Programming and Architecture of Computing Systems

Méndez Carter Diego - 960616

Laboratory 1 5 Oct 2025

Introduction

In this first laboratory session we were introduced to the command shell and the command-line workflow that will underpin the subsequent labs. We explored essential shell commands to navigate the filesystem, manage files, and execute programs, with the goal of becoming fluent in a reproducible, scriptable working environment.

A second core objective was to understand the role of a compiler in a C++ development pipeline. We wrote and compiled simple sequential programs, linked against an external library, and evaluated basic performance characteristics. Together, these activities established the practical foundations of shell literacy, compilation, linking, and measurement that we will build on in later sessions.

The present report is for showing the results from comparing the execution of the sequential programs done without a library vs the library.

Description of the experiment

Matrix multiplication is a very commonly used operation throughout computer vision, robotics and graphics. In this lab we implemented two single sequential programs: one with a generic algorithm written by hand using simple loops which should be compared with another implementing the EIGEN library.

The operation in between the NxN matrices to be multiplied showed on figure 1.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

$$1x5+2x7=19$$
$$1x6+2x8=22$$
$$3x5+4x7=43$$
$$3x6+4x8=50$$

Figure 1. Multiplication of NxN Matrices

This diagram can be shown in the following for loop snipset that illustrates the method.

```
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            double acc = 0.0;
            for (int k = 0; k < N; ++k) {
                acc += A[i][k] * B[k][j];
            }
            C[i][j] = acc;
        }
```

Figure 2. For loop implemented for the lab session

We are interested in the efficiency of the matrix multiplication algorithm; therefore, we measure only the computation time of generating the result matrix, and compare it against an optimized library implementation.

On this point I am expecting slower timings of the “handmade” code vs the EIGEN library.

Results

For these results I selected 10 samples of 128, 256, 512 and 1028 NxN matrices.

With the objective of watching some similar time complexities near to the $O(N^3)$ (cubic time).

Obtaining the following numbers:

N	Mean_Normal (ms)	Sd_Normal	Mean_Eigen (ms)	Sd_Eigen
128	917.797	78.886	265.444	71.363
256	7.477.368	118.185	1.067.299	775.389
512	66.130.905	661.697	4.597.086	1.211.135
1024	1.359.900.469	59.011.927	28.351.364	191.678

Figure 3. Mean and standard deviation numbers from the time data

The table shows the average execution times (mean ± standard deviation) obtained from ten runs for each matrix size N. As expected, the execution time increases rapidly with the matrix dimension for both implementations, reflecting the cubic growth $O(N^3)$ of matrix multiplication.

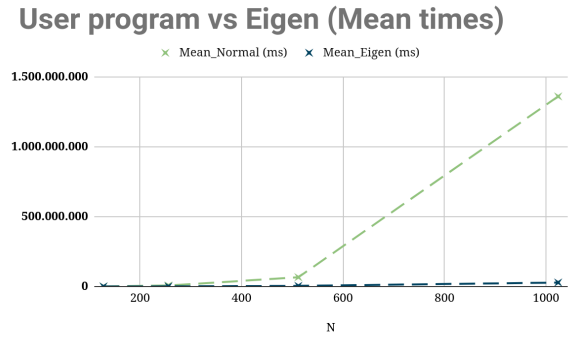

Figure 4. Mean behaviour in between programs

The graph clearly shows a similar trend: as the matrix size N×N increases, the execution time of the simple loop implementation grows following a cubic pattern.

Finally, the execution times show a clear difference between the two implementations: the Eigen version is significantly more efficient than the simple loop algorithm (For loops time / Eigen).

N	Exucation times ratio
128	3,458
256	7,006
512	14,385
1024	47,966

Figure 5. Executing Times ratio Normal/Eigen

Discussion and conclusion

The results obtained for N confirm the expected $O(N^3)$ growth of matrix multiplication. Both implementations show the same cubic trend, but the simple loop version grows much faster, while Eigen maintains considerably lower times due to better optimization.

These differences arise not from the number of operations, but from **how efficiently they are executed**. As Parthasarathi explains[1], the **instruction count** depends on the algorithm, the programmer's skill, the compiler's ability to optimize, and the processor's **instruction set architecture**. Eigen exploits these factors, whereas the simple loop executes more instructions with less efficient memory access.

In summary, both codes have the same theoretical complexity, yet Eigen achieves superior real performance by minimizing instruction count and optimizing data.

References:

[1] R. Parthasarathi, *Computer Architecture: Engineering and Technology*, Univ. of Maryland. Available: http://www.cs.umd.edu/~meesh/cmsc411/CourseResources/CA-online/