Assignment 1

Performance evaluation of the two matrix multiplication.

David Padilla Orenga
Ignacio Pastore Benaim

1 Introduction

This report evaluates the performance of three different methods for performing matrix multiplication. The first method uses traditional 2D arrays in C++ (**2D vec**), the second employs a flattened approach, using a 1D array (**1D vec**), and the third utilizes the Eigen-based C++ library (**EIG**). Tests are conducted across various matrix sizes and optimization levels (-O2, -O3), with performance measured in terms of real times. The analysis highlights significant performance differences observed across two different hardware architectures. Focus will be on heap storage, since tests on stack storage were performed, but $N = 500$ could not be exceeded due to memory limitations.

2 Experimental Setup

The experiments were conducted on two machines. Their specifications are summarized in Table 1. **Matrix Sizes and Executions:** The matrix sizes tested were *N = 100, 250, 500, 600, 700, 800, 900, 1000, 1500.* For each version, different levels of compiler optimizations (*standard, '-O2', '-O3'*) will be performed. Each experiment was executed *10 times* to calculate the mean and standard deviation for real times.

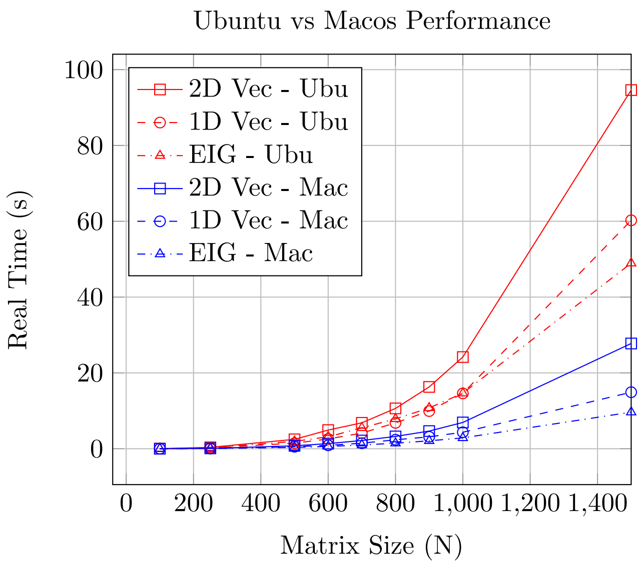
Specification	**Ubuntu (x86_64)**	**Mac (M1)**
Operating System	Ubuntu 20.04 LTS	macOS Big Sur
Processor	Intel Core i7-6700HQ @ 2.60GHz	Apple M1
Cores/Threads	4 Cores / 8 Threads	8 Cores (4 performance, 4 efficiency)
Memory	15.5 GiB	8 GB
Architecture	x86_64	ARM64

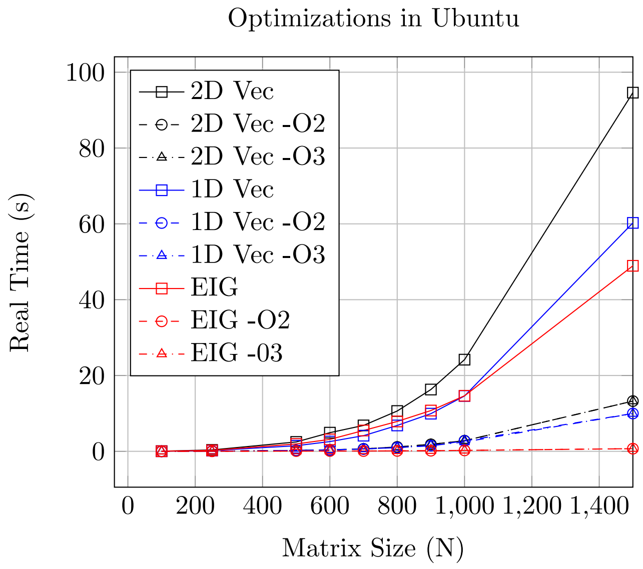
Table 1: Specifications of the two systems used for testing.

3 Performance Results

Figure 1 summarizes the real-time performance of the implementations across both platforms and optimizations. As the matrix size increases, noticeable trends emerge:

- The **Eigen library** consistently outperforms the standard implementations for all matrix sizes. This is especially evident for larger matrices ($N > 1000$), where the difference in performance becomes more pronounced.
- **2D vs 1D flattened array**: The 1D flattened vector approach shows better performance than the standard 2D array approach, particularly for larger matrix sizes, due to improved memory locality and reduced cache misses (data is stored continuously), but remains slower than Eigen.
- **Ubuntu vs. MacOS**: MacOS, with its M1 chip, consistently outperforms the Ubuntu machine for larger matrix sizes, particularly when using the Eigen library. From matrix size $N = 600$ onwards, MacOS achieves real times that are approximately half, or even less, compared to Ubuntu across all implementations. The performance boost with the '-O2' and '-O3' optimizations is especially significant.
- **Optimization Impact**: Compiler optimizations ('-O2', '-O3') significantly reduce execution time in a similar way across all implementations. The most notable improvements are seen with the Eigen library, where '-O3' provides the best performance across both platforms especially for larger matrices. For instance, at N = 1400, Eigen with '-O3' performs almost the same way as for small matrixe. Standard 2D arrays, even with optimizations, perform worse than the 1D flattened implementation.





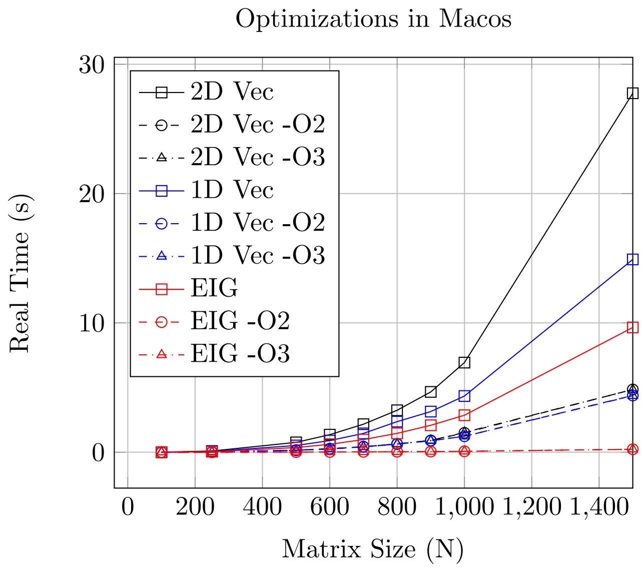


Figure 1: Real time comparison for different implementations.

4 Conclusion

In this study, matrix multiplication was implemented using heap-based allocation, flattened vectors, and the Eigen library. Heap allocation provides flexibility for handling larger matrix sizes, avoiding stack overflow, while flattened vectors improve memory locality by storing elements continuously. The Eigen library further optimized memory and instruction-level operations, demonstrating efficient handling of large matrices. The standard matrix multiplication algorithm has a time complexity of $\mathcal{O}(N^3)$, meaning that execution time increases cubically with matrix size. This behavior was observed in the results, particularly for matrix sizes above $N = 500$. Although Eigen cannot alter this inherent complexity, its optimizations significantly reduce overhead, leading to lower real times, especially for larger matrices.

Overall, the Eigen library outperformed both the standard heap and flattened vector implementations across all matrix sizes. Compiler optimizations (-O2, -O3) provided substantial performance gains, with Eigen benefiting the most. System time remained consistently low across all tests, indicating minimal overhead from memory allocation. Additionally, MacOS (Apple M1) showed superior performance compared to Ubuntu (x86_64), particularly for larger matrices, likely due to the M1 chip's architecture and its ability to take full advantage of Eigen's optimizations. In conclusion, Eigen, combined with compiler optimizations, offers the best performance for matrix multiplication across both platforms. As matrix size increases, the performance gap between Eigen and the standard implementations grows, with the M1 architecture further enhancing performance for larger matrices.