systems even more vulnerable.
- Although Algorithm-Based Fault Tolerance (ABFT) has been used as traditional solutions, they suffer from high overhead and strong dependence on specific algorithms.
- SDC detection requires a robust monitoring approach that is algorithm-independent and incurs minimal overhead.

Error Propagation



(a) Sparse matrix with non-zero coefficients highlighted.

values **v** :	1	-2.0	4	-1.0	9	2	3	4
column index **c** :	1	4	5	0	5	7	8	3
row total **r** :	0	3	4	7	8			

(b) Resulting CRS representation of the matrix.

Figure: CRS storage format. Three linear arrays encode the sparse matrix.

- Compressed Row Storage (CRS) format efficiently stores sparse matrices. Three linear arrays encode the sparse matrix are used: value v, column index c, and row total r.
- Errors in any of its v, c, or r can affect the computation results, and errors in v in particular can lead to SDCs.

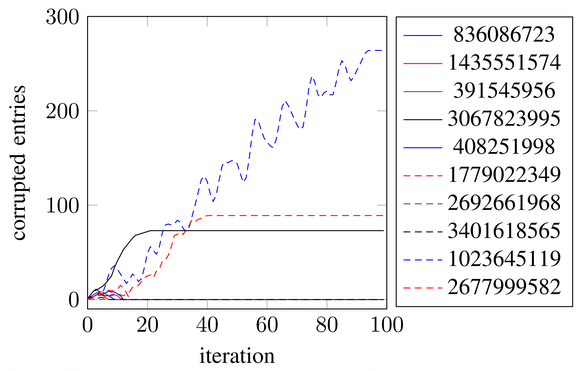


Figure: Error propagation on sparse matrices. Initial position of the error affects the overall propagation of the error.

- From our previous study [2], the figure shows how a single error injected into a sparse matrix propagates during SpMV iterations. The seed number lines (e.g., 836086723) in the figure show how the number of corrupted entries in a 400×400 matrix (sparsity 0.9975) changes with a single error injected. To prevent error propagation, it is essential to investigate the relationship between Performace Monitoring Counters (PMCs) values and errors.
- The location of the error injected with respect to the non-zero values in the matrix directly affects its propagation in the computation.

Process Overview

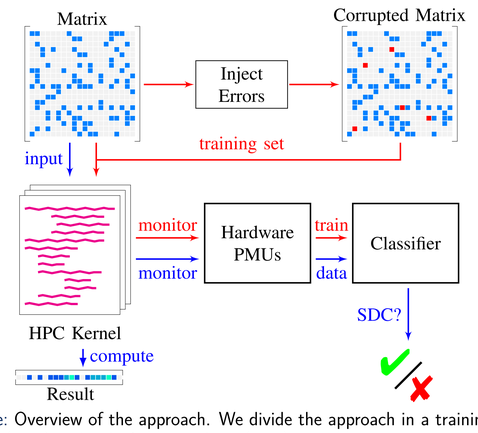


Figure: Overview of the approach. We divide the approach in a training phase (——) and a deployment phase (——).

- Error Injection
  Generate 100 error matrices $\mathbf{Z}_i \sim \mathcal{N}(\mu, \mathbf{X})$ by Gaussian noise with error rate (Error bound for the error injection) $e \in [0.1, 1.0]$, and injection rate (Frequency of the error injection) $i \in [0.01, 0.1]$.
- Computation & PMC Collection
  - Run SpMV once on 100 error-free matrices and 100 error-injected matrices.
  - Collect 5,670 PMCs using the Linux `perf` subsystem.
  - During the training phase (——), all PMCs are used to identify the most important counters, and in the deployment phase (——), only a few key PMCs are monitored for efficient detection.
- Classifier Evaluation
  - Train classifiers to classify runs as Error or Non-Error.
  - Use 70/30 train-validation split with random oversampling to balance classes.
  - We choose DT classification for its high accuracy and low inference overhead compared to other methods.

Experiment

- Hardware counters collected on a single CloudLab [3] node
  - Two Intel Xeon Silver 4114 10-core CPUs @ 2.20,GHz.
  - 192,GiB ECC DDR4-2666 RAM.
  - Ubuntu 22.04.5 LTS, kernel 5.15.0-122-generic.
- 100 noisy matrices generated using combinations of error rate and injection rate
- 4 real-world matrices selected from the SuiteSparse Matrix Collection (e.g., `494_bus.mtx`)

Datasets	Size (rows × cols)	Sparsity	Type
`494_bus.mtx`	494 × 494	0.993	Real-world
`662_bus.mtx`	662 × 662	0.994	Real-world
`can_62.mtx`	62 × 62	0.943	Real-world
`bcspwr03.mtx`	118 × 118	0.965	Real-world

Results

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 across the four datasets (<2% overhead) using only 8 selected PMCs in the deployment phase (as shown in the figure).

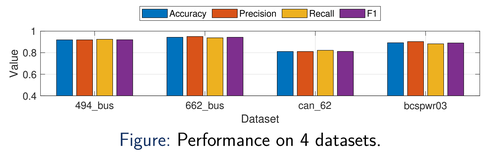


Figure: Performance on 4 datasets.

The 8 key PMCs used for detection are listed below.

- `mem_inst_retired.all_loads`
- `br_misp_exec.indirect`
- `br_inst_retired.not_taken`
- `br_inst_retired.near_taken`
- `br_inst_retired.conditional`
- `br_inst_retired.cond`
- `br_inst_retired.all_branches_pebs`
- `mem_inst_retired.any`

Conclusion & Future Work

- Our study demonstrates that error propagation in sparse matrices significantly depends on the initial location of the injected error, as shown in our propagation analysis.
- We confirmed that PMC data can be used to detect such corruption with high accuracy and minimal overhead. However, since the weights of each PMC vary across datasets, the lower performance on some datasets suggests that further improvements are needed.
- Future work will explore additional experiments aimed at stabilizing the importance of each PME across different data conditions (e.g., 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.

References

[1] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson. 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*, pages 233–244, 2009.

[2] M. Choi, K. Chaisson, O. Arias, and S. W. Son. Detecting silent data corruption from hardware counters. In *2025 IEEE International Conference on Cluster Computing (CLUSTER)*, 2025.

[3] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra. The Design and Operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[4] A. Moon, M. Kim, J. Chen, and S. W. Son. Anomaly Detection in Scientific Datasets using Sparse Representation. In *Proceedings of the First Workshop on AI for Systems*, AI4Sys '23, page 13–18, 2023.

Additional Information

Additional information can be found in the project's open-source repository at `github.com/swson/ADSP`.

Acknowledgements

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**.