Benchmark of matrix multiplication (3)

Jaime Ballesteros Domínguez

October 29, 2023

Abstract

Continuing with our study of programming languages, we must consider not only which one may be better depending on the task but also their inherent capabilities for performance improvement. To do this, we decided to convert different matrices, some sparser and others denser, into COO, CRS, and CCS formats. Once these conversions were performed, we proceeded to execute a matrix multiplication while monitoring their execution times. Once these experiments were completed, we were able to assess the influence of the number of non-zero values contained in the matrices, thanks to some outliers, while also determining the size limits that our code can handle.

1 Motivation

The reason for conducting this experiment arises from the need to optimize a series of processes, which on their own can be either very costly or outright impossible to perform. Thanks to the development of this practice, we have been able to firsthand see how the proper implementation of alternatives can help make our code more efficient for upcoming computationally expensive experiments.

2 Methodology

First and foremost, this experiment was conducted in IntelliJ IDEA Community Edition, so we will need to install the JetBrains Toolbox application following the steps in this tutorial to get started. Once this tool is installed, we will select matrices of different sizes and with varying quantities of non-zero elements from the following link.

In this experiment, we have used matrices with up to 14,000 rows and columns because larger sizes resulted in size-related issues, which we will address in the final section. We have also attempted to select matrices with varying quantities of non-zero elements to assess their impact on process execution.

3 Experiments

As mentioned earlier, we evaluated the execution times of matrix multiplication in different CRS and CCS formats. These experiments are measured in milliseconds because smaller sizes were completed in very short times.

As we can see in Figure 1, the execution times generally correlate with the matrix sizes. However, we can observe a minor anomaly, which we can better understand with Figure 2.

As we can see in Figure 1, there is a slight decrease in the execution time, even as the matrix size continues to increase. Figure 2 reveals a sudden increase in the number of non-zero elements, which likely influenced the matrix multiplication's execution time.

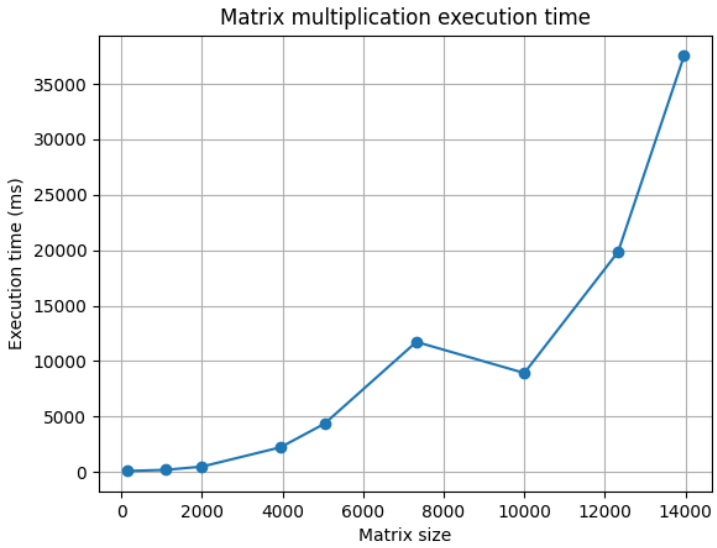

Figure 1: Runtime by Matrix Size.

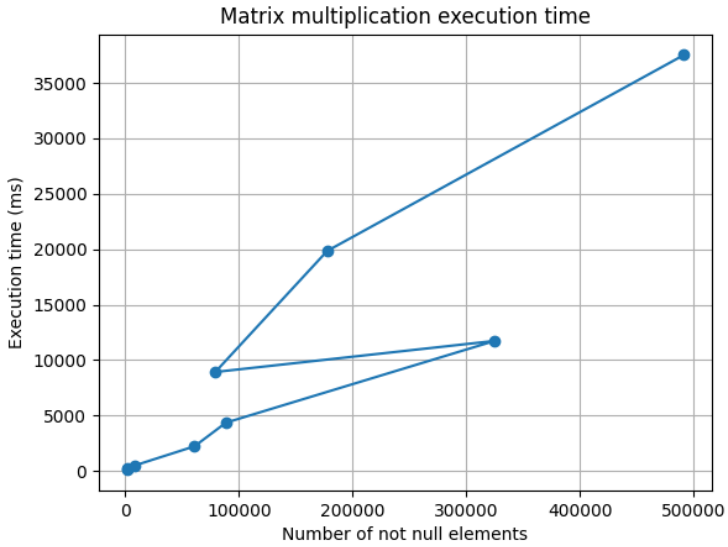

Figure 2: Runtime by Number of Non-Zero Elements.

4 Conclusions

In summary, our work involved analyzing the performance of matrix multiplications with different storage formats, density, and sparsity of non-zero elements. To conduct this experiment, we used a set of matrices with various sizes and values, performed their corresponding matrix multiplications, and saved the results.

The results obtained show an exponential growth in execution times as the size of the matrix to be evaluated increases. Additionally, we can also conclude that large volumes of non-zero elements result in high computational costs.

5 Future work

This experiment serves as the first step in understanding how important it is for our code to be well-optimized when executing processes. As this is a specific case of matrix multiplication, we could also perform similar experiments with other processes.

One of the areas for improvement in our code is that we couldn't perform tests with the matrix recommended by the tutor. This was because when certain dimensions were exceeded, the program returned a heap space error. Therefore, for future work, we would need to investigate how to address these dynamic storage space problems, as we do not know whether it is due to poor code development or an insuperable size limit that the program can handle.