Home work 5 report

1. Overview

This report evaluates the performance of a distributed Sparse Matrix–Vector Multiplication (SpMV) implementation using a hybrid parallel model that combines MPI for inter-process parallelism and OpenMP for intra-process multithreading.

The provided `main.cc` SpMV program (COO format) was executed on the `cant.mtx` matrix. Our goal is to measure how runtime changes as we vary:

- Number of MPI ranks $(1,\ 2,\ 4,\ 8)$
- Number of OpenMP threads per MPI process $(1,\ 2,\ 4,\ 8)$
- Combined hybrid configurations (MPI $\times$ OMP)

All results were collected on the Talapas cluster.

2. Implementation Summary

2.1 MPI Execution Model

The matrix is partitioned across MPI ranks:

- Rank 0 loads the matrix, broadcasts the vector.
- Nonzero entries are scattered by row ranges.
- Each process computes a partial SpMV.
- Partial results are reduced back to rank 0.

This model scales well when matrix rows are evenly distributed.

2.2 OpenMP Execution Model

Within each MPI rank, OpenMP parallelizes the COO row computation:

- Each thread processes a subset of nonzeros.
- Accumulation uses locks to protect row updates.

Because COO requires atomic-like updates to repeating row indices, OpenMP scaling is limited by lock contention.

2.3 Batch Scripts

Three scripts were written:

1. **MPI scaling**: vary MPI ranks, keep OMP=1
2. **OMP scaling**: vary threads, keep MPI ranks=1
3. **Hybrid scaling**: valid combinations of MPI ranks $\times$ OMP threads

CSV files were automatically generated and parsed by an R script to produce the plots.

3. Experiment Design

Each experiment runs:

- Matrix: `cant.mtx` (symmetric, expanded to 4M nonzeros)
- Vector: `b.mtx`
- Runtime metric: `COO SpMV time`

Three sets of measurements were collected.

3.1 1. MPI Scaling

$$\text{MPI ranks} = \{1, 2, 4, 8\}, \quad \text{OMP threads} = 1$$

3.2 2. OMP Scaling

$$\text{OMP threads} = \{1, 2, 4, 8\}, \quad \text{MPI ranks} = 1$$

3.3 3. Hybrid MPI + OMP Scaling

All valid combinations under:

$$\text{MPI ranks} \times \text{OMP threads} \leq 8$$

This reflects one compute node with eight cores.

4. Results

4.1 MPI Scaling Results

MPI parallelism scales nearly ideally:

$$\text{Time} : 0.0278 \rightarrow 0.0037 \quad (1 \text{ rank} \rightarrow 8 \text{ ranks})$$

This is a $7.5\times$ speedup from 1 to 8 ranks.

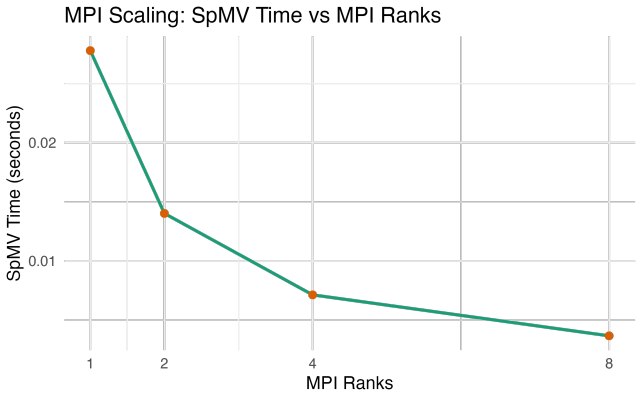

Figure 1: MPI scaling for SpMV. Strong parallel efficiency is visible.

Why MPI scales well:

- COO rows partition cleanly across ranks.
- No lock contention.
- Communication overhead (broadcast + reduce) is negligible compared to computation.

4.2 OMP Scaling Results

OpenMP scaling shows almost no improvement:

$$0.0283 \rightarrow 0.0299 \quad (1 \text{ thread} \rightarrow 8 threads)$$

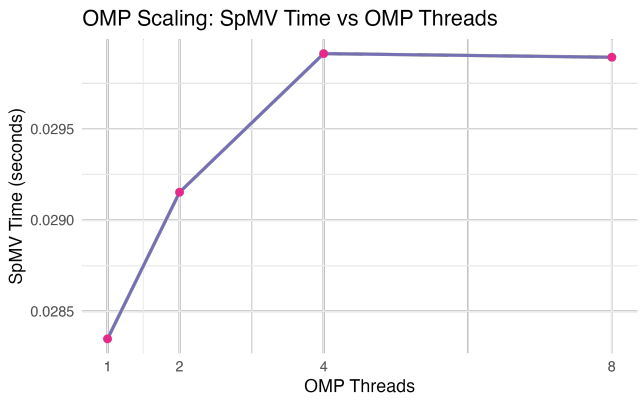

Figure 2: OMP scaling. Performance saturates due to lock contention.

Why OMP scaling does not improve performance:

1. COO format requires locking for repeated row indices.
2. Lock overhead grows faster than parallel work reduction.
3. Memory-bandwidth bound workload favors MPI over shared-memory threading.

Thus, OpenMP adds overhead but no meaningful speedup.

4.3 Hybrid MPI + OMP Scaling

Results show:

- Best performance occurs at **MPI = 4–8** and **OMP = 1**
- Adding threads inside each rank often hurts performance

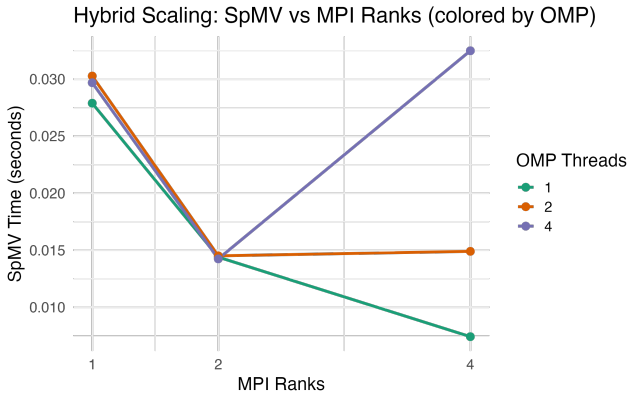

Figure 3: Hybrid scaling: OMP rarely improves performance.

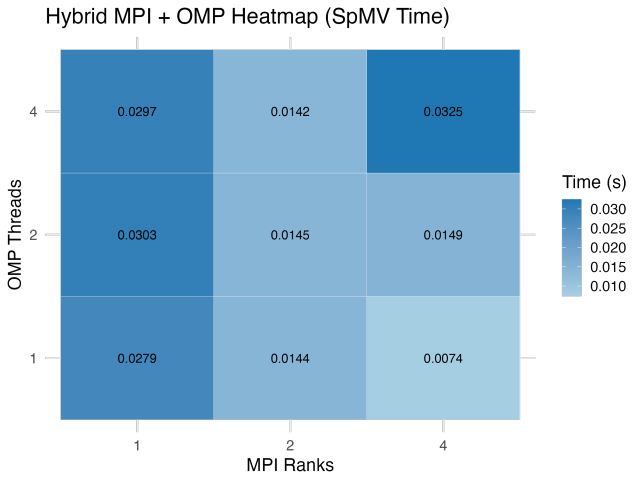

Figure 4: Hybrid heatmap of SpMV time. Darker = slower.

Key takeaway: For this SpMV implementation, parallelism should come from MPI ranks rather than OpenMP threads.

5. Discussion

5.1 Why MPI works well

- The workload partitions cleanly across rows.
- No shared-memory synchronization.
- MPI communication volume is small.

This leads to near-linear strong scaling.

5.2 Why OpenMP performs poorly

- Locking overhead dominates.
- COO format produces highly irregular memory access.
- Multiple threads compete for the same row accumulator.

Thus OpenMP is not well-suited for this COO SpMV implementation.

5.3 Hybrid behavior

Hybrid combinations show:

- Best configurations: many MPI ranks, few OMP threads.
- Worst configurations: few MPI ranks, many threads (max lock contention).

This is consistent with known behavior of lock-heavy shared-memory SpMV.

6. Conclusions

- MPI parallelism provides strong scaling and significantly reduces runtime.
- OpenMP provides no measurable benefit for this implementation and often slows execution.
- Hybrid MPI + OpenMP shows that adding threads inside each rank is typically detrimental.
- For COO SpMV, **MPI-only parallelism** is the optimal strategy on Talapas.

MPI is the dominant source of speedup, while OpenMP introduces synchronization overhead that outweighs its benefits.