SpMV Performance Evaluation Report

1. Experiment Setup

We evaluate the performance of two sparse matrix vector multiplication implementations: COO-based SpMV and CSR-based SpMV. Both serial and parallel versions were implemented and tested on the `cant` matrix provided in the assignment. After running each implementation, the output vector was compared against the reference `ans.mtx` file using `diff`, and the results matched exactly, confirming correctness.

The program executes `MAX_ITER = 100` SpMV operations per timing measurement, since this constant is defined in the provided code. All timings therefore reflect the accumulated cost over 100 iterations.

We test thread counts:

$$\{1, 2, 4, 8, 16, 32, 64, 128\}.$$

Table 1 shows the most relevant timing fields extracted from the experiment.

Table 1: Key Timing Results (seconds)

Threads	COO Par	CSR Par	COO Ser	CSR Ser
1	6.25	1.18	1.49	1.14
2	3.21	0.59	1.47	1.12
4	3.93	0.29	1.48	1.13
8	2.60	0.15	1.48	1.16
16	1.66	0.07	1.48	1.12
32	0.91	0.04	1.48	1.12
64	0.49	0.06	1.48	1.12
128	0.33	0.058	1.49	1.13

2. Execution Time Behavior

CSR consistently performs better than COO for all thread counts. The difference is large even at one thread: COO requires 6.25 seconds while CSR takes only 1.18 seconds for 100 SpMV iterations. As the number of threads increases, both versions improve, but CSR improves significantly faster because it does not require synchronization.

By 32 threads, CSR reaches 0.04 seconds, while COO is still above 0.9 seconds. This reflects the extra locking required in the COO kernel and its scattered memory access pattern.

3. Serial Baseline

The serial runtimes remain nearly constant regardless of thread configuration:

- **COO Serial:** $\approx$1.48 s
- **CSR Serial:** $\approx$1.12–1.16 s

CSR is faster due to its row-based layout and better cache behavior.

4. Speedup Behavior

CSR shows strong parallel scalability. Speedup reaches:

- $\sim 2\times$ at 2 threads
- $\sim 7\times$ at 8 threads
- peak of $\sim 26\times$ at 32 threads

Beyond 32 threads the kernel becomes bandwidth-limited, so additional threads do not significantly reduce runtime.

COO exhibits limited speedup because each nonzero update requires acquiring a row lock. With highly irregular row occupancy, many threads attempt to update the same row and contention becomes the dominant bottleneck.

5. COO vs CSR Comparison

CSR outperforms COO for four main reasons:

1. The CSR kernel assigns entire rows to each thread, eliminating write conflicts.
2. All value and column arrays are contiguous, improving cache locality and prefetch efficiency.
3. No locks are required in CSR, whereas COO must lock for each nonzero.
4. CSR parallel scaling is limited only by memory bandwidth, not synchronization.

6. Plots

Figures 1–3 present the runtime measurements in visual form.

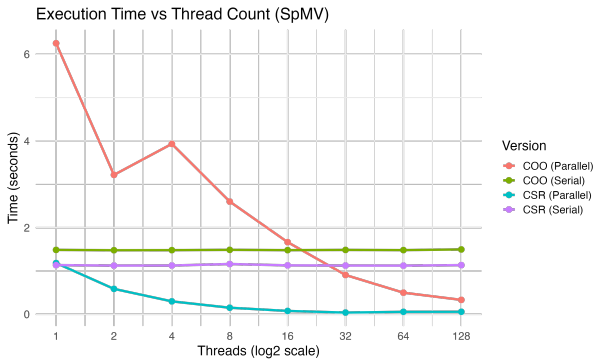

Figure 1: Execution time vs. thread count.

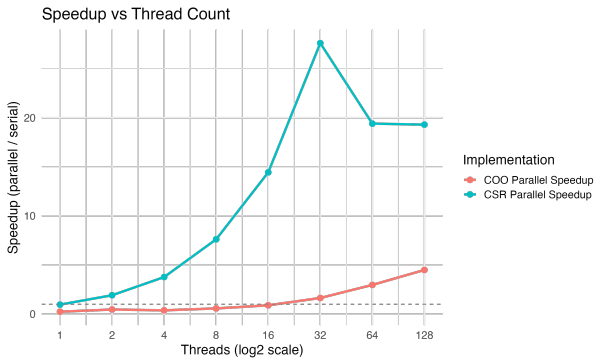

Figure 2: Speedup relative to the serial versions.

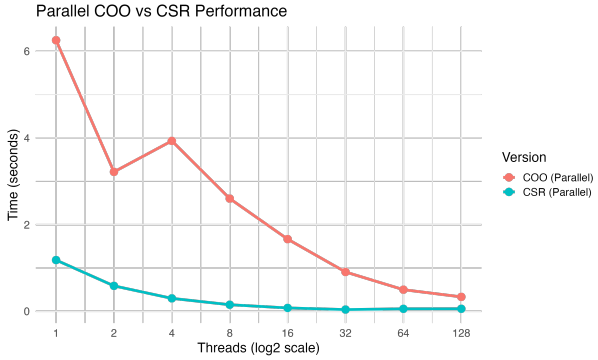

Figure 3: Parallel COO vs CSR runtime comparison.

7. Conclusion

CSR is consistently faster in both serial and parallel forms, and it scales efficiently across increasing thread counts. COO suffers from lock contention and scattered memory access, which prevents meaningful speedup on many-core systems. The experimental results confirm CSR as the preferred format for high-performance SpMV computations.