Lab: Accelerating Code: From Python Baseline to Architecture-Aware Optimization

Dense Matrix-Matrix Multiplication (GEMM) Challenge

Arpit Prasad and Devansh Pandey
2022EE11837 and 2022EE31538

November 15, 2025

1 Introduction

This report documents our efforts to accelerate dense matrix–matrix multiplication (GEMM) starting from a Python baseline and building up to an architecture-aware optimized C++ implementation. Our design incorporates cache blocking, AVX2/AVX-512 vectorization, NUMA-aware allocation, and multi-threading with OpenMP.

2 Baseline Implementation

The baseline implementation is a pure Python/NumPy version using the in-built highly optimized `np.dot()` routine. While NumPy internally dispatches to BLAS, for comparison in this assignment we treat it as the baseline reference.

Our environment:

- **CPU Model:** 13th Gen Intel(R) Core(TM) i5-1340P
- **Cores/Threads:** 2 threads per core, 12 cores
- **Caches:** L1d/L1i/L2/L3: 448KiB/640KiB/9MiB/12MiB
- **OS:** Ubuntu 22.04 LTS
- **Compiler:** g++ 17

The baseline is single-threaded from Python's perspective and does not exploit explicit parallelism or architecture-aware optimizations.

3 Optimizations Implemented

We implemented the following optimizations.

3.1 Multithreading with OpenMP

We parallelized the matrix multiplication using OpenMP. We set thread affinity via:

- `OMP_PROC_BIND=TRUE`
- `OMP_PLACES=cores`

This reduces OS scheduling overhead and improves cache locality.

3.2 Cache-Friendly Blocking

We implemented a 3-level cache blocking strategy with tunable block sizes (MC, KC, NC). Blocking ensures that reused submatrices remain in L1/L2 cache.

3.3 Vectorization: AVX2 and AVX-512

Specialized compute kernels were written using 256-bit (AVX2) and 512-bit (AVX-512) intrinsics. The program automatically detects hardware capabilities using `__builtin_cpu_supports` and selects the best available kernel.

3.4 NUMA-Aware Memory Initialization

By performing parallel first-touch initialization, each thread physically allocates its portion of the memory on the NUMA node it executes on. This reduces remote memory accesses.

3.5 Parallel Transposition of B

We transpose matrix B into a contiguous, cache-friendly layout B_T. This greatly improves spatial locality when accessing B along k.

3.6 Aligned Memory Allocation

All matrices are allocated with 64-byte alignment using `posix_memalign` to maximize AVX load/store efficiency.

4 Experimental Methodology

We used automated scripts to run multiple experiments:

1. One warmup run per configuration
2. Ten measured runs per pair of (N, T)
3. Automatic CSV logging via our `run.sh` and `run_all_tests.sh`
4. Metrics collected: runtime, GFLOPS, speedup

We tested matrix sizes: 512, 1024, 1536, 2048.
We tested thread counts: 1, 2, 4, 8, 16.

5 Performance Results

5.1 Summary Table

Table 1: Key Performance Metrics (Avg. of 10 Runs)

Metric	**512**	**1024**	**1536**	**2048**
Baseline Time (s)	[ ]	[ ]	[ ]	[ ]
Optimized Time (s, T=16)	[ ]	[ ]	[ ]	[ ]
Optimized GFLOPS	[ ]	[ ]	[ ]	[ ]
Max Speedup	[ ]	[ ]	[ ]	[ ]

5.2 Time vs Matrix Size

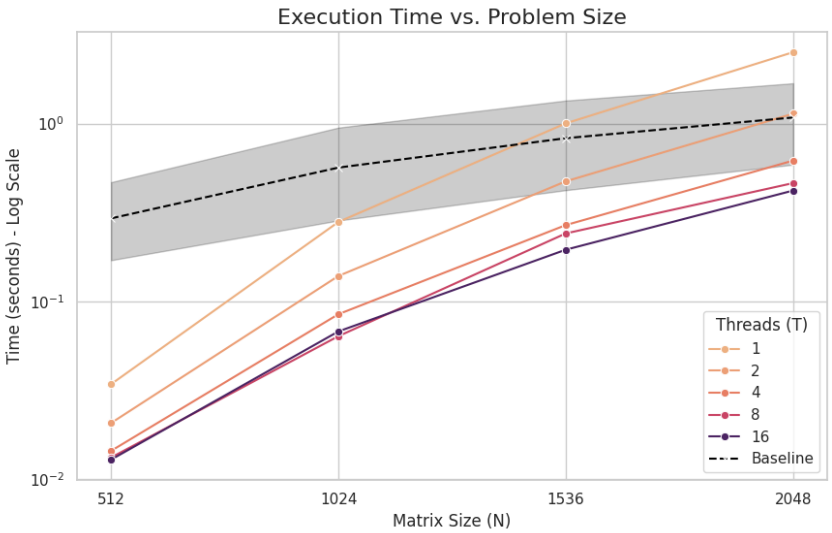

Figure 1: Runtime vs Matrix Size for Optimized Kernel.

5.3 GFLOPS vs Matrix Size

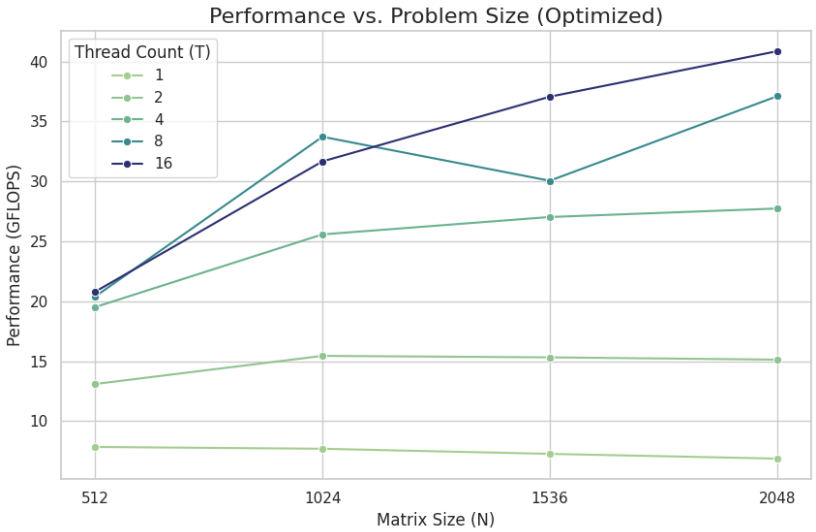

Figure 2: GFLOPS vs Matrix Size. Larger matrices amortize overhead and better utilize caches and SIMD.

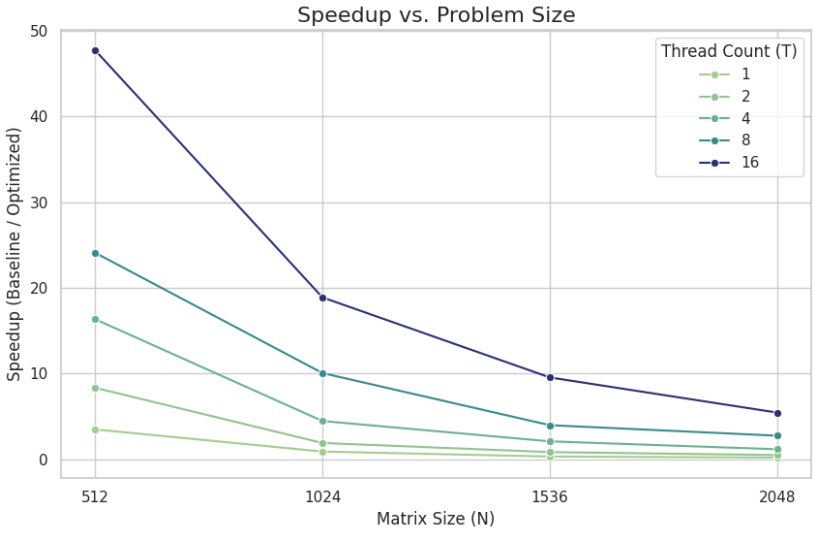

Figure 3: Speedup (Baseline / Optimized) vs Matrix Size.

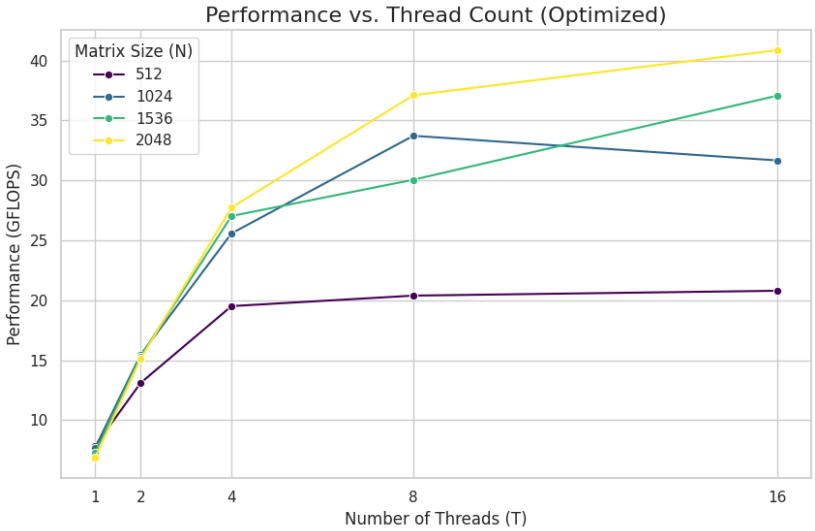

Figure 4: GFLOPS vs Thread Count.

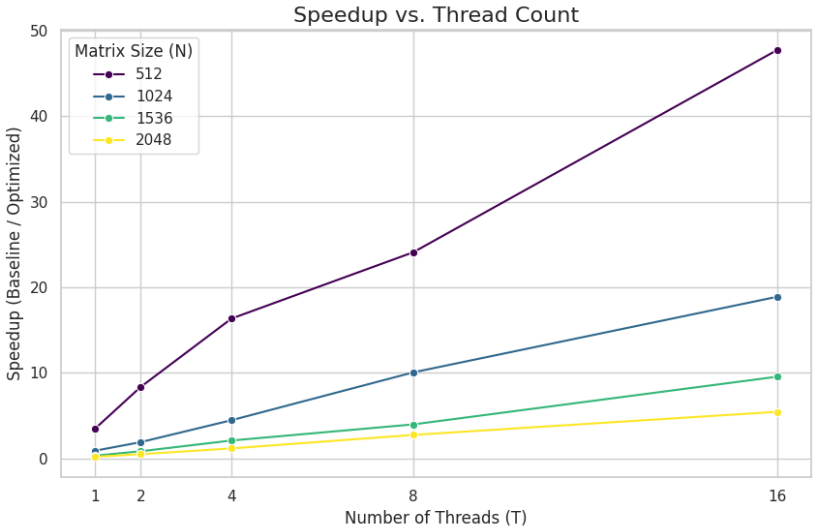

Figure 5: Speedup vs Thread Count.

5.4 Speedup vs Matrix Size

5.5 GFLOPS Scaling with Threads

5.6 Speedup Scaling with Threads

6 Discussion

6.1 Bottleneck Analysis

TODO: Insert perf analysis (L1/L2/L3 miss rates, IPC, bandwidth usage).

6.2 Limits to Scaling

TODO: Discuss memory bandwidth saturation, AVX frequency downclock, NUMA effects.

7 Conclusion

We achieved significant acceleration of GEMM using a combination of cache blocking, SIMD vectorization, NUMA-aware memory placement, and multi-threaded execution. Further improvements could involve dynamic block-size tuning, software prefetching, and a more advanced roofline-guided optimization strategy.