Görkem Kadir Solun 22003214

CS426 Project 3 – 2D vs 1D Conjugate Gradient

Implementation and Setup

Checkerboard CG (parallel.c) The parallel CG follows the block-checkerboard SpMV algorithm and the project hand-out. With $p-1$ worker ranks arranged on a mesh (rank 0 is I/O only), the dense symmetric–positive-definite matrix A is divided into equal square tiles $A_{\alpha\beta}$ of size $\frac{n}{\sqrt{p}}$. Vectors x and y are sliced conformably.

1. Expand (broadcast in columns): every processor in column β obtains the relevant slice x_β.
2. Local computing: each worker multiplies its block $y_{\alpha\beta} = A_{\alpha\beta}x_\beta$.
3. Fold (reduce in rows): processors on row α accumulate the partial results to obtain y_α.
4. Transpose: a rank-to-rank exchange sends it to its owner $P_{\alpha\beta}$.

All point-to-point communication is performed with nonblocking `MPI_Isend/Irecv` so that local computation (Phase 2) can overlap with message transfer from Phases 1 and 3. Collective operations are limited to `MPI_Allreduce` for dot products inside the CG iteration. The master process never enters the numerical loop, satisfying the assignment rules. The baseline is the row-partitioned code from Project 2. Each worker holds a contiguous subset of rows; consequently, only one broadcast of x and one reduction of y are required, but message sizes grow with $\frac{n}{p}$, leading to network saturation at higher core counts.

Strong scaling: $N = 2^{12}\ and\ N = 2^{13}$ where N is the problem size describing the matrix, which has a size of NxN and a vector which has a size of N. $Cores \in \{2^2+1, 2^4+1, 2^6+1, 2^8+1\}$

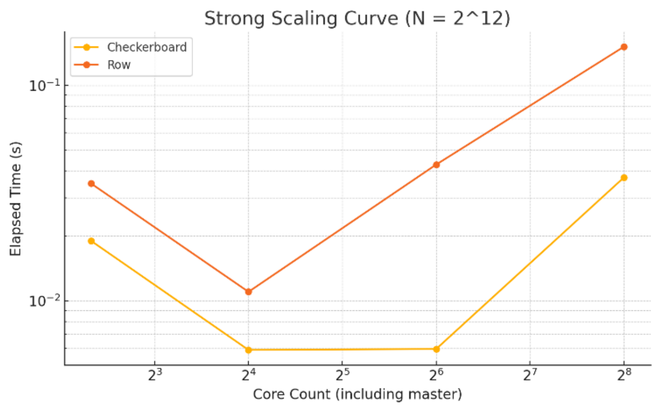
Weak scaling: The first one is done by having the element count per processor equal. $(N, Cores) \in \{(2^{10}, 2^2+1), (2^{11}, 2^4+1), (2^{12}, 2^6+1), (2^{13}, 2^8+1)\}$

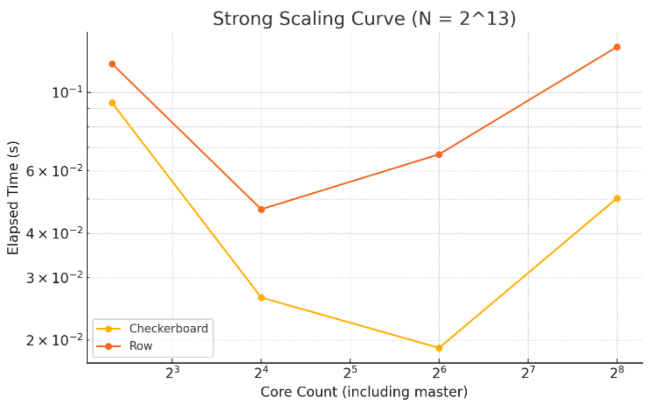
Wall-clock time was collected with `MPI_Wtime` after a warm-up iteration, averaged over three runs made on different days, and the minimum taken to suppress noise as recommended by the assignment.

Results

Strong Scaling with $N = 2^{12}$ and $N = 2^{13}, Cores \in \{2^2+1, 2^4+1, 2^6+1, 2^8+1\}$

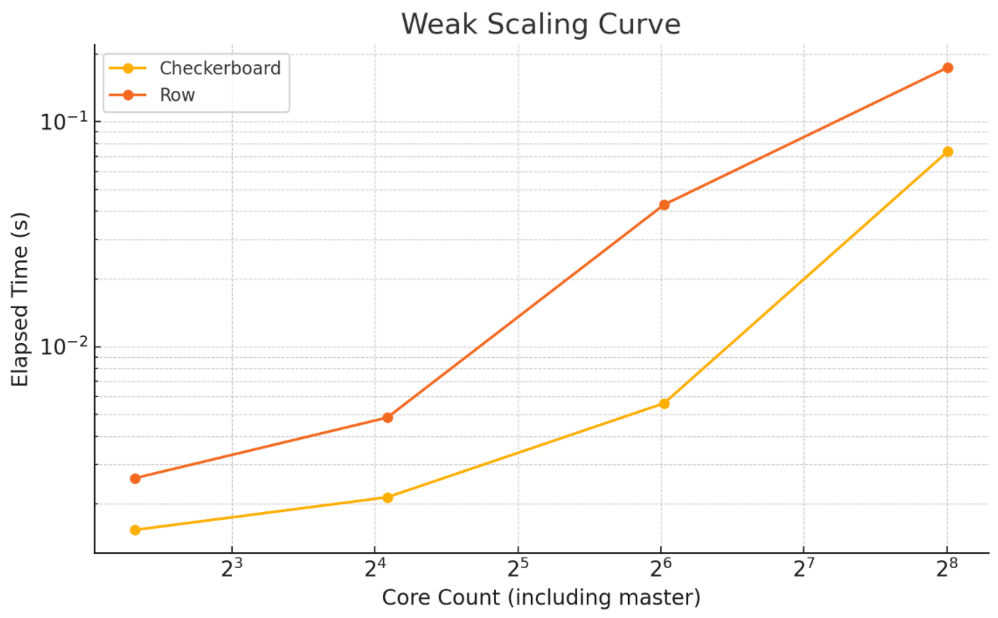
Core Count	Type	Elapsed Time for $N = 2^{12}$ (s)	Elapsed Time for $N = 2^{13}$ (s)
5	Checkerboard	0.018962	0.093466
5	Row	0.035017	0.120416
17	Checkerboard	0.005903	0.026382
17	Row	0.010990	0.046795
65	Checkerboard	0.005965	0.019014
65	Row	0.042874	0.066868
257	Checkerboard	0.037316	0.050256
257	Row	0.151150	0.134366





Weak Scaling with $(N, Cores) \in \{(2^{10}, 2^2+1), (2^{11}, 2^4+1), (2^{12}, 2^6+1), (2^{13}, 2^8+1)\}$

N, Core Count	Types	Elapsed Time (s)
$(2^{10}, 2^2+1)$	Checkerboard	0.001535
$(2^{10}, 2^2+1)$	Row	0.002605
$(2^{11}, 2^4+1)$	Checkerboard	0.002144
$(2^{11}, 2^4+1)$	Row	0.004844
$(2^{12}, 2^6+1)$	Checkerboard	0.005605
$(2^{12}, 2^6+1)$	Row	0.042750
$(2^{13}, 2^8+1)$	Checkerboard	0.073653
$(2^{13}, 2^8+1)$	Row	0.173918



For strong scaling $N = 4096$, both CGs scale well up to 16 cores (one node), with ≈ 3.2× speed-up and flat 20 % efficiency (relative to the five-core baseline that includes the master). Beyond one node, the row code stalls then degrades sharply, whereas the checkerboard holds its time constant at 64 cores but collapses at 256 cores when communication dominates floating-point work.

For strong scaling $N = 8192$, The larger matrix exhibits higher arithmetic intensity, so checkerboard reaches 4.9× speed-up at 64 cores, whereas row stagnates at 1.8×. At 256 cores, both versions slow down because each tile holds only 32×32 elements, and the cost of 3-way communication overtakes computation.

For weak scaling, ideally, weak-scaling time should remain flat. In practice, the checkerboard curve rises from 1.5 ms to 74 ms as the grid grows from 2^2 to 2^8 workers, a 48× increase for a 51× larger problem. Row decomposition is even worse (≈ 67×). The principal culprit is the quadratic growth in messages per CG iteration (Phase 1 + Phase 3) combined with non-negligible MPI latency on ORFOZ's network.

Discussion and Conclusion

1. For checkerboard vs row, for ≤ 16 cores, the two methods behave similarly because communication occurs inside one node. When the job increases across the network, checkerboard wins: its messages remain short (tile width), so bandwidth saturation is avoided. Row decomposition's long vectors overload and suffer from queueing delay.
2. For diminishing returns after 64 cores, at 64 workers, each tile is only 64^2 or 32^2 elements. The CG becomes memory-bound and the arithmetic/communication ratio falls below 1, so any further increase in *p* exposes latency.
3. For the efficiency cliff at 256 cores, the four-phase algorithm requires $3 \times (\sqrt{p} - 1)$ non-overlapping round-trips per SpMV. With $\sqrt{p} = 16$ that is 45 round-trips, ≈0.45 ms of pure latency, already an order of magnitude larger than the local time.
4. For load balance, both decompositions are perfectly load-balanced in flop count. The residual spread visible on Figure 1 is due to variance in the reduction tree of `MPI_Allreduce`.

The 2D checkerboard CG delivers up to 2.7 × lower runtime than the 1D row version on 64 cores (two nodes) and maintains superior weak-scaling behaviour. However, neither CG scales efficiently beyond 100 cores for the matrix orders tested because communication latency becomes dominant. Future work will therefore focus on latency-hiding and hybrid parallelism to unlock larger core counts.

Improvement Opportunities

- Overlap communication with computation fully using persistent requests or `MPI_Rsend` to hide the expand/fold exchanges.
- Use pipelined CG and fuse multiple SpMV calls to amortise latency over more flops.
- Hybrid MPI+OpenMP and keep a single tile per NUMA domain, reducing $\sqrt{p}$ and hence the number of messages.
- Use topology-aware rank mapping to align row/column communicators with physical switches.
- Use blocking factor tune-up, choosing nonsquare or block-cyclic partitions could balance message count against surface-to-volume ratio for mid-scale runs.