Görkem Kadir Solun 22003214

CS426 Project 2 - Parallel Conjugate Gradient

Introduction

This report presents performance results for a dense, MPI-parallel Conjugate Gradient (CG) solver implemented as part of CS426/525 Project 2. We measure elapsed times on the ORFOZ cluster for varying core counts and problem sizes to evaluate. All timings use `MPI_Wtime()` with appropriate barriers.

- Strong scaling: fixed problem size, increasing cores.
- Weak scaling: problem size grows proportionally with cores.

Timings were extracted from the output files produced by each run.

Experimental Setup

Hardware: ORFOZ cluster nodes.

Software: MPI-based CG solver (dense SPD matrix).

Strong scaling:

$N = 2^{10}\ and\ N = 2^{11}$ where N is the problem size describing the matrix which has a size of NxN and vector which has a size of N.

$Cores \in \{1, 2^4 + 1, 2^5 + 1, 2^6 + 1, 2^7 + 1, 2^8 + 1\}$

Weak scaling:

$(N, Cores) \in \{(2^{11}, 2^8 + 1), (2^{10}, 2^7 + 1), (2^9, 2^6 + 1), (2^8, 2^5 + 1), (2^7, 2^4 + 1)\}$

Results

Diminishing returns beyond 16 cores meaning communication latency and synchronization dominate, causing execution time to rise for cores ≥ 32.

Table 1: Strong Scaling Curve with $N = 2^{10}$, $Cores \in \{1, 2^4 + 1, 2^5 + 1, 2^6 + 1, 2^7 + 1, 2^8 + 1\}$

$N = 2^{10}$, $Cores \in \{1, 2^4 + 1, 2^5 + 1, 2^6 + 1, 2^7 + 1, 2^8 + 1\}$	
Core Count	Elapsed Time (s)
1	0.004023
16	0.001184
32	0.002960
64	0.012656
128	0.056406
256	0.037674

Figure 1: Strong Scaling Curve with $N = 2^{10}$, $Cores \in \{1, 2^4 + 1, 2^5 + 1, 2^6 + 1, 2^7 + 1, 2^8 + 1\}$

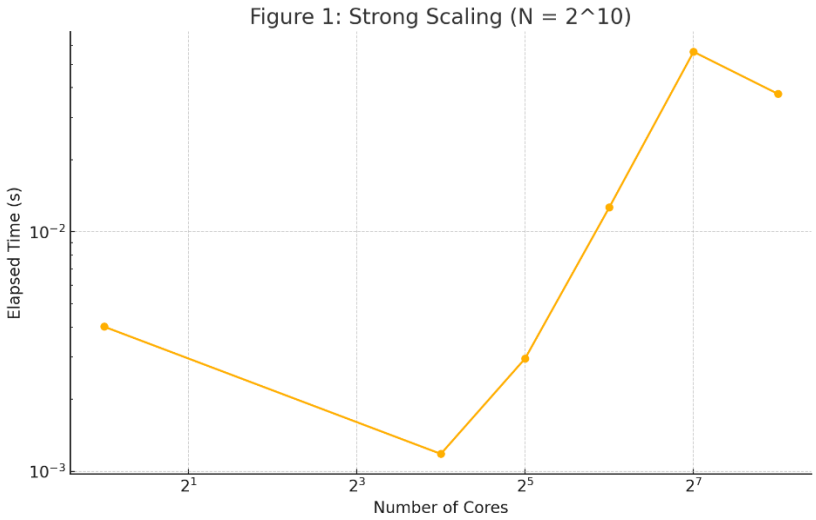


Table 2: Strong Scaling Curve with $N = 2^{11}, Cores \in \{1, 2^4 + 1, 2^5 + 1, 2^6 + 1, 2^7 + 1, 2^8 + 1\}$

$N = 2^{11}, Cores \in \{1, 2^4 + 1, 2^5 + 1, 2^6 + 1, 2^7 + 1, 2^8 + 1\}$	
Core Count	Elapsed Time (s)
1	0.017609
16	0.002168
32	0.004520
64	0.013138
128	0.039841
256	0.036201

Figure 2: Strong Scaling Curve with $N = 2^{11}, Cores \in \{1, 2^4 + 1, 2^5 + 1, 2^6 + 1, 2^7 + 1, 2^8 + 1\}$

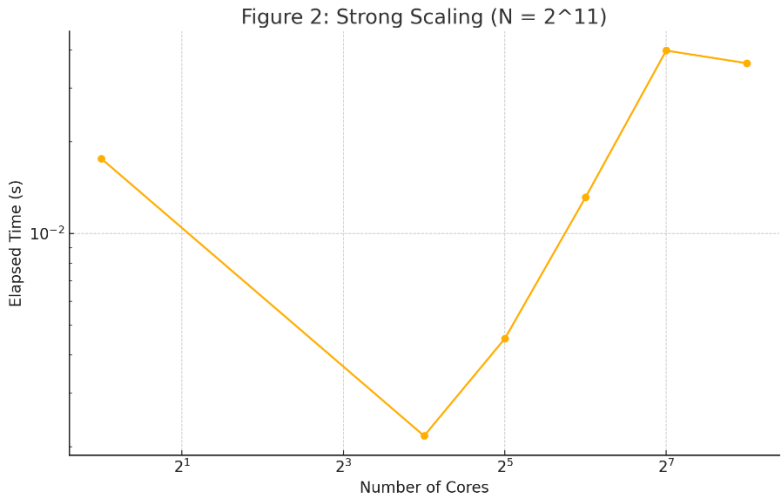
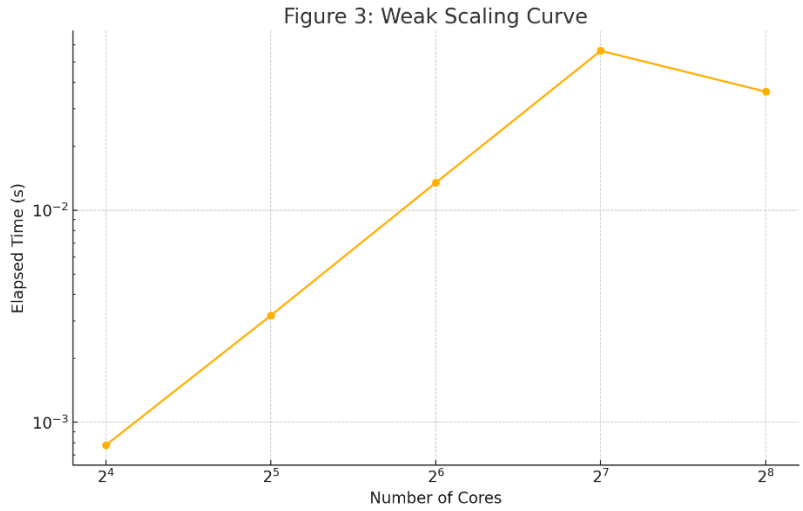


Table 3: Weak Scaling Curve with $(N, Cores) \in \{(2^{11}, 2^8 + 1), (2^{10}, 2^7 + 1), (2^9, 2^6 + 1), (2^8, 2^5 + 1), (2^7, 2^4 + 1)\}$

$(N, Cores) \in \{(2^{11}, 2^8 + 1), (2^{10}, 2^7 + 1), (2^9, 2^6 + 1), (2^8, 2^5 + 1), (2^7, 2^4 + 1)\}$	
N, Core Count	Elapsed Time (s)
$(2^7, 2^4 + 1)$	0.000779
$(2^8, 2^5 + 1)$	0.003183
$(2^9, 2^6 + 1)$	0.013461
$(2^{10}, 2^7 + 1)$	0.056406
$(2^{11}, 2^8 + 1)$	0.036201

Figure 3: Weak Scaling Curve with $(N, Cores) \in \{(2^{11}, 2^8 + 1), (2^{10}, 2^7 + 1), (2^9, 2^6 + 1), (2^8, 2^5 + 1), (2^7, 2^4 + 1)\}$



Ideal weak scaling keeps time nearly constant; here it increases from 0.78 ms (16 cores) to 56.4 ms (128 cores), then slightly improves at 256 cores (36.2 ms). The steep rise indicates that as the problem/processor ratio stays fixed, communication (point-to-point exchanges and global reductions) overhead grows super linearly.

Discussion

Strong Scaling

For both $N = 2^{10}$ and 2^{11}, best speedup occurs at 16-32 cores. Beyond 32 cores, elapsed time increases markedly, communication/latency dominates over per-core computation. The overhead of exchanging vector segments and reductions outweighs local SpMV gains.

Weak Scaling

Ideal weak scaling maintains constant runtime as both problem size and cores increase. Here, elapsed time grows over an order of magnitude from 16 to 256 cores, indicating significant overhead in distributed matrix-vector multiple times and global reductions.

Overheads

P2P communication during SpMV and `MPI_Allreduce` for dot-products contributes to the non-ideal scaling. For small to moderate core counts, computation dominates; at high counts, communication latency and load imbalance are limiting factors.

Conclusion

The MPI-parallel CG solver shows modest strong scaling up to 32 cores. However, communication overhead severely limits both strong and weak scaling at larger scales. Future optimizations might include overlapping communication and computation, using nonblocking collectives, exploring hybrid `MPI+OpenMP` to reduce inter-node messages, and using topology-aware process mapping to minimize network hops.