Task 2: Performance Benchmark of Dense and Sparse Matrix Multiplication

Ángela López Dorta

November 22, 2025

Abstract

This report presents a performance analysis of several matrix multiplication algorithms implemented in Java. Dense multiplication methods include a baseline triple-nested loop, a cache-friendly variant, and tiled versions with different block sizes. Additionally, sparse matrix multiplication is implemented using the Compressed Sparse Row (CSR) format and benchmarked using a real-world Matrix Market dataset. The objective is to evaluate the performance implications of algorithmic optimizations and sparsity on execution time. Experimental results are collected, aggregated, and visualized through Python scripts, producing both summary CSV files and plots.

1 Introduction

Matrix multiplication is a fundamental operation in numerical simulation, scientific computing, and machine learning. The classical dense algorithm has cubic time complexity, making optimization strategies essential when working with large datasets. Sparse matrices, where most entries are zero, offer opportunities for significant performance improvements by reducing the number of operations.

This task evaluates multiple dense matrix multiplication methods and a sparse CSR-based multiplication. The performance of each method is measured across several matrix sizes or sparsity levels. A real sparse matrix from the Matrix Market repository is used to assess the practical benefits of sparsity.

2 Methods

2.1 Dense Matrix Multiplication

Three dense multiplication strategies were implemented:

Baseline. A standard triple-nested loop algorithm that directly computes each value in the result matrix. This method provides a reference for comparison.

Cache-Friendly. The loop order is rearranged to improve memory locality. Accessing elements in a cache-friendly pattern reduces cache misses, especially for large matrices.

Tiled Multiplication. Matrix multiplication is performed in blocks of size $t \times t$. Blocking improves data reuse by operating on smaller submatrices that fit better into cache. Tile sizes of 16, 32, and 64 were tested.

2.2 Sparse Matrix Multiplication (CSR Format)

Sparse matrices are stored in the Compressed Sparse Row (CSR) format. CSR uses three arrays:

- `rowPtr`: starting index of each row's non-zero entries,
- `colIndex`: column index of each non-zero element,
- `values`: non-zero values themselves.

The multiplication computes $y = Ax$, where A is sparse and x is a dense vector. Only non-zero elements are processed, reducing computational cost from $\mathcal{O}(n^2)$ to $\mathcal{O}(\text{nnz})$, where `nnz` is the number of non-zero entries.

The matrix `mc2depi.mtx` from the Matrix Market dataset is used to benchmark the sparse implementation.

2.3 Benchmark Strategy

All benchmarks were executed in Java using `System.nanoTime()` to measure execution time. For each algorithm:

- multiple runs were performed,
- the mean, best, and worst times were recorded,
- results were exported to `benchmark_task2_results.csv`.

A Python script processed the CSV file and generated plots and summary tables.

3 Results

3.1 Dense Multiplication

Figure 1 shows the execution time of dense algorithms as matrix size increases. The baseline method grows cubically as expected. Cache-friendly and tiled methods demonstrate improved performance for larger matrices due to reduced cache misses and better data locality.

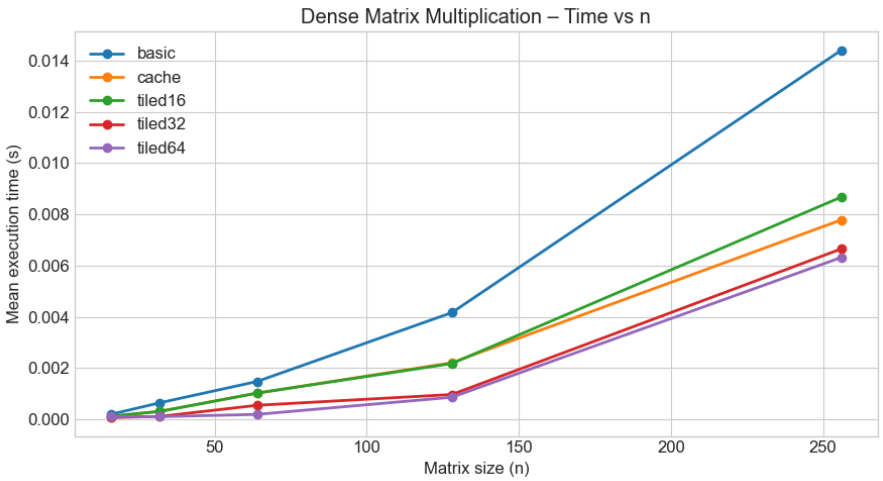

Figure 1: Dense multiplication: execution time vs. matrix size.

3.2 Dense Speedup

Figure 2 compares the relative speedup of optimized dense methods against the baseline. Tiled implementations achieve significant improvements as matrix size increases.

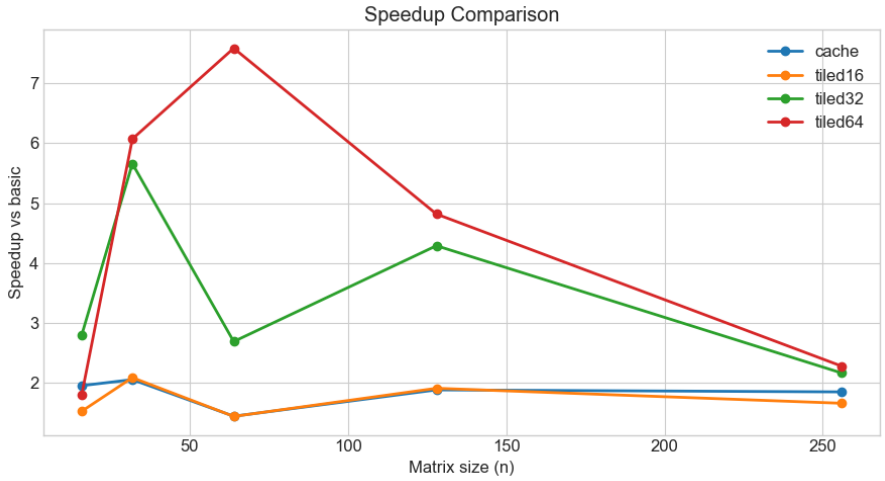

Figure 2: Speedup of optimized dense algorithms relative to the baseline.

3.3 Dense vs. Sparse Performance

Figure 3 compares dense and sparse multiplication performance. Sparse multiplication is significantly faster due to processing only non-zero entries.

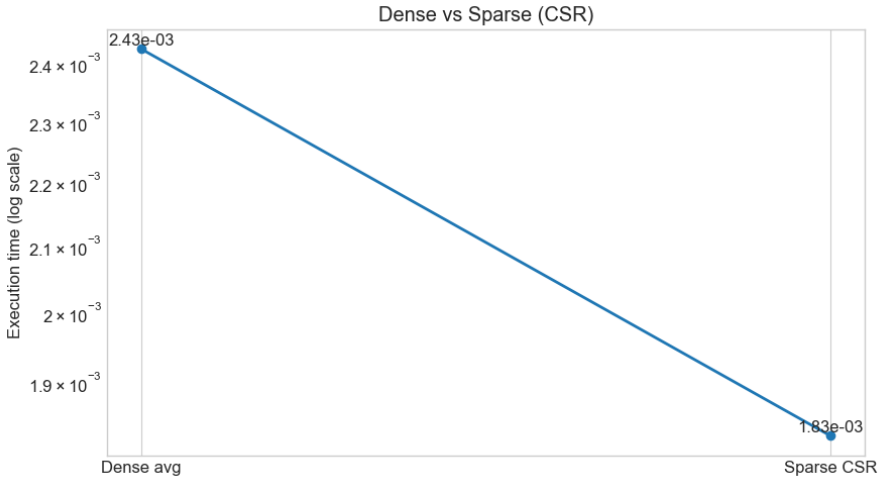

Figure 3: Comparison between dense and sparse multiplication times.

4 Discussion

Dense multiplication exhibits cubic runtime growth, with significant differences in constant factors between implementations. Cache-friendly and tiled algorithms outperform the baseline, especially for large matrices. Tile size plays an important role: larger tiles improve locality but may increase overhead for smaller matrices.

Sparse multiplication using CSR format demonstrates substantial performance benefits for real-world sparse data. The `mc2depi.mtx` matrix contains a small number of non-zero values relative to its size, resulting in much faster computation compared to dense multiplication.

The results highlight the importance of data structure selection and algorithmic optimization when scaling matrix operations.

5 Conclusion

This work benchmarked several matrix multiplication algorithms in Java. Dense multiplication benefits noticeably from cache-aware techniques and tiled processing. Sparse matrix multiplication using CSR format offers dramatic speedups when operating on real-world sparse datasets.

The automated benchmarking and plotting scripts provide a reproducible pipeline for performance analysis. Future improvements could include parallel execution, additional tile sizes, or integration with optimized native libraries.

References

[1] Mc2depi Matrix Market Dataset, University of Florida Sparse Matrix Collection. [2] J. Dongarra et al., "Anatomy of High-Performance Matrix Multiplication," 2017. [3] Goto & Van de Geijn, "Anatomy of High-Performance Matrix Multiplication," ACM TOMS.