Basic Matrix Multiplication in Different Languages

Radosława Żukowska

October 20, 2024

Big Data

Grado en Ciencia e Ingenieria de Datos
Universidad de Las Palmas de Gran Canaria

Abstract

Matrix Multiplication is one of those core building blocks in lots of domains such as machine learning and computer graphics, but where performance becomes especially important when size starts to grow. The report contrasts functionality with distinct implementation of matrix multiplication algorithms composed in C, Python and Java. All the experiments are applied to square matrices, from size 100x100 up to 1000 x 1000 running on equivalent hardware platform in order that all results will be fair compared.

C, as a memory-prone language though providing high efficiency in computation due to its direct management of memory and low-level operations always showed the highest speed than the other two languages. Java was able to demonstrate good performance mostly because of its garbage collection-based strong memory management and Java Virtual Machine optimizations, being only a little off from C. Python ended up way behind the other languages due to having much slower runtime speed but has exceptional rapid application development capabilities for those with higher priority on fast write-time over read-time in deployed applications. Furthermore, except for small matrices the impact of disabling Python's garbage collector on performance was negligible.

As you can see C is the fastest language followed by Java and Python. Although Python can be considered higher-level and more friendly, but with a bit of sacrifice on performance (it is generally way slower than C or even Java), java has a very good balance, fast enough for most tasks while it still have user friendly syntax compared to other low level languages such as C.

1 Introduction

Matrix multiplication is a fundamental operation in many areas of computer and data science. It is utilized in machine learning for training neural network, in computer graphics for transformations and projections in three-dimensional graphics and for coloring and shading figures. The standard method for matrix multiplication has a time complexity of $O(n^3)$, which is quite poor compared to other algorithms. As the size of the matrices increases, the time required for calculations rises rapidly. This poses a challenge in fields that rely heavily on multiplying large matrices, making optimization crucial for satisfactory results. Considering machine's limited resources such as memory and processing power, there is a need for more optimized algorithms and implementations across various platforms. The necessity for improved performance is not only relevant in scientific or professional contexts but also for students and novices entering these fields. Usually, they lack access to powerful resources like computational clusters or advanced processing units.

Optimization can be approached from various angles: refining the algorithm itself, selecting the best execution platform, or developing suitable hardware. A significant factor in this exploration is the choice of programming language. Many languages are available today, each with unique characteristics tailored to specific purposes. This report will focus on three popular high-level programming languages: C, Python, and Java.

C is known for its speed and efficiency, providing programmers with significant control over memory management. However, it lacks built-in memory management and modern features like Object-Oriented Programming (OOP). Python, in contrast, is versatile and easy to understand, with a rich variety of libraries, making it a popular choice in data science. Yet, it is generally slower than C and may not be as efficient for computationally intensive tasks. Despite this, Python's simplicity and libraries, such as NumPy, make it appealing for rapid prototyping and data analysis. Java strikes a balance, offering both performance and ease of use. As a statically typed, object-oriented language, Java features robust memory management through garbage collection and optimization capabilities via the Java Virtual Machine (JVM), making it suitable for numerical computations, though it may not match C's speed.

This report will compare matrix multiplication implementations in C, Python, and Java, with a focus on execution speed.

2 Methodology and code development

The experiment consists of implementation of the same algorithm for matrix multiplication in three different programming languages: Python, C and Java. Each implementation is tested on various sizes of the matrices.

All matrices are square matrices, which sizes are from 100x100 to 1000x1000.
They will be run on the same machine which is a portable computer with the processor: Intel(R) Core(TM) Ultra 7 155H with 32GB of RAM. Python and Java are run on the Windows 11 Professional operating system and C is run in the same system but using Windows Subsystem for Linux.
Each programming language has its properties that impact how the code performs. In order to obtain as reliable results as possible, it is crucial to understand and optimize the benchmarking accordingly.

2.1 Python

Gathereing data for benchmarking with Python is done with Pytest Benchmark ran with flag –benchmark-only. Python utilizes has built-in garbage collector function which runs cyclicly and helps in memory management, however as it runs in the background normally it can slow down some executions of the code. Therefore firstly is tested which way is better in the context of matrix multiplication. Then the better one is used to compare with other languages.

2.2 C

C benchmarking and measuring of the results is done using Perf. C does not have an built-in memory management therefore here it isn't necessary to take care of it here. Developing the same code in C is a bit more complicated than in Python. To separate the implementation of the matrix multiplication from the test, it was necessary to use pointers, so that the matrices can be passed to a function as an argument. The property of the language that the programmer allocates and deallocates dynamic memory is evident in the implementation.

2.3 Java

Java Microbenchmark Harness tool is used for measuring the performance of algorithm in Java. It is used in Mode.SampleTime to obtain the results to compare execution time with other languages. As Java is an OOP language, therefore the implementation of the function for matrix multiplication is similar to the one in Python. The matrices can be passed as arguments to the function and also returned result can be a matrix.

3 Experiments and analysis of results

Firstly the experiment to check whether to enable or not the garbage collector in Python was conducted and then the results of benchmarking for each language were collected and compared.

3.1 Python

The test on garbage collector with Python was done using the flag –benchmark-disable-gc to run without it running in the background during the benchmark. It was done for matrices with dimentions ranging from 16x16 to 1024x1024. The Figure 1 shows the results on a graph.

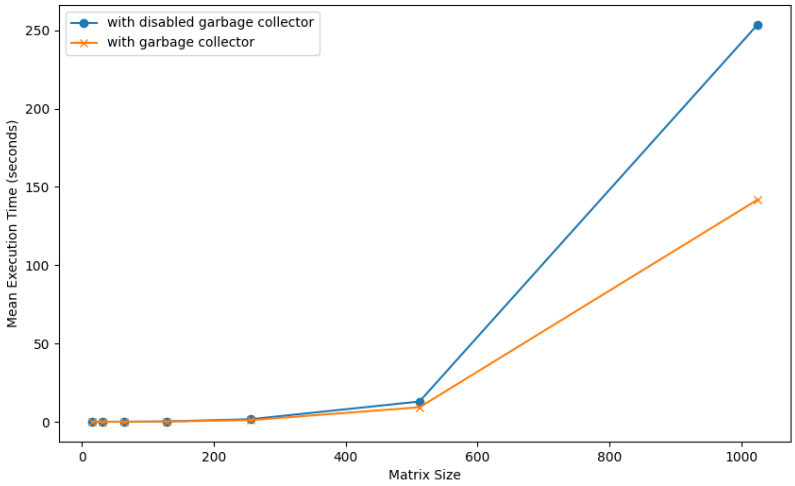

Figure 1: Comparison between execution in Python with garbage collector on and disabled

When the size of the matrices increases it is evident that running the benchmark with enabled garbage collector gives better results. This is because when the matrices are big, the importance of memory management increases, so that the values can be accessed faster. When the matrices are smaller the difference is almost not noticeable.

3.2 Comparison between C, Java and Python

Then the experiments for all languages were run for matrices with size from 100 to 1000. The Figure 2 shows the graph which shows the average time of execution of the matrix multiplication for all three languages.

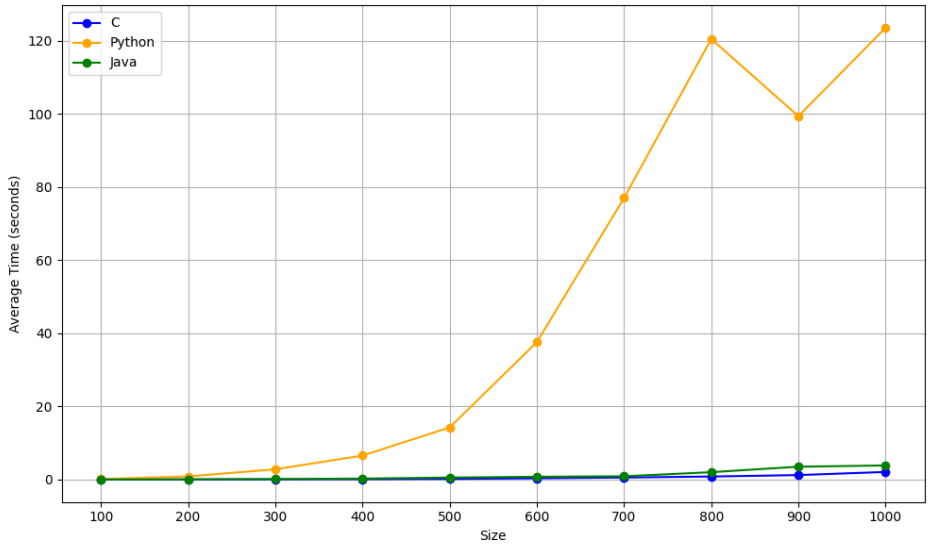

Figure 2: Comparison between languages on matrix multiplication algorithms with standard y-axis scale

As provided by a graph above Python performs significantly poorer than Java and C. When the matrices have size up to 200x200 the difference is almost unnoticeable, but then it becomes evident.
To get a better view on the performance difference between Java and C, on the Figure 3, there is a graph of the same values but in logarithmic scale on y-axis.

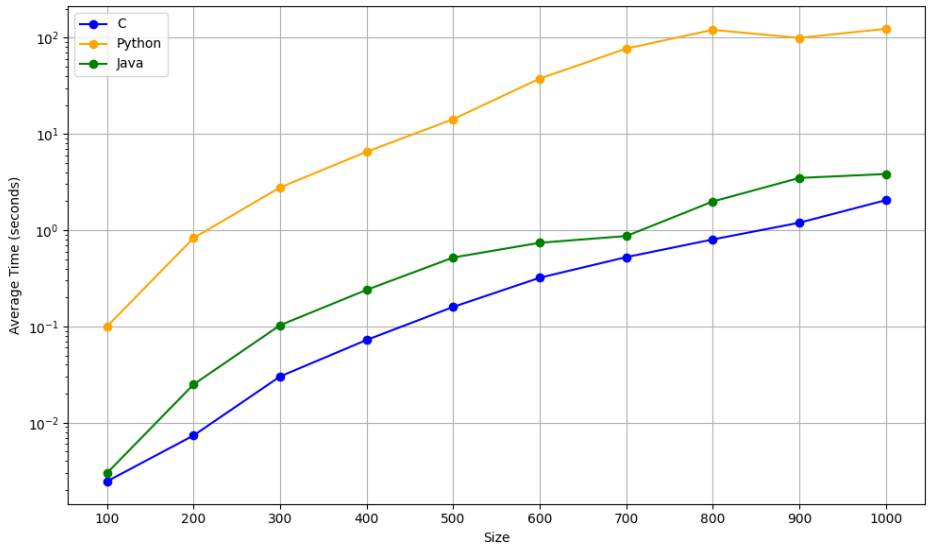

Figure 3: Comparison between languages on matrix multiplication algorithms with logarithmic y-axis scale

Here we can see that the fastest is C programming language followed by Java. Their performance is quite close to one another. The logarithmic scale enables for more precise comparison between languages, because as mentioned in the introduction, matrix multiplication has a cubic time complexity, therefore it compounds

and exponentiates the difference in the performance even from single operation.

4 Conclusion

The experiment comparing matrix multiplication performance across three programming languages—C, Python, and Java — shows the impact that language choice has on computational efficiency. C, known for its direct memory management and low-level operations, consistently outperformed both Python and Java, as expected. Java, with its balance of performance and ease of use, came close to C's execution times, proving to be a solid choice for numerical computations that benefit from garbage collection and JVM optimizations. Python, although much slower than both C and Java, demonstrated its value for ease of development and prototyping, but at the cost of speed. Its performance, particularly with larger matrices, was significantly slower. However, enabling Python's garbage collector showed that it could mitigate memory management issues for larger matrices, improving performance somewhat when handling more memory-intensive tasks. In summary, although C programming language performs faster, for the ease of implementation, Java can be considered its substitute to use for matrix multiplication while not sacrificing as much on the performance.

5 Future work

To continue exploring the topic of matrix multiplication and speeding it up, there are many more areas that one can consider. From trying out more optimized algorithms, to building computing clusters and connecting multiple computers together. Also using graphical cards for this operation may bring better results. It might be worthy to also try other programming languages and see how they will perform.

6 Appendices

GitHub repository

Here is whole code used in experiments and in obtaining the results together with the results.

https://github.com/radoslawazukowska/ulpgc-big-data-ind