

Generalized Randomized Asynchronous Linear Solver with Randomized Preconditioner on Massively Parallel Architectures

Balint M. Nyakas¹, Dennis Giannacopoulos¹ and Zeljko Zilic¹

¹McGill University, Montreal, H3A 0G4 Canada, balint.nyakas@mail.mcgill.ca

Asynchronous linear solvers and preconditioners have gained significant attention as effective strategies to mitigate synchronization bottlenecks in computational processes, particularly prevalent in the field of computational electromagnetics (CEM). These solvers and preconditioners have been demonstrated to enhance the efficiency of solving large-scale linear systems, which are frequently encountered in CEM applications, such as modeling electromagnetic wave propagation and scattering. In this study, we introduce a mixed-precision randomized preconditioner implemented on GPUs, alongside a generalized row-based randomized asynchronous linear solver designed to leverage GPU capabilities. Reduced-precision preconditioners are well-established for their ability to accelerate computations while maintaining convergence properties, making them particularly suitable for the large, sparse systems characteristic of CEM problems. This approach offers a promising pathway to address the computational challenges inherent in high-frequency electromagnetic simulations and large-scale system analyses.

Index Terms—Iterative Algorithms, Linear Systems, Parallel Algorithms, Relaxation Methods.

I. INTRODUCTION

LINEAR SYSTEMS are central to numerous scientific and engineering applications, spanning fluid dynamics signal processing and computational electromagnetics, as shown in [1]-[2]. Solving these systems, which are fundamental in any implicit Finite Element electromagnetics solver, efficiently is critical as the size of problems grow in tandem with advancements in High-Performance Computing. Traditional methods whether direct or iterative, often rely on synchronization among computational processes. While it ensures consistency of the data, it can impose significant communication overheads, particularly in parallel and distributed systems [3].

Asynchronous linear solvers have emerged as a promising alternative to circumvent the bottlenecks associated with synchronization [4]-[5]. By allowing calculations to proceed without waiting for other ones to complete, these solvers exploit modern architectures' parallelism capabilities more efficiently. They are especially attractive in heterogeneous environments, where varying processor speeds and communication latencies can slow down synchronized algorithms. They offer the potential to reduce idle times, balance workloads, and achieve faster convergence. However, asynchronous solvers are not without their challenges. They can suffer from numerical instability, slower convergence or divergence in some cases. Addressing these issues requires novel preconditioning techniques [6].

In this work, we generalize a row-based asynchronous linear solver with different non-uniform distributions with a mixed-precision randomized preconditioner utilizing heterogeneous computing resources, including CPUs and GPUs. We evaluate the performance of the developed method compared to other linear solvers on a collection of benchmark datasets. We empirically estimate the convergence rate of the solver on different datasets. Furthermore, we examine how employing mixed-precision arithmetic influences the accuracy and computational performance of the linear solver, particularly in terms of convergence rate and time-to-solution.

II. METHODOLOGY

A. Mixed-Precision Randomized Preconditioner

Preconditioning is a technique for improving convergence rates for iterative solvers. Traditional preconditioners often rely on computationally expensive constructions. In recent years, randomized algorithms have emerged as a powerful alternative, offering efficient and scalable approaches for preconditioners. They leverage probabilistic techniques to approximate key properties of the matrix, providing reduced computational overhead while maintaining accelerated convergence. In [7], a mixed-precision preconditioner has been proposed for least squares with great success. We aim to extend this work for asynchronous linear solvers.

The procedure for constructing the preconditioner is as follows. First, a Gaussian random matrix, $\mathbf{G}$, is generated. At this stage, both the input matrix $\mathbf{A}$ and the Gaussian random matrix can be converted to lower-precision formats, such as fp16, tf32 (TensorFloat-32) or fp32. Beginning with NVIDIA's Ampere architecture, fp32 and tf32 formats are natively supported on NVIDIA GPUs. The input matrix $\mathbf{A}$ is then sketched through multiplication with the random matrix, resulting in $\mathbf{A} * \mathbf{G}$. A sketch of a matrix $\mathbf{A}$ is another matrix $\mathbf{B}$ which is significantly smaller than $\mathbf{A}$ but still approximates it well. The preconditioner is subsequently derived from the triangular factor of the economy QR factorization, as in [7]. If necessary, the sketched matrix can be promoted to higher precision prior to the factorization. The numerical experiments conducted in [7] demonstrate that convergence remains unaffected for the least square solver when using lower-precision tf32, with a runtime comparable to fp16.

This approach is particularly well-suited for GPU acceleration due to the high computational intensity of QR factorization and matrix-matrix multiplication, especially when performed with tf32. Furthermore, the method enables an alternative parallelization strategy. However, this alternative requires further experimental evaluation.

B. Asynchronous Linear Solver

A novel block-based and row-based asynchronous linear solver for the two-dimensional Laplacian using a five-point stencil was introduced in [9]. The row-based solver with non-uniform distribution demonstrated a 10% reduction in iteration count compared to the Gauss-Seidel method, along with excellent parallel performance. This work aims to generalize the method to solve arbitrary $\mathbf{Ax} = \mathbf{b}$ equations, explore multiple non-uniform distributions, and investigate variations in ranking periodicity. Additionally, GPU acceleration and AVX-512 vectorization on CPUs are tested to enhance performance.

The algorithm is based on selecting rows to relax based on a probability distribution function and moving towards the selected rows following the shortest distance. A shared array is used to keep track of row availabilities. Row-sum difference is used for ranking instead of the residuals due to 7 times fewer floating-point operations [9]. The exact algorithm is shown, as seen in [9].

Require: probability distribution function f , ranking period τ , number of rows n

- 1: Set row-sum differences $\Delta = \{\delta_j = N_{\max} \mid j = 1, \dots, n\}$
- 2: Set row ranking R as ascending natural ordering
- 3: Set sorted rows $S = (1, 2, 3, \dots, n)$
- 4: Set $c = 0$
- 5: **for each thread do**
- 6: Set $r_v \in \{1, 2, \dots, n\}$ for initial thread target row
- 7: **for** $i = 1, 2, \dots$ **until convergence do**
- 8: Set previous target as new start row $\tilde{r} = r_v$
- 9: Set target row r_v from sorted rows S using f
- 10: Compute d_v
- 11: **for** $j = 1, \dots, d_v$ **do**
- 12: $\tilde{r} = \text{next}(\tilde{r}, j)$
- 13: **if** $\tilde{r}$ **is free then**
- 14: Perform a relaxation of $\tilde{r}$
- 15: Update $\tilde{r}$
- 16: Compute δ_f
- 17: Set $c = c + 1$
- 18: **if thread is master and** $(c \bmod \tau) = 0$ **then**
- 19: Update R and S based on Δ
- 20: **end if**
- 21: **end if**
- 22: **end for**
- 23: **end for**
- 24: **end for**

The quality and effectiveness of an iterative method are determined by its convergence rate. Although the convergence rate does not directly represent the total time to solution—since the computational cost per iteration may vary across different iterative methods—it serves as a reasonable indicator of the method’s effectiveness for a specific input.

This process has proven challenging for asynchronous methods, as the concept of a single iteration becomes ambiguous. In order to provide convergence estimates for the proposed asynchronous linear solver for the different numerical experiments, we aim to use the method used in [8]. The core idea is to construct an estimate by applying linear regression to the solver’s residual history. This approach has been validated for GMRES, CG, and BiCGSTAB methods with different systems.

III. PRELIMINARY RESULTS

We aim to emphasize the critical role of mixed-precision techniques in the preconditioner implementation. As demonstrated in [10], a considerable variety of novel precision options are available, offering tailored choices for various applications in scientific computing. The QR factorization of the matrix, which constitutes the most computationally intensive operation in the preconditioner calculation, serves as a focal point for this analysis. By evaluating the runtime performance across various matrix sizes using fp32 and tf32 precisions on a GPU, we demonstrate the efficiency gains achieved through the application of mixed-precision operations. We ran QR decomposition on 3 different Hilbert matrices with the size of 100, 1000 and 10000 rows and columns. We also averaged 5 runs for each combinations. Results are shown in Table I.

TABLE I
QR DECOMPOSITION PERFORMANCE WITH TF32 AND FP32 PRECISION

matrix size / precision	tf32	fp32
100x100	0.677s	0.689s
1000x1000	0.94s	0.96s
10.000x10.000	178.46	179.71s

The experiments were conducted utilizing an Nvidia 3090 graphics card. As the matrix size increases, the efficiency gains associated with tf32 precision become increasingly significant. Moreover, the matrix dimensions analyzed in this study are substantially smaller than those encountered in real-world engineering applications. However, since the preconditioner is recomputed during each iteration, even these efficiency improvements can result in significant time savings, potentially reducing the solution time by several minutes. The performance of the developed solver will be further evaluated on larger-scale problems to assess its applicability and efficiency in such contexts.

REFERENCES

- [1] Dehnavi, M., Fernández, D. & Giannacopoulos, D. Finite-element sparse matrix vector multiplication on graphic processing units. *IEEE Transactions On Magnetics*. **46**, 2982-2985 (2010)
- [2] Dehnavi, M., Fernández, D. & Giannacopoulos, D. Enhancing the performance of conjugate gradient solvers on graphic processing units. *IEEE Transactions On Magnetics*. **47**, 1162-1165 (2011)
- [3] Anzt, H., Tomov, S., Gates, M., Dongarra, J. & Heuveline, V. Block-asynchronous multigrid smoothers for GPU-accelerated systems. *Procedia Computer Science*. **9** pp. 7-16 (2012)
- [4] Avron, H., Druinsky, A. & Gupta, A. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. *Journal Of The ACM (JACM)*. **62**, 1-27 (2015)
- [5] Chazan, D. & Miranker, W. Chaotic relaxation. *Linear Algebra And Its Applications*. **2**, 199-222 (1969)
- [6] Frommer, A. & Szyld, D. On asynchronous iterations. *Journal Of Computational And Applied Mathematics*. **123**, 201-216 (2000)
- [7] Georgiou, V., Boutsikas, C., Drineas, P. & Anzt, H. A mixed precision randomized preconditioner for the LSQR solver on GPUs. *International Conference On High Performance Computing*. pp. 164-181 (2023)
- [8] Nayak, P. & Anzt, H. A Probabilistic Model for Asynchronous Iterative Methods. *2024 IEEE International Parallel And Distributed Processing Symposium Workshops (IPDPSW)*. pp. 260-269 (2024)
- [9] Jensen, E., Coleman, E. & Sosonkina, M. Implementing asynchronous linear solvers using non-uniform distributions. *Journal Of Simulation Engineering*. **2** (2020)
- [10] Dongarra, J., Gunnel, J., Bayraktar, H., Haidar, A. & Ernst, D. Hardware Trends Impacting Floating-Point Computations In Scientific Applications. *ArXiv Preprint ArXiv:2411.12090*. (2024)