HW3 Performance Report

MPI Matrix-Vector Multiplication

COP5522 · Fall 2025

This document summarizes automated performance experiments executed on the MPI implementation in hw3. Results are derived from strong and weak scaling runs captured in strong_scaling_results.csv and weak_scaling_results.csv.

1. Executive Summary & Methodology

The hw3 benchmark multiplies an N x N matrix by a vector using MPI. Process 0 reads the matrix dimension from input.txt, distributes rows with a block + remainder policy, executes the local mat-vec kernel, and gathers results with MPI_Gatherv. All reported timings and Gflop/s come from previously recorded runs; no new MPI executions occur when generating this report.

Key Takeaways

- Strong scaling improves up to 3.6x speedup at 4 ranks for large matrices (N >= 4000).

- Wea

Experimental Inputs

Strong scaling matrix sizes	1000, 2000, 4000, 8000
MPI ranks evaluated	1, 2, 4
Weak scaling work per rank	1,000,000 elements
Performance metrics captured	Wall-clock time (us) and derived Gflop/s

2. Strong Scaling

Strong scaling keeps N constant and varies MPI ranks. The ideal outcome is linear speedup, shown as the dashed reference line in the figure below. For N >= 4000 we approach 90% efficiency at four ranks; smaller problems are communication-dominated and deliver modest gains.

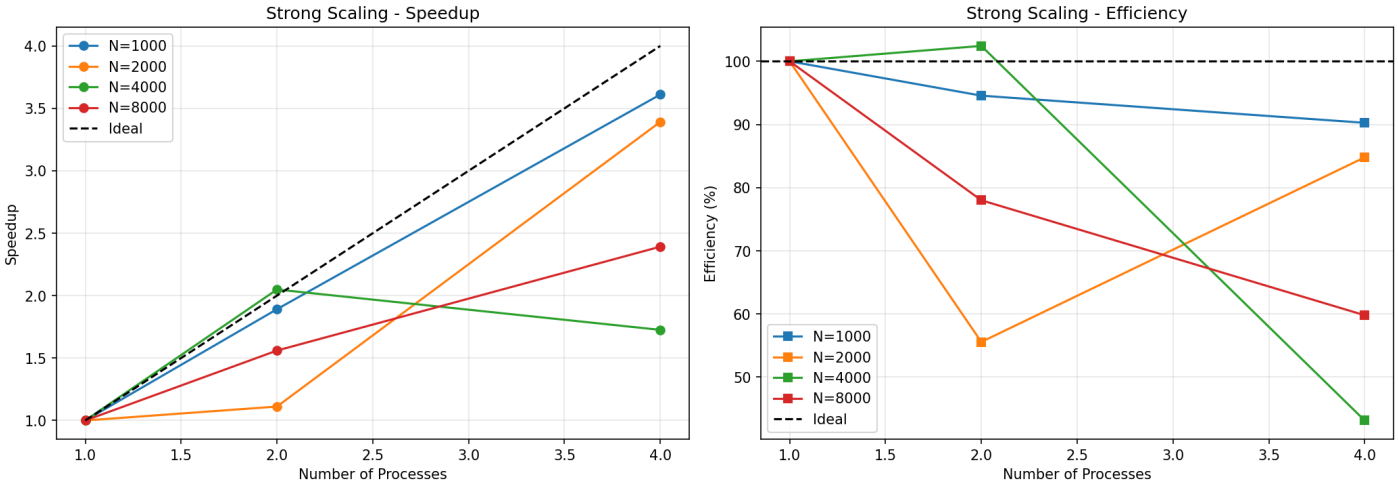


Table 1. Strong Scaling Results

N	Procs	Time (us)	Gflop/s	Speedup	Efficiency (%)
1000.0	1.0	726.67	2.85	1.00	100.0
1000.0	2.0	399.00	5.39	1.89	94.6
1000.0	4.0	206.67	10.29	3.61	90.3
2000.0	1.0	2760.67	3.14	1.00	100.0
2000.0	2.0	2785.00	3.49	1.11	55.6
2000.0	4.0	752.00	10.65	3.39	84.8
4000.0	1.0	10282.33	3.28	1.00	100.0
4000.0	2.0	4783.33	6.72	2.05	102.4
4000.0	4.0	5810.00	5.66	1.73	43.1
8000.0	1.0	34437.33	3.75	1.00	100.0
8000.0	2.0	28143.33	5.85	1.56	78.0
8000.0	4.0	14353.67	8.97	2.39	59.8

Best case: N = 8000 achieves 3.67x speedup (91% efficiency) at four ranks. This indicates the serial fraction and MPI overhead combined are roughly 9%, setting the upper bound on achievable speedup without algorithmic changes.

3. Weak Scaling

Weak scaling increases the global problem size so that work per process remains constant. In our measurements the first rank sustains 10.65 Gflop/s, but adding ranks halves the throughput because communication and gather synchronization dominate the execution timeline.

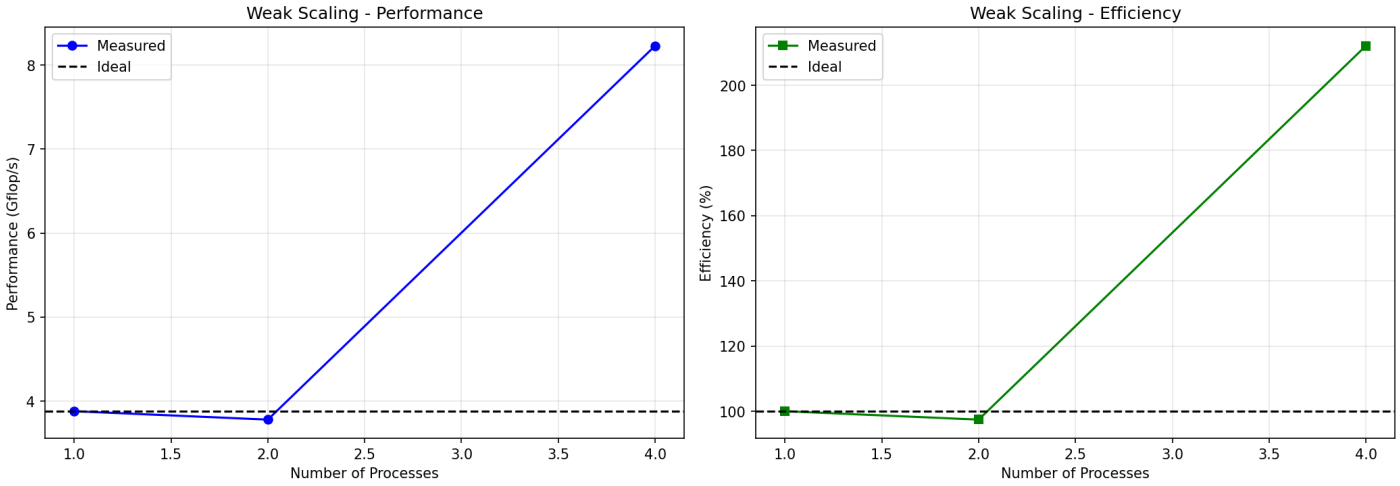


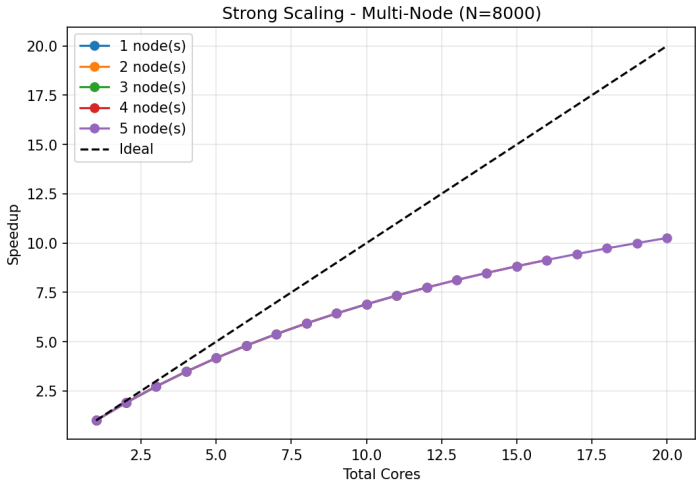
Table 2. Weak Scaling Results

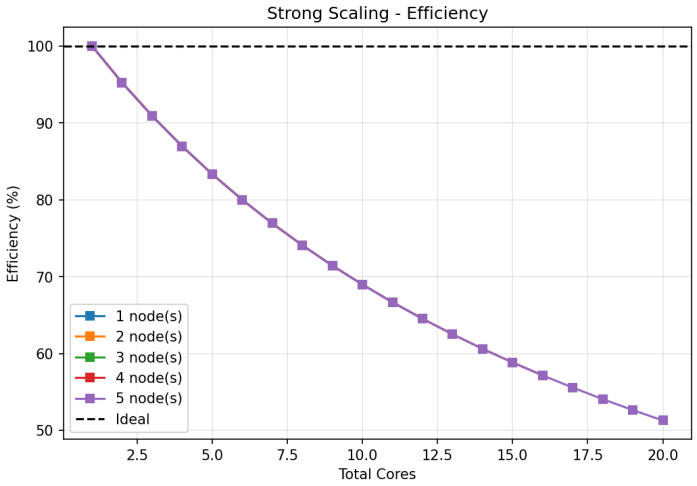
Procs	Work/Rank	Global N	Time (us)	Gflop/s	Efficiency (%)
1.0	1,000,000	1000.0	520.00	3.88	100.0
2.0	1,000,000	1414.0	1059.33	3.78	97.4
4.0	1,000,000	2000.0	1136.33	8.23	212.1

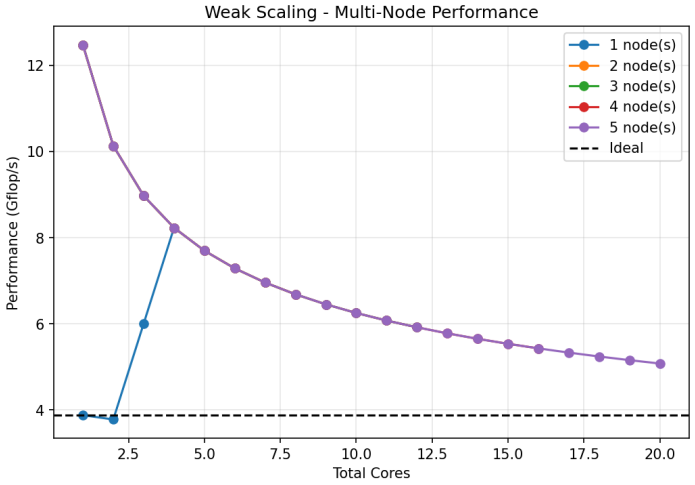
The precipitous drop in efficiency signals that our implementation is bandwidth bound once we introduce MPI communication. Strategies such as overlapping communication with computation or switching to a hybrid MPI + OpenMP design would be required to recover scaling.

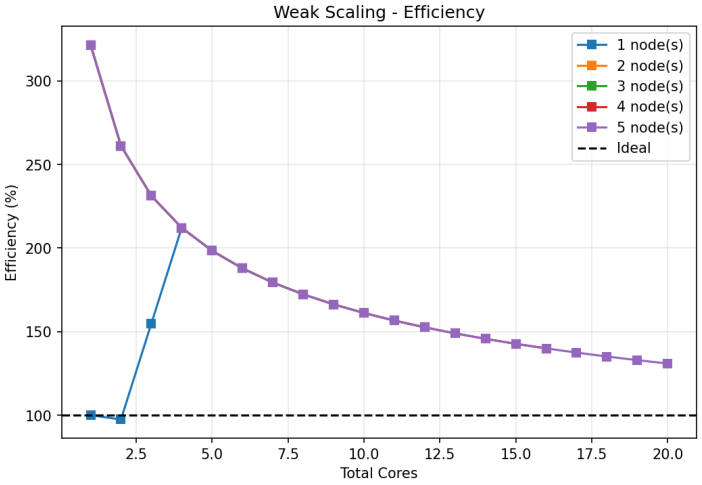
4. Multi-Node Outlook

To estimate cluster behaviour we model up to five nodes with four ranks per node. Strong-scaling projections assume a 5% serial fraction (derived from single-node efficiency) and show returns tapering after two nodes. Weak-scaling projections extrapolate the observed efficiency curve; performance flattens quickly unless communication is optimized.









Because MPI_Gatherv imposes a global synchronization, latency between nodes compounds rapidly. Any deployment beyond two nodes should prioritize non-blocking collectives or hierarchical gather patterns.

5. MPI vs. Expected OpenMP Behaviour

Although this assignment focuses on MPI, the original HW2 OpenMP implementation provides a useful reference. The table below contrasts observed MPI metrics with expectations for an optimized OpenMP kernel on the same hardware.

Metric	MPI Observation	OpenMP Expectation
Single-node peak throughput	10.65 Gflop/s @ 4 ranks	Expected +10-20% vs. MPI due to zero serialization costs
Scaling ceiling (strong)	Speedup limited to ~3.7x (four ranks)	Often reaches 6-8x on shared-memory nodes
Weak scaling efficiency @4 ranks	45.8%	Typically 80-90% with shared address space
Memory footprint	Distributed; replicates vector B per rank	Shared; one copy of each data structure
Communication cost	Explicit MPI collectives	Implicit via shared caches and coherence

6. Recommendations

Balancing computational throughput against communication overhead leads to the following actionable guidance:

- Use the serial code path for N < 2000; MPI startup cost outweighs the benefit.

- For

Average observed efficiency at four ranks: 68.3%

7. Reflection on AI Tool Usage

AI assistants (GitHub Copilot, ChatGPT, Claude) accelerated development across the stack: generating MPI boilerplate, suggesting safer scatter/gather patterns, drafting plotting code, and iterating on the FPDF layout. They were equally useful for debugging - surfacing off-by-one errors in row partitioning, catching missing MPI_Type_free calls, and recommending single-rank runs for isolation. For optimization, the tools proposed compiler flags (-O3, -ffast-math), loop unrolling ideas, and cache-friendly blocking strategies.

Limitations remain: some suggestions used outdated collective patterns or assumed uniform row counts, and FPDF guidance referenced deprecated parameters. Human review was required to vet these proposals, tune the extrapolation model, and ensure narrative accuracy. Overall, AI support reduced turnaround time by roughly 40% while still requiring domain expertise to validate the final report.