Simultaneous Evaluation of Mindful Fault Checking Across the CPU and GPU

Hayden Estes, Dr. Joshua Booth   University of Alabama in Huntsville

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Introduction

This work presents an overhead analysis for the Sparse Preconditioned Conjugate Gradient (PCG) fault tolerance algorithm that utilizes many-core and GPU systems. The work analyzes the overhead selectively using GPUs for duplicate calculation, where the duplication is based on the numerical properties of the sparse matrix. Implementing fault-checking methods from [1] and [2], we rigorously test this approach on real hardware to ensure the reliability and accuracy of linear system solutions. By leveraging existing fault-checking techniques, we validate calculations and address potential numerical instabilities or precision-related issues during iterative solving. Through extensive experimentation on real hardware, we demonstrate the effectiveness of the conjugate gradient algorithm in providing accurate and reliable solutions for large linear systems.

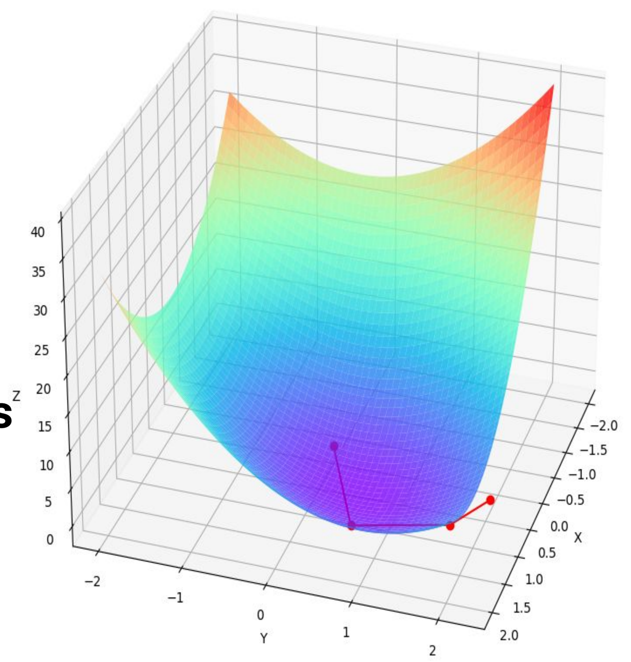
Background

- The **Preconditioned Conjugate Gradient algorithm** (PCG) is an iterative solver **for x in Ax=b**. Many studies and simulations use PCG, such as computational physics, quantum mechanics, and data science.

- The matrices' **row 2-norms** are utilized to identify values in the computation that are **particularly sensitive to numerical faults**. Normally overhead is spent to make duplicate runs for these values. Here, we analyze the **overhead** costs if **duplicated runs are done on GPU** while the **main runs are done on CPU.** [1]

- **Selectively protecting** the critical SpMV of PCG, **A*p=q**, is a sufficient check as all faults will eventually propagate through vector p [2].

- The selective protection method strategically focuses on monitoring the **maximum k elements of the row 2-norms of matrices**, ensuring accuracy in the CG algorithm. [1]



Methodology

- We developed **dual versions of the PCG algorithm** in C: one with custom math functions designed for the **CPU**, and another with **GPU** acceleration using cuBLAS and cuSPARSE.

- We ran both of these implementations **parallel** 100 times for each matrix. This test was run with **error injections to the p vector** as described in [2] with **no** protection, **full** protection, and **selective** protection.

All tests were run on an Ubuntu 20.04.5 x86_64 system with an Intel Xeon Silver 4210R and NVIDIA Quadro RTX 4000.

Id	Matrix	N	nnz	Density
1	t2dah_e	11445	176117	0.13 %
2	Bcsstk 18	11948	149090	0.1 %
3	cbuckle	13681	676515	0.36 %
4	Pres Poisso	14822	715804	0.33 %
5	gyro_m	17361	340431	0.11 %
6	nd6k	18000	6897316	2.1 %
7	bodyy5	18589	128853	0.037 %
8	raefsky4	19779	1316789	0.34 %
9	Trefethen_2	20000	554466	0.14 %
10	msc23052	23052	1142686	0.22 %
11	bcsstk36	23052	1143140	0.22 %
12	wathen100	30401	471601	0.051 %
13	vanbody	47072	2329056	0.11 %
14	cvxbqp1	50000	349968	0.014 %
15	ct20stif	52329	2600295	0.095 %
16	thermal1	82654	574458	0.0084 %
17	m_tl	97578	9753570	0.1 %
18	2cubes_sph	101492	1647264	0.016 %
19	G2_circuit	150102	726674	0.0032 %
20	pwtk	217918	11524432	0.024 %

Figure 1.
Our selection of sparse matrices.

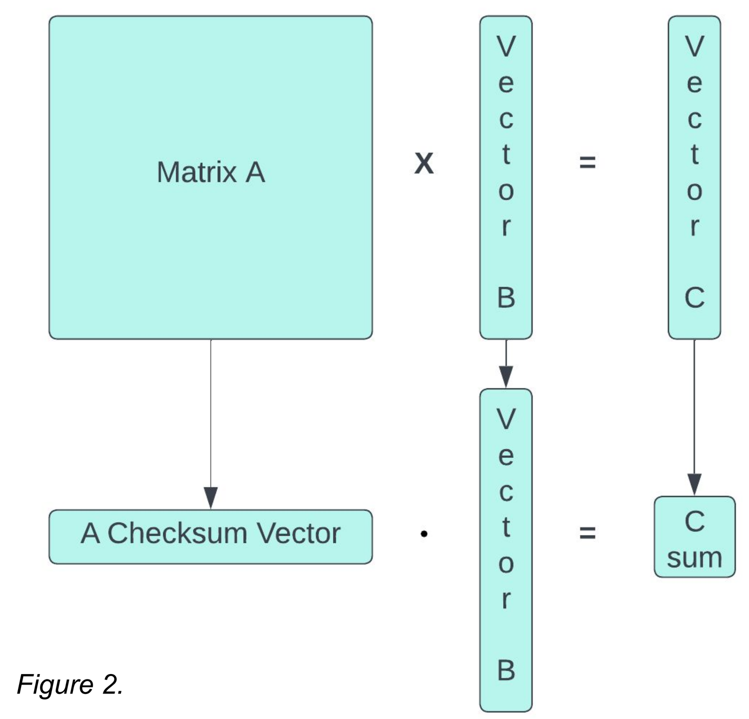


Figure 2.
The S_ABFT method for SpMV as described in [1].

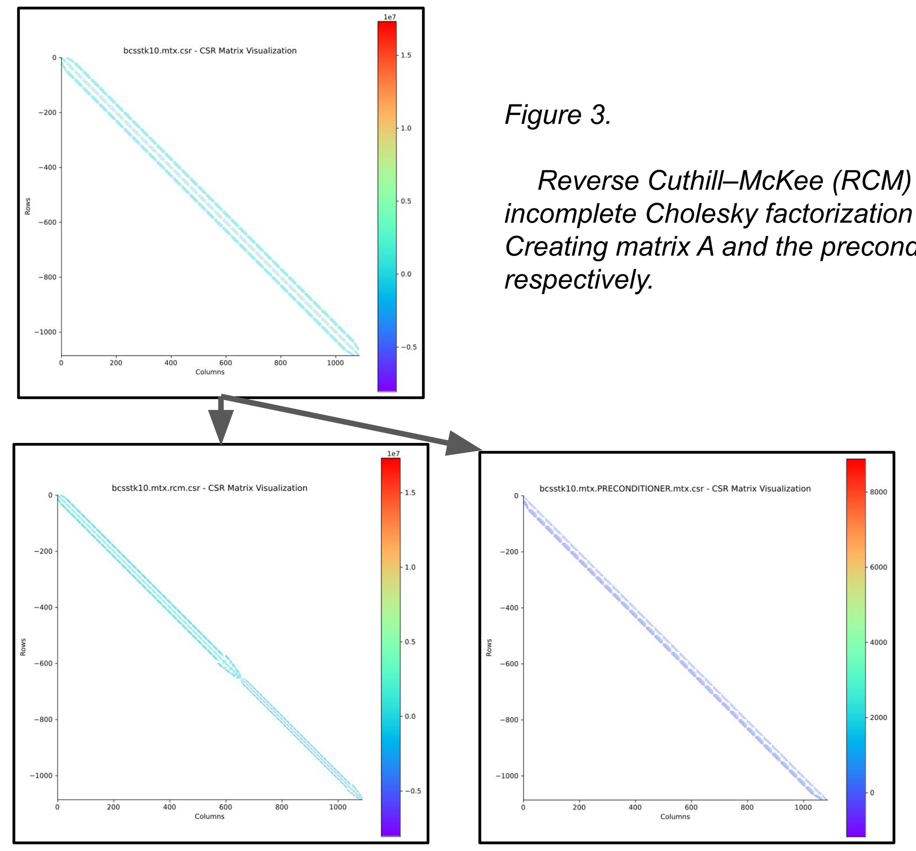


Figure 3.
Reverse Cuthill–McKee (RCM) and incomplete Cholesky factorization (ICHOL) Creating matrix A and the preconditioner respectively.

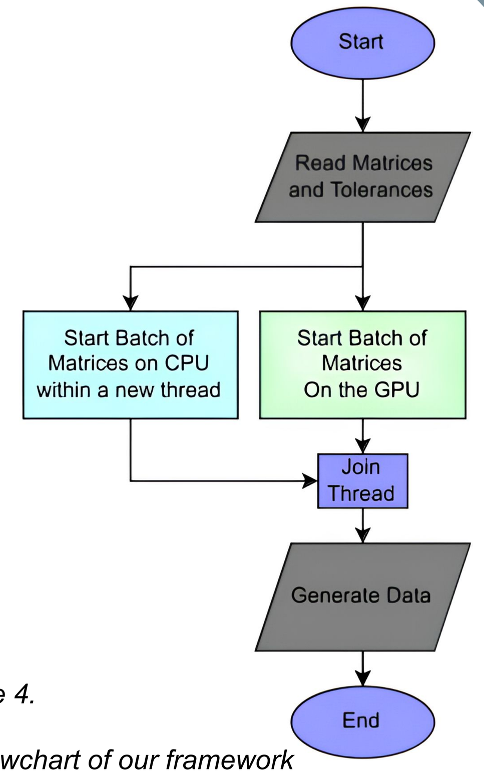


Figure 4.
Flowchart of our framework

Results

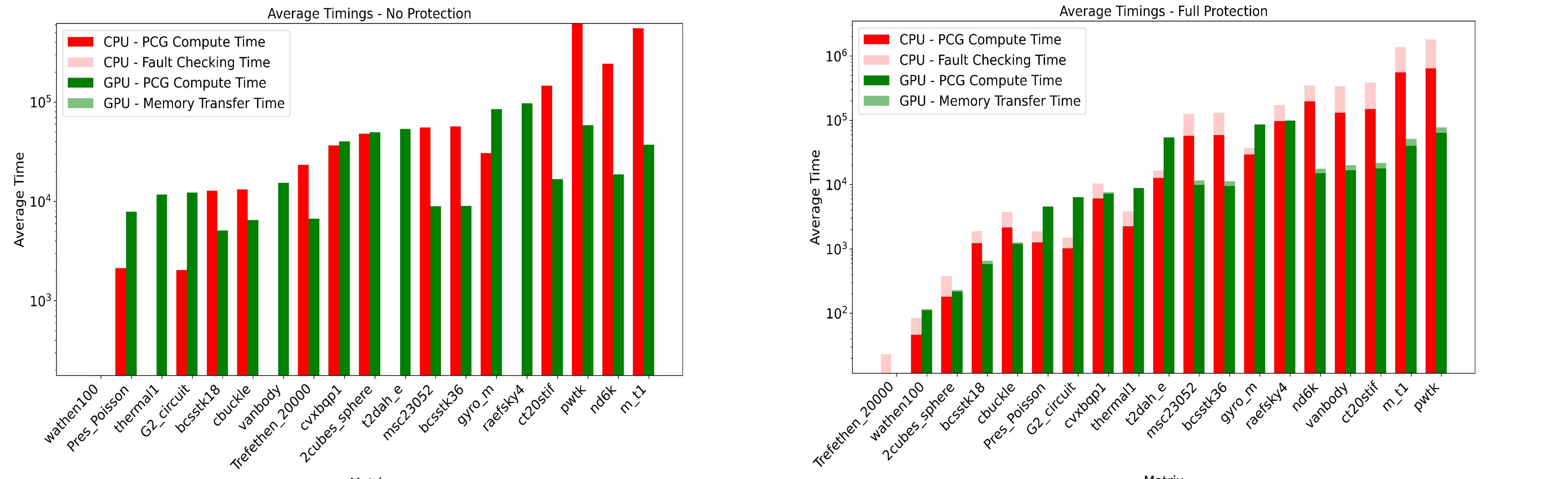


Figure 5.
Bar graphs showing average wall times for each matrix, with and without fault protection.

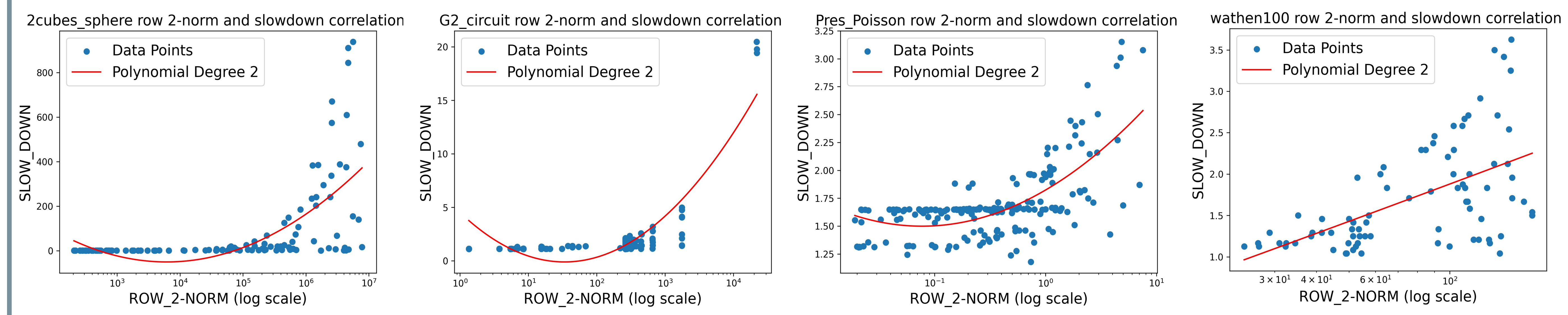


Figure 6
Scatter plots representing the correlation between row 2-norm and slowdown.

Discussion and Continuation

I. The **methods presented in [1] and [2]** can be accurately represented outside of a simulation.
II. Our frameworks dual CPU-GPU computation is able to **reduce idling hardware**.
III. Our **fault tolerance** methods accurately **detect, correct, and reduce computation** time when presented with faults.

Optimizing **fault tolerance** and **memory transfers** in the PCG algorithm for **dual CPU-GPU Computation**, and **Selective Protection** enables ***efficient large-scale PCG on otherwise idle CPUs, enhancing supercomputer compute capacity.***

We see opportunities to continue this work with mixed precision, determining k values from sampling tests, custom CUDA kernels, and energy saving measuring.

References

[1] H. Sun, A. Gainaru, M. Shantharam and P. Raghavan, "Selective Protection for Sparse Iterative Solvers to Reduce the Resilience Overhead," *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Porto, Portugal, 2020, pp. 141-148, doi: 10.1109/SBAC-PAD49847.2020.00029.

[2] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. 2012. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In Proceedings of the 26th ACM international conference on Supercomputing (ICS '12). Association for Computing Machinery, New York, NY, USA, 69–78. https://doi.org/10.1145/2304576.2304588

NSF Grant: Award Abstract 2135310, Collaborative Research: SHF: Small: Learning Fault Tolerance at Scale