Distributed and GPU-Accelerated Conjugate Gradient Solver

Joshua Dover

December 13, 2025

1 Introduction

The Poisson equation is a fundamental elliptic partial differential equation found in applications such as heat flow, electrostatics, and incompressible fluid dynamics. When discretized on a uniform grid with the five-point Laplacian stencil, the resulting linear system is large, sparse, and symmetric positive definite, making the Conjugate Gradient (CG) method an appropriate solver.

This project progressively develops a high-performance CG solver across four architectures: a serial CPU baseline, a distributed-memory MPI implementation, a GPU-accelerated version, and finally a fully distributed MPI+GPU solver. The goal is to examine how performance, communication costs, and memory-bandwidth limitations evolve across these phases. We evaluate strong scaling on up to 8 MI210 GPUs, analyze a roofline model of the stencil kernel, and compare end-to-end speedup across implementation stages.

2 Mathematical Formulation

2.1 PDE and Discretization

We consider the Poisson equation

$$-\Delta u(x,y) = f(x,y), \qquad (x,y) \in \Omega = [0,1]^2,$$

with Dirichlet boundary conditions $u = 0$ on the boundary $\partial\Omega$. The domain is discretized using a uniform $N \times N$ grid with spacing

$$h = \frac{1}{N-1}.$$

The Laplacian at interior points is approximated using the standard five-point finite difference stencil

$$-\Delta_h u_{i,j} = \frac{4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}}{h^2}.$$

This yields a sparse linear system

$$Au = b,$$

where the matrix A corresponds to the discrete negative Laplacian operator $-\Delta_h$. The matrix A is symmetric positive definite (SPD). Symmetry follows from the structure of the stencil, while positive definiteness follows from the discrete gradient identity

$$v^T A v = \|\nabla_h v\|_2^2 > 0,$$

for any nonzero vector satisfying the Dirichlet boundary conditions.

Three types of right-hand sides were implemented in this project. First, a single Laplacian eigenfunction

$$u_1(x,y) = \sin(\pi x)\sin(\pi y),$$

with

$$f_1(x, y) = 2\pi^2 u_1(x, y),$$

provides a manufactured solution and yields exact one-iteration convergence in CG. Second, a linear combination of several eigenmodes is used to provide a more complex forcing term while still admitting an exact solution. Finally, for performance studies a smooth Gaussian bump is used,

$$f(x, y) = \exp\big(-50\big[(x - 0.5)^2 + (y - 0.5)^2\big]\big),$$

which represents a localized forcing term and leads to realistic CG iteration counts even though no closed-form solution is available.

2.2 Conjugate Gradient Method

The linear system $Au = b$ is solved using the CG method, an iterative solver well suited to large sparse systems arising from elliptic PDEs. Starting from an initial guess $u_0 = 0$, the residual and search direction are initialized as

$$r_0 = b - Au_0, \qquad p_0 = r_0.$$

The core CG iteration updates the solution, residual, and search direction via

$$\begin{aligned}
\alpha_k &= \frac{r_k^T r_k}{p_k^T A p_k}, \\
u_{k+1} &= u_k + \alpha_k p_k, \\
r_{k+1} &= r_k - \alpha_k A p_k, \\
\beta_{k+1} &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}, \\
p_{k+1} &= r_{k+1} + \beta_{k+1} p_k.
\end{aligned}$$

The iteration terminates once the relative residual norm

$$\frac{\|r_k\|_2}{\|r_0\|_2}$$

falls below a tolerance of 10^{-8}.

Although a Jacobi preconditioner was included in the initial project plan, it was not used in the final solver. For the five-point Laplacian, the diagonal of A is constant, so M^{-1} reduces to a scalar multiple of the identity. In this case, preconditioned CG generates the same search directions as the unpreconditioned method and provides no reduction in iteration count, while adding extra kernel and memory operations.

3 Implementation

3.1 MPI Domain Decomposition

The Poisson problem is parallelized using row-wise slab decomposition across MPI ranks. Each rank owns a contiguous block of interior rows of the global $N \times N$ grid, along with a single halo row above and below to store ghost values. This layout preserves contiguous memory access and minimizes communication, since only nearest-neighbor boundary rows must be exchanged.

In each iteration, halo rows are exchanged with neighboring ranks using nonblocking point-to-point communication and Dirichlet boundary conditions are enforced by setting halo values to zero at physical domain boundaries. The halo exchange is performed prior to applying the discrete Laplacian, ensuring that stencil evaluations reference consistent neighbor data.

In addition to halo exchange, CG requires global reductions for inner products, formed by combining local contributions across ranks using `MPI_Allreduce`. Although the communication volume is small, these collective operations introduce synchronization points that ultimately limit strong scaling as the number of ranks increases.

3.2 GPU Acceleration

In the GPU-accelerated phases, each MPI rank is mapped to a single GPU and offloads the computationally intensive components of the CG iteration. Device memory is allocated for the solution vector u, residual r, search direction p, and stencil result Ap. The five-point Laplacian is the primary computational operation, performing most of the floating-point work per iteration, and is implemented using a HIP kernel over the rank's local slab.

All vector updates are performed on the GPU, keeping CG data resident on the device throughout the iteration loop and avoiding unnecessary host–device transfers.

Dot products are evaluated in two stages. A GPU reduction first computes the local inner product over the rank's owned interior rows, after which the resulting scalar is copied to the host and combined across ranks using `MPI_Allreduce`. This approach limits global communication to a single scalar per dot product while keeping all CG vectors resident on the GPU.

The primary distinction between the GPU-only phase and the distributed GPU phase lies in the handling of halo exchanges. In the GPU-only phase, halo exchange is performed on host memory prior to kernel launches. In the distributed GPU phase, the implementation assumes GPU-aware MPI, which was supported on the target system and verified by eliminating explicit device–host copies. This enables a largely GPU-resident iteration loop, apart from the unavoidable global reductions.

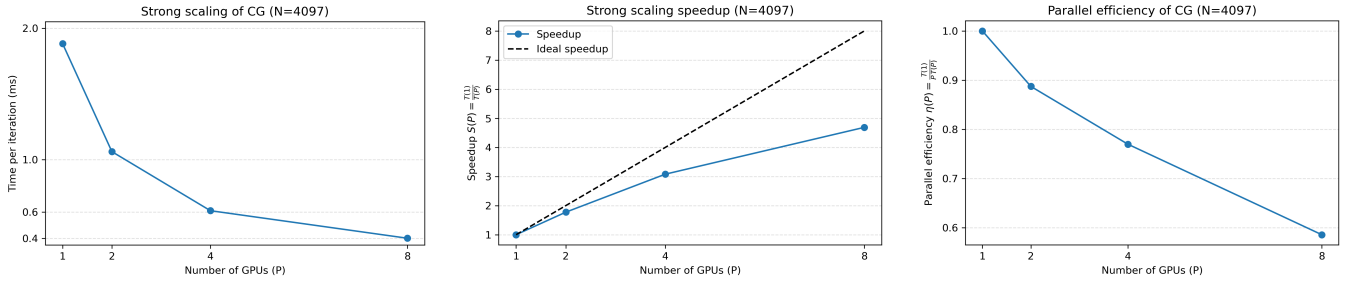
3.3 Validation and Performance Metrics

Correctness was validated by confirming convergence to the same relative residual tolerance (10^{-8}) across all architectures. Accuracy for manufactured solutions was assessed by computing L^2 and maximum errors against the known exact solution. Performance experiments using a Gaussian forcing term characterized convergence through iteration counts and residual norms, while parallel performance was evaluated using time per iteration, strong scaling speedup, and parallel efficiency.

4 Results and Discussion

4.1 Strong Scaling

Strong scaling performance was evaluated for a fixed problem size of $N = 4097$ by measuring time per iteration, speedup, and parallel efficiency as the number of GPUs increased.



(a) Time per iteration (b) Speedup $S(P)$ (c) Efficiency $\eta(P)$

Figure 1: Strong scaling of CG for fixed $N = 4097$.

As the number of GPUs increases, the time per iteration decreases monotonically, though speedup is sublinear. Parallel efficiency drops from near-ideal at $P = 2$ to approximately 60% at $P = 8$, indicating increasing communication and synchronization overheads.

4.2 Convergence

Figure 2 shows the number of CG iterations required to reach a relative residual tolerance of 10^{-8} as a function of grid size for the CPU implementation. The iteration count remains constant across all architectures for a fixed grid size and grows approximately linearly with N.

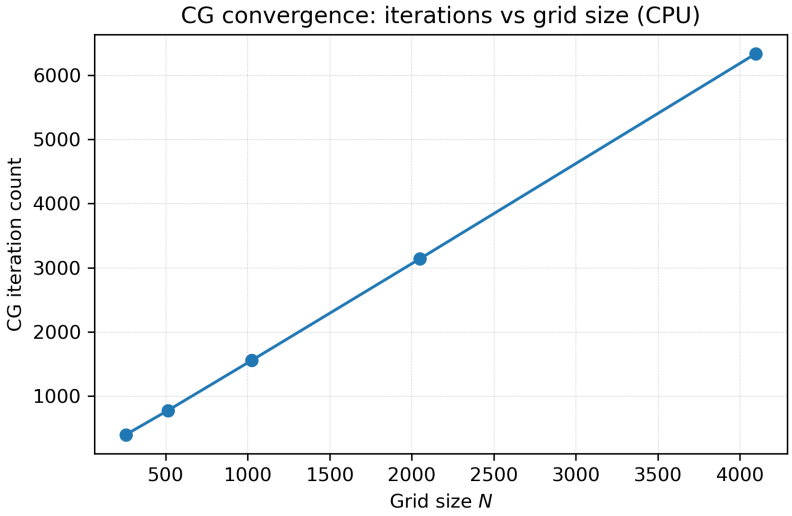


Figure 2: CG iteration count vs grid size.

4.3 Roofline Performance

Figure 3 places the `apply_A` stencil kernel on a roofline model for a single MI210 GPU.

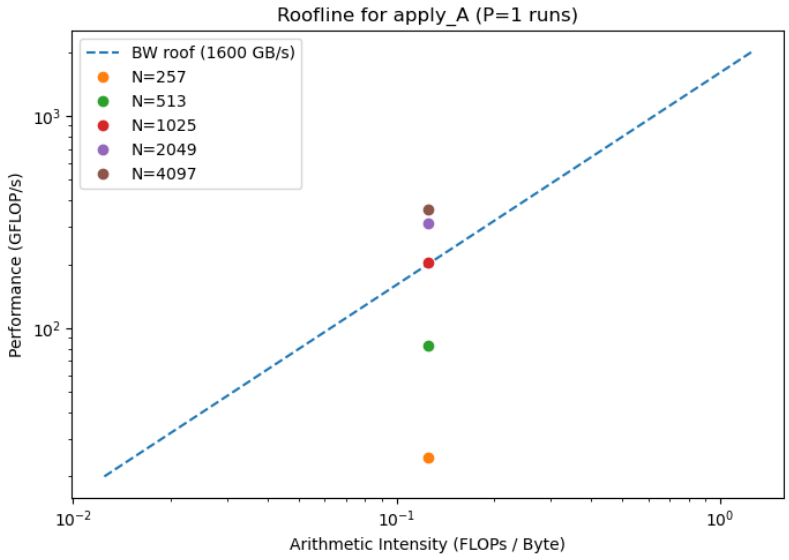


Figure 3: Roofline model for the apply_A stencil kernel on a single MI210 GPU (P=1).

For the five-point Laplacian stencil, the arithmetic intensity is inherently low. For each interior grid

point, the `apply_A` kernel performs five double-precision loads (the center value and four neighbors), one double-precision store, and approximately six floating-point operations. Under a conservative no-reuse traffic model, this corresponds to an arithmetic intensity of

$$\frac{6 \text{ FLOPs}}{6 \times 8 \text{ bytes}} \approx 0.125 \text{ FLOPs/byte}.$$

Even with some effective cache reuse, the arithmetic intensity remains low, placing the kernel firmly in the memory-bandwidth-bound regime on modern GPU architectures.

4.4 Runtime Breakdown

Although the stencil application dominates the computational workload of each CG iteration, it does not dominate wall-clock time at scale. As the number of GPUs increases, the cost of global dot-product reductions becomes the primary runtime bottleneck due to synchronization and communication overhead.

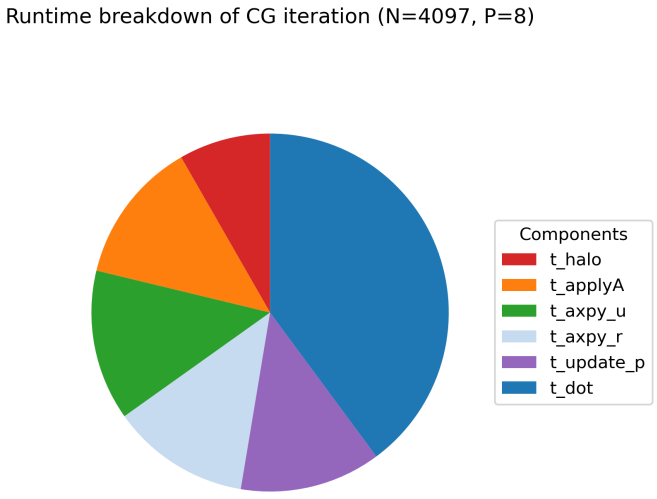


Figure 4: Fraction of CG time spent in each kernel for $P = 8$.

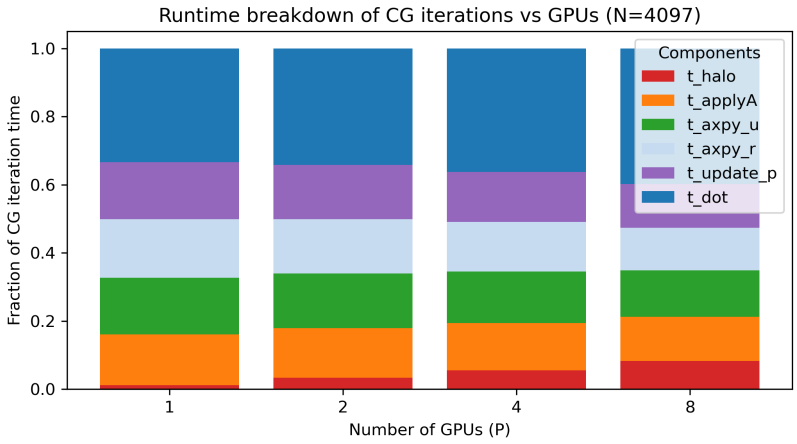


Figure 5: Runtime breakdown versus number of GPUs.

4.5 End-to-End Speedup Across Phases

Figure 6 summarizes cumulative speedup across implementation phases.

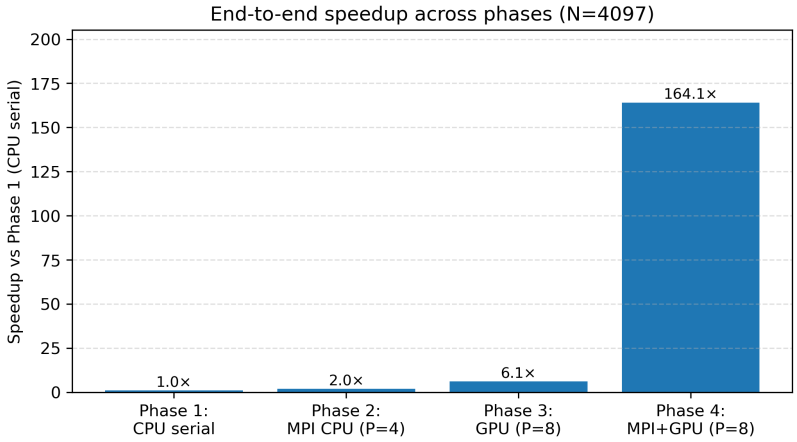


Figure 6: End-to-end speedup from Phase 1 to Phase 4.

The final MPI+GPU implementation achieves over two orders of magnitude improvement relative to the serial CPU baseline, highlighting the combined impact of GPU acceleration and distributed parallelism.

Overall, the results show that while GPU acceleration substantially reduces the cost of local stencil operations, global synchronization increasingly dominates runtime at scale. This interaction between memory-bound kernels and collective communication ultimately limits strong scaling efficiency on multi-GPU systems.

5 Conclusion

This project developed a high-performance Conjugate Gradient solver for the Poisson equation across CPU, MPI, GPU, and distributed MPI+GPU architectures. GPU acceleration substantially reduced the cost of local stencil operations, while distributed execution enabled strong scaling to multiple devices. Performance analysis demonstrated that although the five-point Laplacian dominates the computational workload, scalability is ultimately limited by the global dot-product reductions required by CG.

Because these global synchronizations occur every iteration, this limitation is fundamental to the algorithm and motivates future work on communication-avoiding Krylov methods or multigrid preconditioning for large-scale systems.