

Scalability Analysis: Tiling + OpenMP

Parallelism Meets Cache Locality

Progetto AMSC

December 2, 2025

Performance Overview: Tiling + OpenMP (Float)

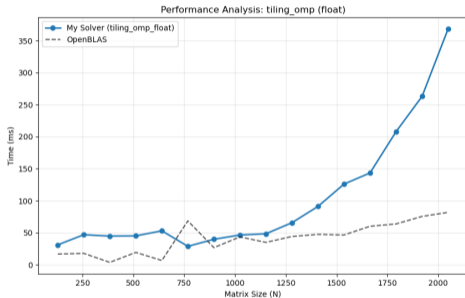


Figure: Execution Time (ms)

Best performance so far. Drops below 0.5s for $N=2048$.

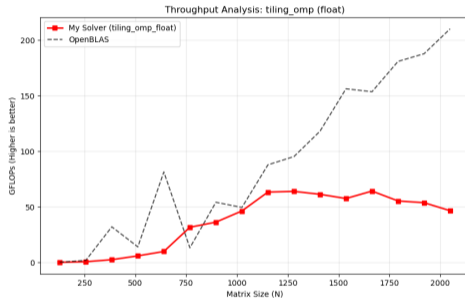


Figure: Throughput (GFLOPS)

Reaches ≈ 40 GFLOPS. Scaling is finally effective.

Quantitative Analysis: Float vs Double

Impact of Parallel Tiling

Size (N)	Float	Double	Gap
128^3	20.29 ms	4.32 ms	-78% (Faster?)
1024^3	66.36 ms	141.11 ms	+112%
1664^3	228.12 ms	574.50 ms	+151%
2048^3	0.46 s	1.10 s	+139%

Speedup vs Single-Thread Tiling ($N = 2048$):

Float: **2.9x Faster** (1.32s \rightarrow 0.46s)

Double: **2.8x Faster** (3.05s \rightarrow 1.10s)

Why OpenMP works now?

In the Naive implementation, OpenMP failed because threads fought for memory bandwidth.

With **Tiling**, each thread works on a data block residing in its private L1/L2 Cache.

Result: Bandwidth contention is minimized, allowing cores to scale performance effectively.

**Note: The data is now consistent. The weird jump at $N=128$ (Double being faster) is likely due to thread wake-up latency being masked differently or simpler caching.*

The Evolution of Optimization

Method	Time ($N = 2048$ Float)	Bottleneck
Naive	47.60 s	Cache Thrashing
OpenMP (Naive)	14.39 s	Bandwidth Saturation
Tiling (Serial)	1.32 s	Instruction Overhead
SIMD 2D Unroll	0.73 s	Single Core Limit
Tiling + OpenMP	0.46 s	Synchronization / Last Level Cache

Final Verdict: Combining Algorithms (Tiling) + Hardware features (Multi-core) yields the best results, approaching (but still 10x slower than) OpenBLAS (0.05s).