Universidad de Las Palmas de Gran Canaria
Grado en Ciencia e Ingenier´ia de Datos

Big Data

Matrix Multiplication

Zuzanna Furtak

Github link

Contents

1 Abstract

Matrix multiplication is a fundamental operation, crucial for various computational tasks, including data analysis and machine learning. This report compares how Python, Java, and C handle that operation, considering their individual strengths. Python's simplicity makes it popular for quick development, while Java's cross-platform abilities and performance are notable. C, known for its speed and resource control, excels in system programming.

By measuring execution times on matrices up to 200x200, the study sheds light on the best language for matrix operations. Developers can use this knowledge to make conscious decisions based on performance, development ease, and resource management requirements.

Results highlight Python's universality, Java's scalability, and C's performance, guiding developers towards optimal language selection for matrix operations. Future work may involve enhancing efficiency through parallel processing and optimization algorithms across Python, Java, and C.

2 Introduction

Benchmarking of programming languages for matrix multiplication is an important way for improving computational efficiency, particularly in areas such as data science. By analysing the methods used by Python, Java and C to handle this operation, we gain insights into their strengths and capabilities. Python's reputation for user-friendly interfaces and extensive library support streamlines development workflows, promoting rapid prototyping and ease of use. By contrast, Java is proving to be a versatile player in the field, using its cross-platform capabilities and performance-oriented design to skilfully serve many applications. Meanwhile, C's resource management and high-speed processing, rooted in its low-level capabilities, positions it as the best choice for tasks requiring optimised performance and efficient memory handling.

3 Problem statement

This study of Python, Java and C encapsulates a delicate balance of performance metrics within a cohesive framework. By methodically examining execution times in the domain of matrix multiplication, this research aims to provide a comprehensive perspective on the language selection process. Through this study, software developers will be able to deepen their understanding of how to carefully make language preferences, thoughtfully align them with project-specific requirements, and increase the accuracy and efficiency of matrix operations.

4 Methodology

The experiments involved measuring the execution time of matrix multiplication operations in Python, Java and C on a Macbook Pro with an Apple M3 Pro chip and 18GB of RAM. The matrix sizes used to evaluate the performance of the algorithms range from 5x5 to 200x200. Larger sizes proved too demanding for Python. A comprehensive analysis of the performance of matrix multiplication operations was carried out using three different methods tailored to each programming language. With warm-up iterations and many rounds to ensure accurate results. Allowing an insightful comparison of execution times between Python, Java and C.

4.1 Python

Within Python, the benchmarking process involved using the Pytest framework together with a specific benchmark function dedicated to measure time of matrix multiplication. Pytest facilitated the efficient measurement of execution times, including minimum and maximum time, median and mean, allowing for comprehensive performance analysis.

4.2 Java

In Java, the Java Microbenchmark Harness (JMH) framework was used alongside the @Benchmark annotation to run performance tests. The JMH framework provided a reliable way to measure the execution time of matrix multiplication operations with high accuracy.

4.3 C

For C implementations, the ‘clock()‘ function from the ‘time.h‘ library was used to manually calculate the execution time as well as minimum time, maximum time, mean and median. This approach was chosen due to compatibility issues of existing benchmarks with MacOS, and ensures accurate timing measurements for matrix multiplication operations in the C language.

5 Experiments

5.1 Python

5.1.1 Introduction

Python is a high-level, interpreted programming language known for its simplicity and readability, making it a popular choice for beginners and experienced developers as well. Advantages of Python include its clean and concise syntax,, broad community support and cross-platform compatibility. In addition, Python is suitable for a wide range of applications, including web development, data analysis and machine learning.

However Python has its down sides too. Foremost these include slower performance compared to languages such as C or Java. Despite this limitation, Python's widespread adoption and strong community make it a powerful and flexible language for various programming tasks.

5.1.2 Execution time

As shown in a *Table 1*, Python works well and fast for smaller sizes, but in realistic cases we would need matrices larger than 10 or 50. The time taken to compute matrices of size 200 was very long and does not look practical, as we would normally probably use matrices even larger than 200. However, we must remember that Python is a great language for such calculations because of its libraries, which were not used in this case.

Size	**Min Time (ms)**	**Max Time (ms)**	**Mean Time (ms)**	**Median Time (ms)**
5	5.1347	5.3196	5.2271	5.2271
10	32.5461	32.7599	32.6034	32.5831
50	3830	3856.2	3843.1	3843.1
100	28895.9	30521.3	29708.6	29708.6
150	90753.5	93360.7	92057.1	92057.1
200	235638.4	240116.1	237977.3	237877.3

Table 1: Benchmarking results for matrix multiplication in Python

5.1.3 Chart

Below we can see the execution times displayed as a graph. The difference between the smallest and largest matrices is huge - almost 250 000 ms, which is 250 s or over 4 minutes.

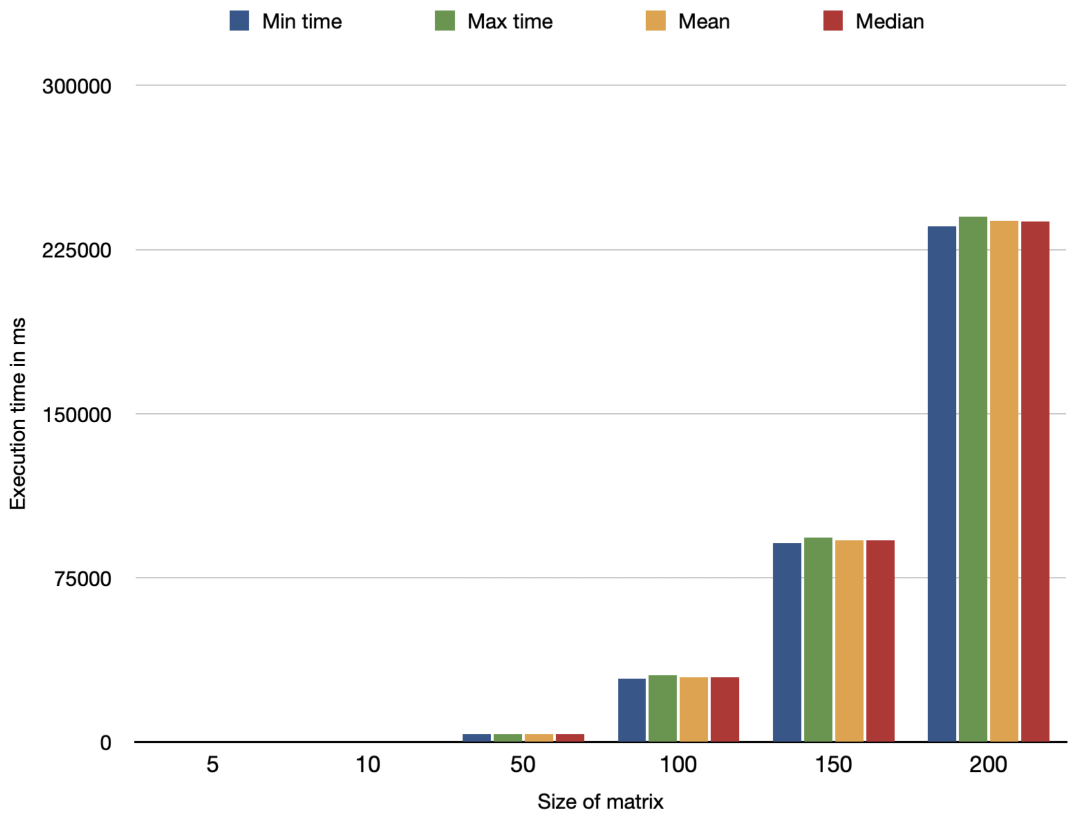

Figure 1: Visualized results for matrix multiplication in Python

5.2 Java

5.2.1 Introduction

Java is a powerful object-oriented programming language known for its platform independence, stability and security features. One of its biggest advantages is the "write once, run anywhere" capability provided by the Java Virtual Machine (JVM), which enables programs written in Java to run on any system that has a JVM installed. Java's extensive standard library, strong community support and scalability make it an excellent choice for building enterprise applications, mobile applications, web services and more.

But Java does have some drawbacks too. For instance, more verbose syntax compared Python and lower performance in certain use cases. In addition, Java's memory consumption can be a challenge, especially for beginners. However, Java's vast ecosystem of tools and frameworks, as well as its long-term compatibility and stability, continue to make it a popular choice for a wide range of software development projects.

5.2.2 Execution time

As it is shown in a *Table 2*, we can already see that Java's results are much better than Python's. The average time to compute the multiplication of matrices of size 200 is 5ms, which is the time it took Python to compute the multiplication of matrices of size 5. Java coped very well with the calculations, but it can be seen that the maximum operation time is much longer than the average, even twice as long.

Size	**Min Time (ms)**	**Max Time (ms)**	**Mean Time (ms)**	**Median Time (ms)**
5	$\approx 10^{-4}$	0.050	$\approx 10^{-4}$	$\approx 10^{-4}$
10	$\approx 10^{-3}$	0.077	0.001	0.001
50	0.058	0.230	0.063	0.063
100	0.493	5.104	0.539	0.530
150	1.808	5.988	1.942	1.931
200	4.817	13.727	5.074	5.014

Table 2: Benchmarking results for matrix multiplication in Java

5.2.3 Chart

The graph also shows that the maximum time stands out from the other measurements. Even with Java's very good results, we must remember that there are worse cases and the time difference is quite large.

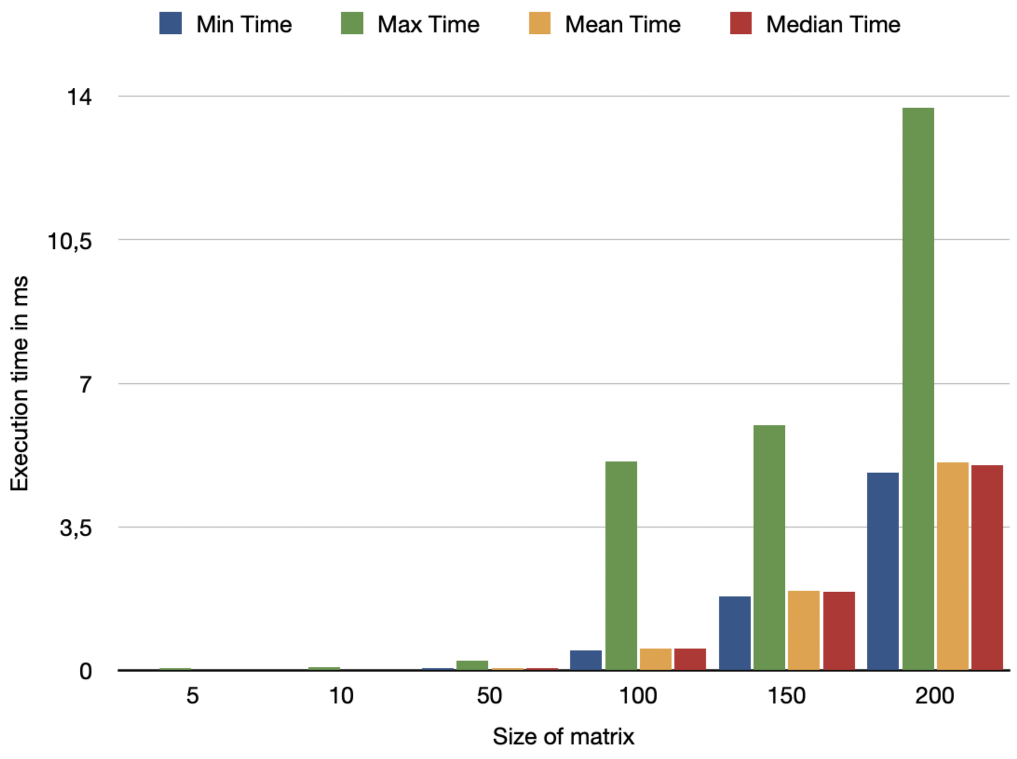

Figure 2: Visualized results for matrix multiplication in Java

5.3 C

5.3.1 Introduction

C is a powerful and widely used procedural programming language known for its speed, versatility and efficiency. One of its main advantages is its high performance, making it ideal for systems programming. C's direct access to memory and extensive libraries allow developers to write efficient and fast code.

On the other hand, C can be considered a more complex and difficult language to learn than higher-level languages like Python or Java, due to its manual memory management and pointer arithmetic. Also, C lacks built-in support for object-oriented programming, which can make it less suitable for certain types of applications. However C remains a fundamental language in computer science and is often the language of choice for performance-critical tasks, operating system development, and hardware-related programming.

5.3.2 Execution time

The results obtained for the C language are really good. The values for larger matrices are worse than for Java, but here we do not have the large gap between the best, average and worst case. Perhaps C is not the winner, although it is a very strong contender, with satisfactory results for all matrix sizes.

Size	**Min Time (ms)**	**Max Time (ms)**	**Mean Time (ms)**	**Median Time (ms)**
5	0.00	0.01	0.00	0
10	0.01	0.01	0.01	0.01
50	0.30	0.68	0..40	0.38
100	1.67	1.89	1.72	1.71
150	5.33	5.81	5.63	5.70
200	12.79	13.71	13.54	13.60

Table 3: Benchmarking results for matrix multiplication in C

5.3.3 Chart

The graph also shows that all measurements are at a similar level for each size. The amplitude is about 14 ms, which is about 17,857 times faster than Python (250,000 ms).

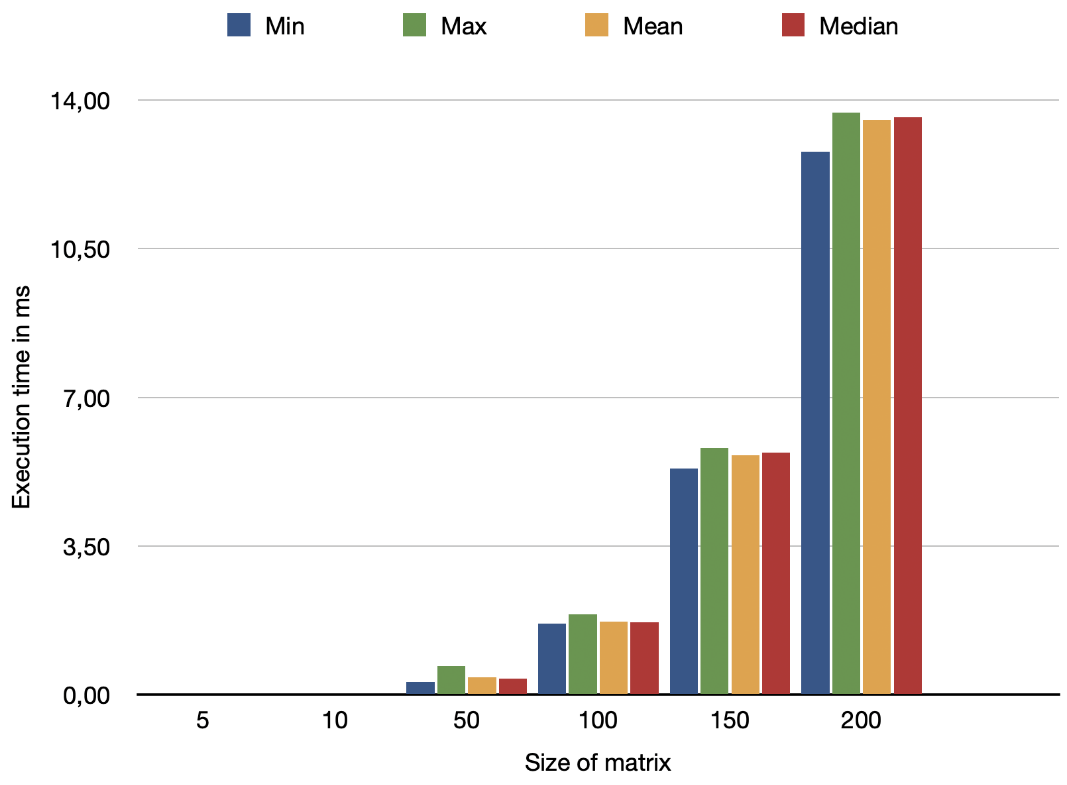

Figure 3: Visualized results for matrix multiplication in C

6 Conclusions

6.0.1 Charts

As highlighted in the chart below, it is a bit pointless to look at the performance of Java or C, as Python's values are too large to even compare with the others. For people who need to compute huge matrices full of data, Python will not be the first choice, at least not in the most basic implementation. In reality, Python can be optimised very well and is widely used.

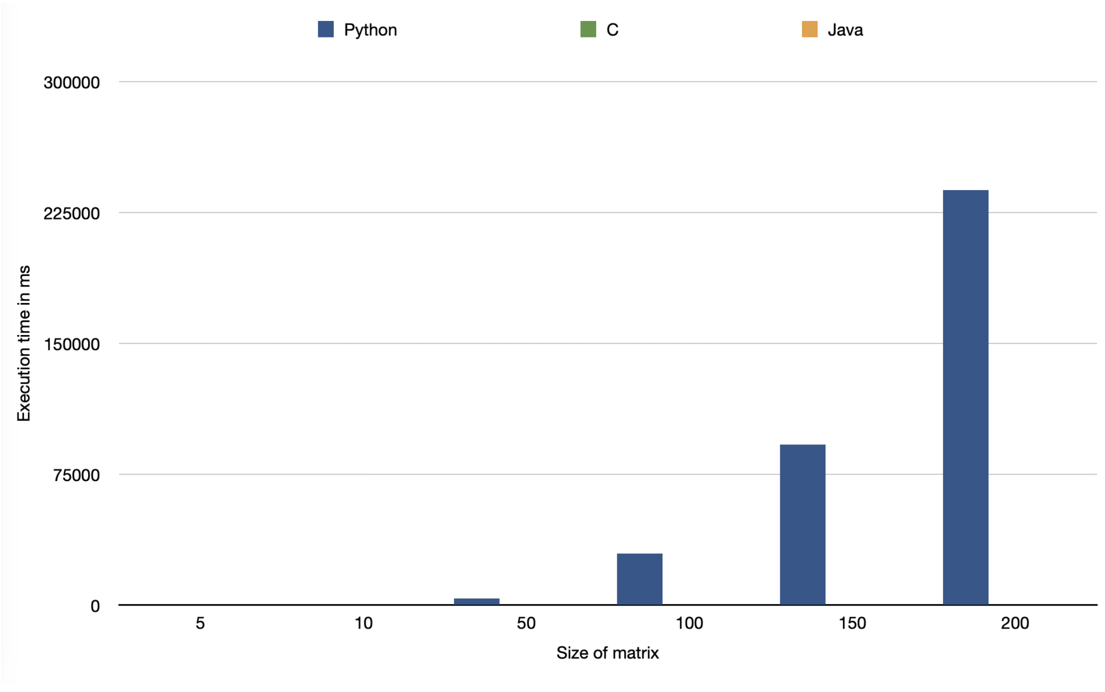

Figure 4: Average time comparison for Python, Java and C

Let me analyse the computation time in Java and C without Python, as here the difference can be seen. Java did a better job, obtaining a shorter time, with the difference increasing with the size of the matrix.

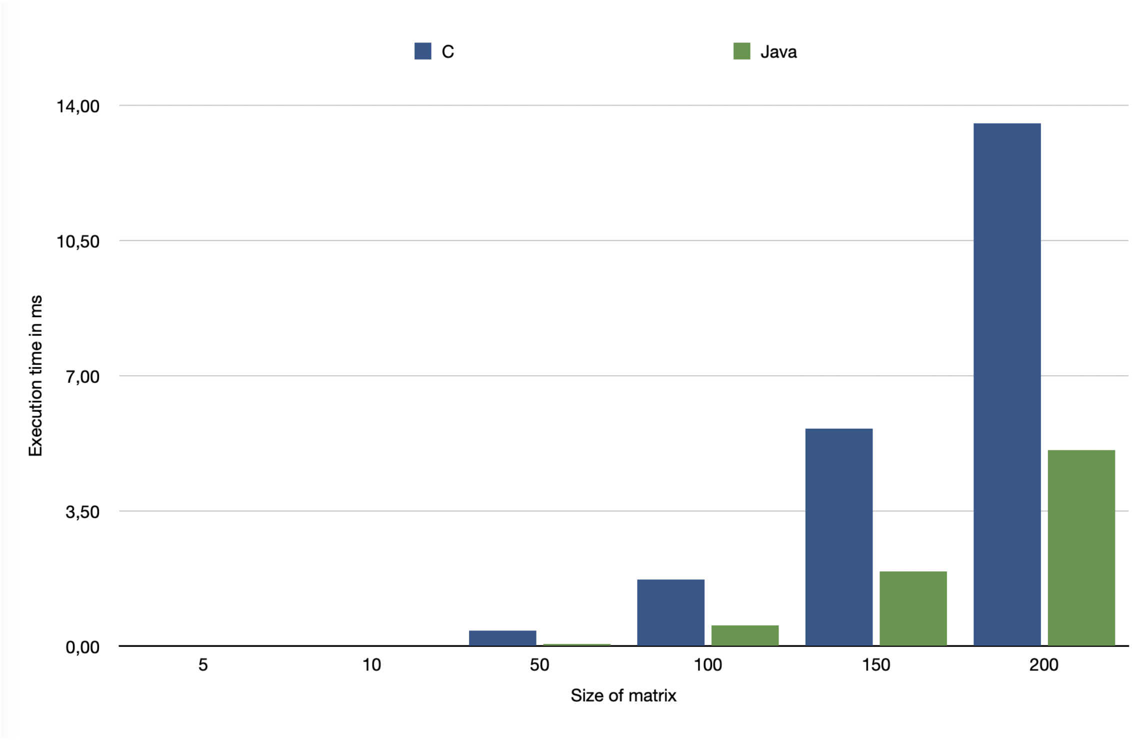

Figure 5: Average time comparison for Java and C

6.0.2 Commentary

After analysing the benchmark results between Python, Java, and C for matrix multiplication, it is clear that each language has its unique strengths and weaknesses. Python stands out for its simplicity and versatility, making it an excellent choice for beginners or applications with less data to analyse. Java's satisfactory results connected to platform independence and vast ecosystem make it suitable for bigger, enterprise-level. On the other hand, C's high performance and efficiency make it ideal for system programming and low-level tasks.

Choosing the right language for matrix operations depends on project requirements, performance considerations, and development complexity, as well as available time and software engineers. Python offers a balance between ease of use and functionality, while Java brings stability and scalability. Meanwhile, C remains the best choice for performance-critical applications. Understanding the strengths and limitations of these languages is crucial for selecting the most appropriate tool for need of you potential system.

7 Future work

There are several promising ways to explore for optimising matrix multiplication operations in Python, Java and C. One of them is to investigate parallel processing such as multi-threading to better exploit the capabilities of multi-core systems. Furthermore, exploring advanced optimisation algorithms such as loop unrolling and cache optimisation could significantly improve computational efficiency.

Finally, the integration of GPUs, offers an opportunity to boost performance. By using these strategies, future iterations of matrix multiplication algorithms could provide better and very promising results.