1 Parallel and Vectorized Matrix Multiplication

1.1 Introduction

This experiment investigates several optimization techniques for matrix multiplication, focusing on parallelism and vectorization. Four different implementations were evaluated:

- Serial baseline multiplication.
- Parallel multiplication using OpenMP.
- Parallel multiplication using `std::async` (executors) and mutex synchronization.
- A hybrid approach combining OpenMP parallelism with AVX2 SIMD vectorization.

Each method was benchmarked using square matrices of sizes 256×256, 512×512, and 1024×1024 on an 8-core machine. Metrics include execution time, speedup relative to the serial implementation, and parallel efficiency.

1.2 Experimental Results

1.2.1 Matrix Size: 256

Method	Time (s)	Speedup	Efficiency
Serial (Baseline)	0.003258	1.00×	–
OpenMP Parallel	0.008454	0.39×	4.82%
std::async Threads	0.011720	0.28×	3.47%
SIMD + OpenMP	0.001237	**2.63**×	**32.9%**

Table 1: Performance results for 256×256 matrices.

Analysis. Parallel methods (OpenMP and `std::async`) perform worse than the serial baseline. This is expected because the matrix is small and parallel overhead dominates execution. The SIMD + OpenMP approach, however, provides a clear performance improvement due to low overhead and efficient vectorized operations.

1.2.2 Matrix Size: 512

Method	Time (s)	Speedup	Efficiency
Serial (Baseline)	0.027643	1.00×	–
OpenMP Parallel	0.007487	**3.69**×	46.2%
std::async Threads	0.019344	1.43×	17.9%
SIMD + OpenMP	0.011588	2.38×	29.8%

Table 2: Performance results for 512×512 matrices.

Analysis. At this size, OpenMP begins to show a strong speedup as thread creation and scheduling overhead becomes small relative to computation time. The SIMD version remains fast, but is slightly behind OpenMP due to memory bandwidth limitations.

1.2.3 Matrix Size: 1024

Method	Time (s)	Speedup	Efficiency
Serial (Baseline)	0.291583	1.00×	–
OpenMP Parallel	0.086667	3.36×	42.0%
std::async Threads	0.089849	3.24×	40.6%
SIMD + OpenMP	0.080789	**3.61**×	**45.1%**

Table 3: Performance results for 1024×1024 matrices.

Analysis. With larger matrices, all parallel methods significantly outperform the baseline. The best performance is achieved by the SIMD + OpenMP implementation, reaching a speedup of 3.61×. Efficiency remains below ideal scaling due to memory bandwidth constraints and limited SIMD register utilization for very large datasets.

1.3 Overall Discussion

The results highlight the following:

- For small matrices, parallel overhead dominates, making serial and SIMD approaches superior.
- OpenMP provides the best performance for mid-sized matrices once parallelism becomes worthwhile.
- The combination of SIMD vectorization and OpenMP yields the best performance for large matrices.
- The `std::async` implementation performs significantly worse than OpenMP due to scheduling overhead and lack of work-sharing optimizations.
- Efficiency remains below the theoretical maximum (100%) because memory access becomes the primary bottleneck.

1.4 Conclusion

Parallelism and vectorization substantially improve matrix multiplication performance, but the benefits depend heavily on the problem size. Small matrices do not amortize threading overhead, while large matrices benefit significantly from both OpenMP parallelization and SIMD acceleration. The combination of

these two techniques (OpenMP + AVX2) consistently achieves the best results among the tested implementations.

`https://github.com/JorsuCT/Big_Data/tree/main/IndividualAssignment/task3`