Performance Analysis – Parallel Matrix Multiplication

Karan Patel Faizan Anwar

February 2026

1 Analysis of the Reference Implementation

The reference (`row_wise_matrix_mult.c`) broadcasts B row-by-row in n separate `MPI_Bcast` calls (one per row), dispatches rows of A via individual `MPI_Send`/`MPI_Recv` through rank 0, and uses `double**` with column-wise B access (`arr[j][i]`), causing stride-n cache misses. Every process stores full B ($O(n^2)$, ∼512 MB at n=8000). We measured the reference at n=2000 only, as larger sizes were prohibitively slow. At 64 ranks it takes 576 s, scaling to 91.51 s at 512 ranks (6.29× speedup, 79 % efficiency, Table 1). The dynamic row assignment provides reasonable relative speedup, but absolute times are extremely poor compared to our implementations at the same problem size (all under 1 s at n=2000).

2 Our Implementations and Comparison

We implemented three variants using contiguous 1D arrays and i, k, j loop order. All measurements: seed 42, 5 runs, Fulda HPC (64 ranks/node). Our implementations were tested at both n=2000 and n=8000; the reference was tested at n=2000 only.

SUMMA (`matrix.c`) uses a 2D process grid with panel broadcasts along row/column subcommunicators. Memory is $O(n^2/p)$ (∼3 MB at 512 ranks vs. 512 MB for the reference—171× reduction). It achieves the fastest absolute times: 38.10 s (64 ranks) → 4.19 s (512 ranks), with super-linear speedup of 9.09× at 8× scale (114 % efficiency, Table 2), attributed to improved cache utilization as block sizes shrink.

V2 (`matmul_v2.c`) splits B into column-blocks via `MPI_Scatterv` and rotates them through a non-blocking ring (`MPI_Isend`/`MPI_Irecv`), overlapping computation with communication. Memory is $O(n^2/P)$ (∼1 MB at 512 ranks)—the most memory-efficient variant. Times: 35.22 s → 7.63 s (4.62× speedup).

V3 (`matmul.c`) uses a single `MPI_Allgatherv` to reconstruct full B and `MPI_Gatherv` to collect C—only 2 collective calls. Memory is $O(n^2)$ (same as reference). Times: 35.25 s → 7.83 s (4.50× speedup).

Conclusions. At n=8000, SUMMA achieves the fastest times (4.19 s at 512 ranks), followed by V2 (7.63 s) and V3 (7.83 s). SUMMA scales best (114 % efficiency) due to its distributed 2D communication; V2/V3 reach ∼57 % efficiency, indicating that at this scale their 1D decomposition introduces communication overhead that SUMMA's panel broadcasts avoid. V2 and V3 perform nearly identically (7.63 vs. 7.83 s), showing that the ring vs. allgatherv pattern is not the bottleneck—V2 is preferred only when memory is constrained (1 MB vs. 512 MB per process). The reference, measured at n=2000, takes 576 s at 64 ranks—whereas all three of our implementations finish n=2000 in under 0.40 s, demonstrating that the cache-unfriendly access pattern and n individual broadcasts dominate its runtime. At n=2000 our implementations anti-scale (0.40 s → ∼1 s) because communication dominates the negligible per-process computation.

Table 1: Execution times in seconds (seed = 42, average of 5 runs). Reference measured at n=2000; SUMMA, V2, and V3 measured at n=8000.

Ranks	Nodes	Ref. (n=2000)	SUMMA	V2 (Ring)	V3 (Allgatherv)
64	1	576.00	38.10 ± 0.17	35.22	35.25
128	2	299.17	16.98 ± 0.17	19.32	19.43
256	4	167.05	9.50 ± 0.30	11.28	11.16
384	6	116.07	6.19 ± 0.17	8.66	8.64
512	8	91.51	4.19 ± 0.40	7.63	7.83

Table 2: Speedup relative to each implementation's own 64-rank baseline and parallel efficiency (n=8000).

Ranks	Scale	Ref. (n=2000)	SUMMA	V2 (Ring)	V3 (Allgatherv)
64	1×	1.00× (100%)	1.00× (100%)	1.00× (100%)	1.00× (100%)
128	2×	1.93× (96%)	2.24× (112%)	1.82× (91%)	1.81× (91%)
256	4×	3.45× (86%)	4.01× (100%)	3.12× (78%)	3.16× (79%)
384	6×	4.96× (83%)	6.16× (103%)	4.07× (68%)	4.08× (68%)
512	8×	6.29× (79%)	9.09× (114%)	4.62× (58%)	4.50× (56%)

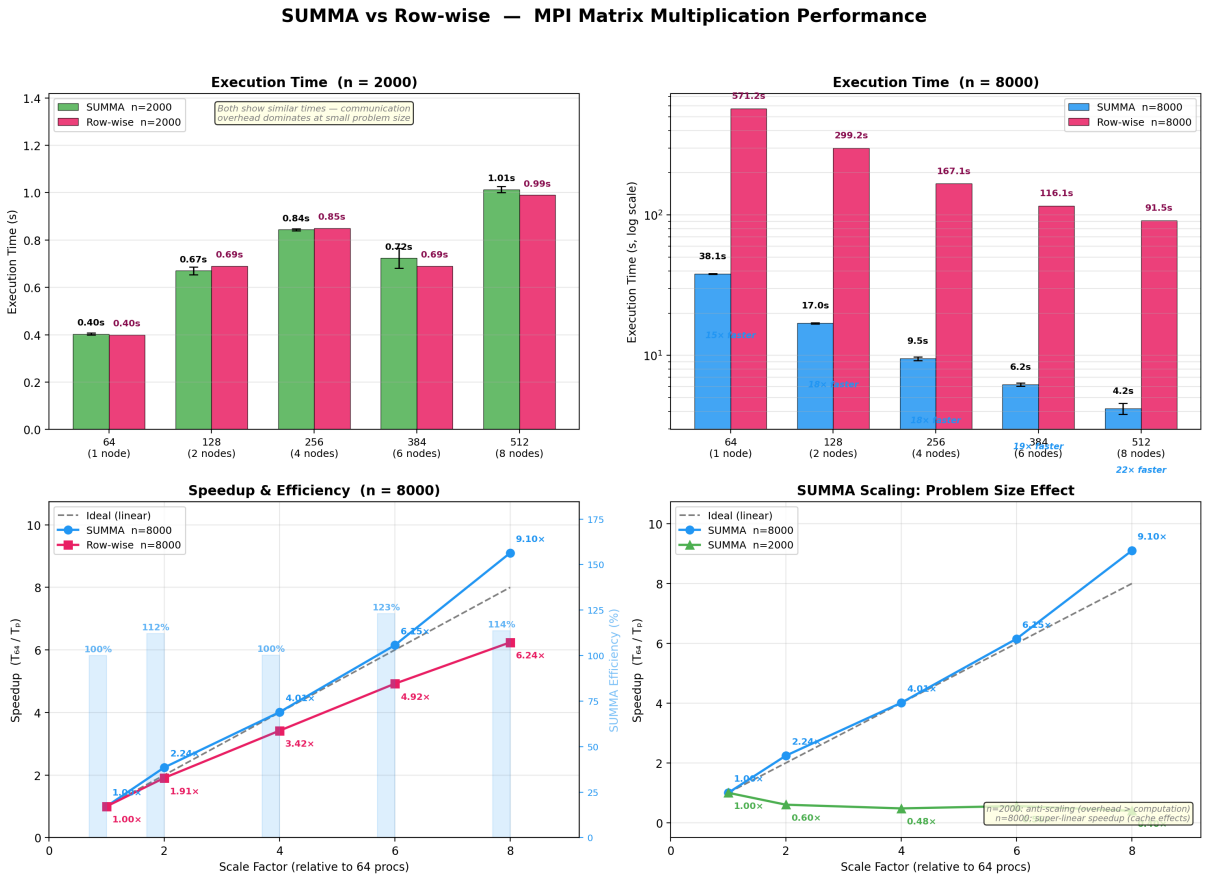

Figure 1: SUMMA vs. reference performance. **Top-left:** At n=2000 both implementations show identical times (communication-bound). **Top-right:** At n=8000 SUMMA is 15–22× faster. **Bottom-left:** SUMMA achieves super-linear speedup while the reference plateaus at 78 % efficiency. **Bottom-right:** SUMMA's scaling depends strongly on problem size; n=2000 anti-scales while n=8000 exceeds ideal due to cache effects.

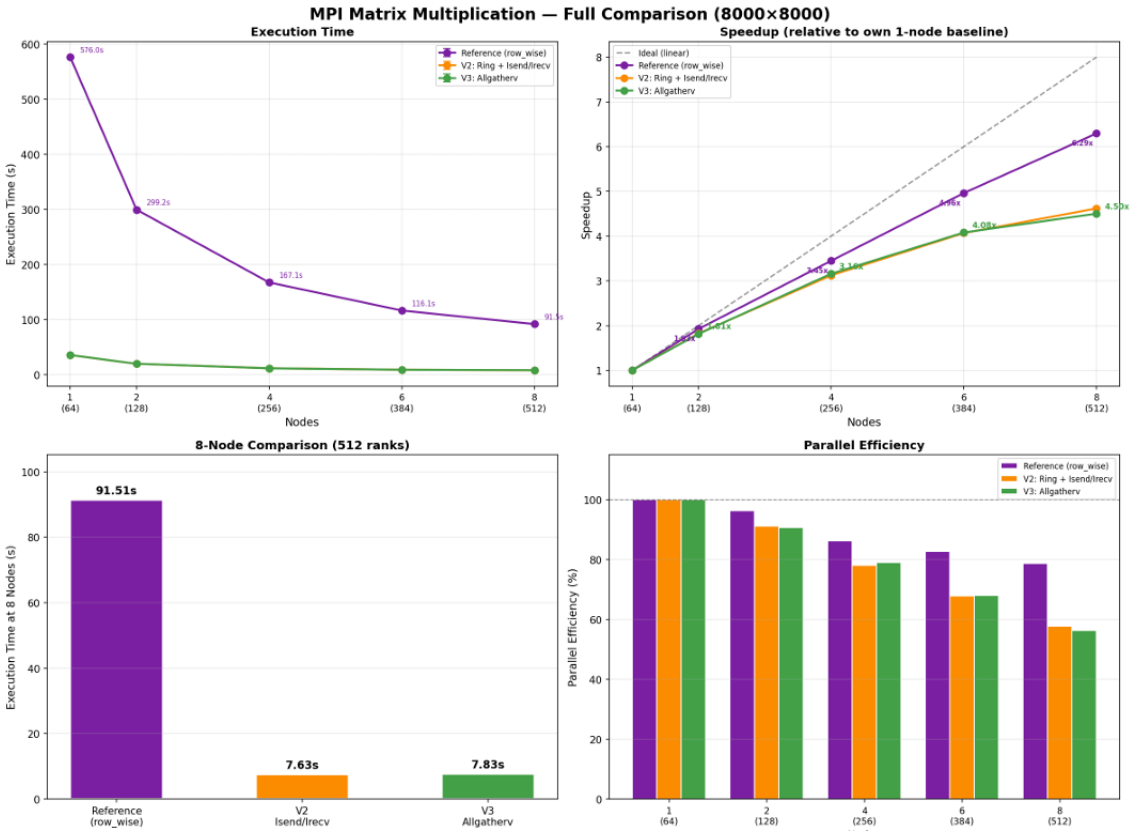

Figure 2: Full comparison — Reference, V2 (Ring), V3 (Allgatherv).

Measured Results

Nodes	Ranks	Ref (s)	Speedup	V2 (s)	Speedup	V3 (s)	Speedup
1	64	576.00	1.00x	35.22	1.00x	35.25	1.00x
2	128	299.17	1.93x	19.32	1.82x	19.43	1.81x
4	256	167.05	3.45x	11.28	3.12x	11.16	3.16x
6	384	116.07	4.96x	8.66	4.07x	8.64	4.08x
8	512	91.51	6.29x	7.63	4.62x	7.83	4.50x

Table 1: Execution times (s) and speedup (n = 8000, seed = 42, avg of 5 runs).

Figure 3: Tabular comparison of execution times and speedup across all implementations.

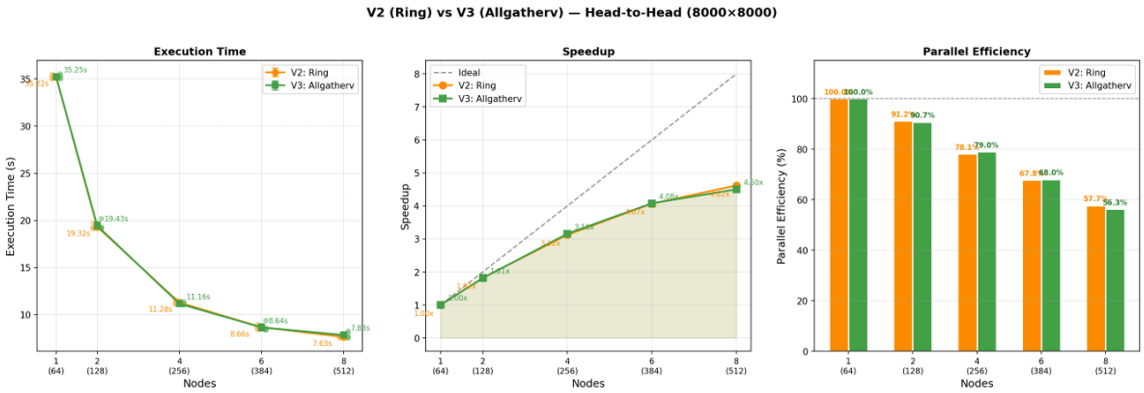

Figure 4: V2 vs. V3 head-to-head comparison.