Scalability Analysis: OpenMP Implementation

Multithreading Overhead & Bandwidth Saturation

Progetto AMSC

November 27, 2025

Performance Overview: OpenMP (Float)

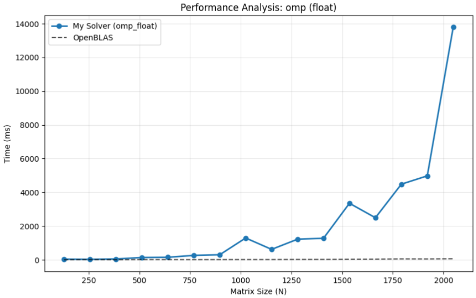

Figure: **Execution Time (ms)**

Significant instability (spikes) due to thread synchronization overhead.

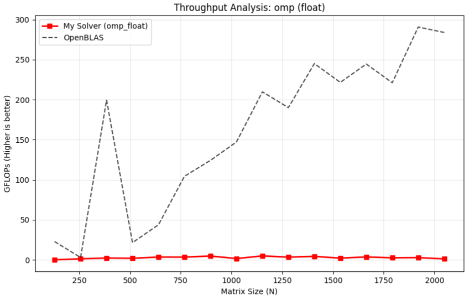

Figure: **Throughput (GFLOPS)**

Throughput is unstable and surprisingly lower than SIMD.

Quantitative Analysis: Float vs Double

Impact of Precision on Multithreading

Size (N)	Float	Double	Δ Overhead
128^3	28.7 ms	25.7 ms	-10% (Noise)
1024^3	1.24 s	1.48 s	+19%
1536^3	3.51 s	5.06 s	**Huge Spike**
2048^3	14.39 s	16.27 s	+13%

The "Parallelizing Garbage" Trap

Compared to Naive (47s), OMP is faster (14s, $\approx$ 3.3x speedup).

BUT it is slower than SIMD (1.07s)!

Why? We are parallelizing the inefficient *Naive* logic. Running 8 threads that all experience Cache Misses just saturates the Memory Bus 8 times faster.

Key Takeaway: More cores cannot fix bad memory access patterns.

Why is OpenMP slower than SIMD?

Simply adding `#pragma omp parallel for` does not magically solve architecture bottlenecks:

- **Memory Bandwidth Saturation:** Since the code is not blocked (tiled), every thread constantly fetches data from RAM. The shared memory bus becomes a traffic jam.
- **False Sharing / Cache Contention:** Multiple cores trying to access the same cache lines or evicting each other's data (Cache Pollution).
- **Thread Overhead:** For smaller N (or efficient N), the cost of spawning and syncing threads outweighs the compute benefits (evident in the spikes).

Bottleneck Shift

Naive: CPU Waiting for RAM.

$\downarrow$

OpenMP: *Multiple* CPUs Fighting for the *Same* RAM Channel.