Documentation
Language Benchmark of matrix multiplication

Olga Kalisiak 00325481

October 12, 2024

1 Abstract

Matrix multiplication plays a crucial role in areas like scientific computing, machine learning, and data processing. The speed at which this operation is performed can vary widely depending on the programming language chosen.

This study compares how three popular programming languages—Java, Python, and C—handle matrix multiplication. Standard matrix multiplication was implemented in each language. It was tested by changing the matrix sizes. The program execution times were measured using a benchmark.

The findings revealed that C consistently executed matrix multiplication faster than both Java and Python. Python has the slowest performance due to its high level of abstraction and dynamic typing, while Java has moderate processing speed.

This study emphasizes how the choice of programming language can impact performance in computation-intensive tasks, particularly in applications where speed is essential.

2 Introduction

Matrix multiplication plays a key role in processing and analyzing data. It is widely used in fields like computer graphics, machine learning, and scientific simulations. The language chosen for performing this algorithm has a major impact on execution time. This project compares matrix multiplication across three popular languages: Python, Java, and C. By tracking execution time, memory use, and CPU load, we assess the computational efficiency of each language for handling large-scale matrix operations.

The project's main contribution is a detailed comparison of how each language performs in matrix multiplication.

3 Problem statement

Matrix multiplication is compute-intensive, and the performance can vary significantly depending on the programming language used due to differences in memory management, garbage collection, and compiled vs. interpreted execution. The problem is to determine which language among Python, Java, and C performs best for large matrix multiplications, including also memory and CPU usage.

4 Methodology

I tested each language's matrix multiplication benchmarks. The machine used for the research had the following setup:

- Laptop model: Acer Nitro 5
- Processor: AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz
- Installed RAM: 16.0 GB (available: 15.4 GB)
- Product ID: 00328-00801-21601-AA448
- System Type: 64-bit operating system, x64 processor
- Operating system: Windows 11 and Ubuntu-24_04-lts

Languages and compilers:

- Python (version 3.10.11, PyCharm 2024.2.3)
- Java (version 20.0.1, IntelliJ IDEA Community Edition 2024.2.2)
- C (version C11, CLion 2024.2.2)

To see how performance scales, two big matrices of different sizes are multiplied in each experiment. Execution time is measured using a stopwatch function in each language. Memory usage and CPU utilization are tracked with appropriate profiling tools in each language:

- Python: time and psutil libraries
- Java: ThreadMXBean from java.lang.management package and Runtime memory methods
- C: time.h for timing and psapi.h (with PROCESS_MEMORY_COUNTERS) for memory usage

Three important indicators are CPU utilization (measured in milliseconds), memory use (measured in bytes), and average runtime (measured in milliseconds).

4.1 Experiment Setup

4.1.1 Programming Languages and Libraries

To test performance, I conducted experiments in C on Windows, and I used Ubuntu to use the perf tool for profiling

- Python: Implemented with the NumPy library and benchmarked using pytest in PyCharm.
- Java: Implemented with the Java standard library, benchmarked using JMH in IntelliJ
- C (Windows): Implemented with the standard C library, compiled in CLion.
- C (Ubuntu): Implemented with the standard C library, compiled with gcc in CLion, and profiled using perf

4.2 Performance Metrics

Three indicators are measured in order to guarantee thorough performance evaluation:

- Execution Time: Runtime for multiplications is measured using benchmarking tools.
- Memory Usage: Memory usage is tracked to evaluate how well each language uses the memory resources that are available.
- CPU Utilization: CPU usage is monitored to observe how effectively each language uses available computational resources.

4.3 Experimental Procedure

Matrices of sizes 8x8, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, 2048x2048 and 4096x4096 are used to assess how each language handles increased computational load.

5 Experiments

Codes from these experiments are available in the repository: Github repository

5.1 Execution time

The results of the execution time experiment are presented in the Table 1 and in the chart Figure 1 below.

Performance Time [ms]				
Size	**Python**	**Java**	**C (Windows)**	**C (Ubuntu)**
8x8	0.2549352	0.001	0.002	0.2
32x32	22.8455	0.023	0.055	0.32
64x64	180.8903	0.192	0.806	1
128x128	1159.7357	1.633	6.277	5.45
256x256	6862.8	19.112	41.069	44.39
512x512	54443.6	179.193	417.417	401.28
1024x1024	471033.4	2990.323	4196.876	3070.39
2048x2048	-	64921.41	38861.719	32508.04
4096x4096	-	-	327696.081	305771.39

Table 1: Performance Comparison for Different Sizes in Python, Java, C (Windows), and C (Ubuntu)

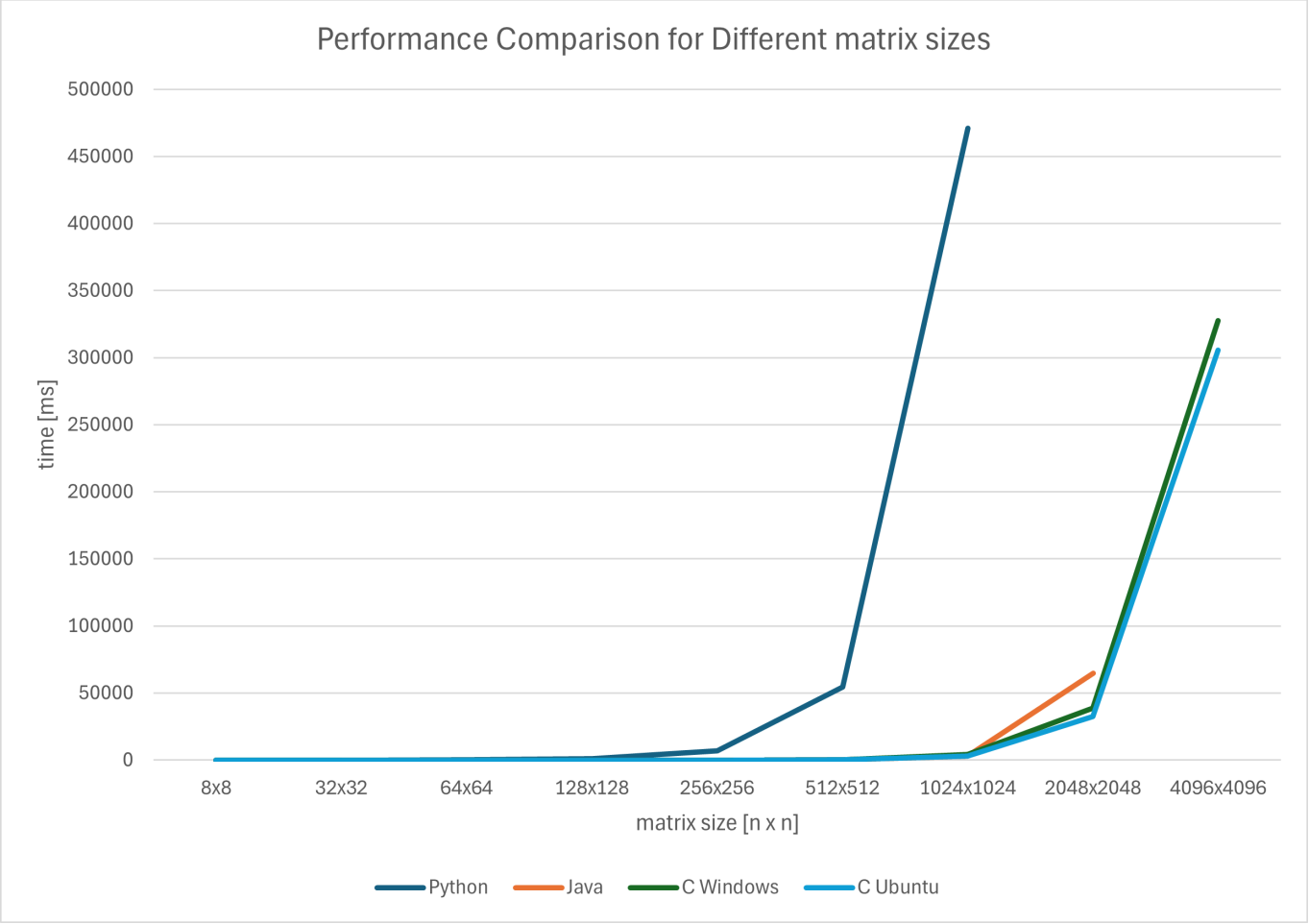

Figure 1: Performance Comparison for Different Sizes in Python, Java, C (Windows) and C (Ubuntu)

Execution time, recorded in seconds, reveals clear performance differences:

- Python: The Python implementation is significantly slower than others, especially as matrix size increases. It was to slow to perform operations on a matrix of size 2048x2048 and 4096x4096. This is because, in contrast to compiled languages, Python is an interpreted language that relies on slower looping structures.
- Java: Although it performs slower than C on large matrices, Java scales effectively up to large sizes and has competitive performance for smaller matrices. It was to slow to perform operations on a matrix of size 2048x2048.
- C on Windows: The C implementation on Windows performs well but lags slightly behind C on Ubuntu for larger matrices.
- C on Ubuntu: The C implementation on Ubuntu is the fastest for the largest matrix sizes, especially at 4096x4096. The use of perf on Ubuntu helps achieve optimized memory and CPU usage, making it highly efficient for large computations.

5.2 Memory usage

The results of the memory usage experiment are presented in the Table 2 and in the chart Figure 2 below.

Memory Usage [bytes]			
Size	**Python**	**Java**	**C (Windows)**
8x8	50000	6.2351	60000
32x32	50000	28.15	60000
64x64	170000	204.13	60000
128x128	600000	1042.04	60000
256x256	2390000	23406.8	60000
512x512	10090000	428906.8	60000
1024x1024	40320000	2747499	60000
2048x2048	-	30486166.17	48000
4096x4096	-	-	56000

Table 2: Memory usage Comparison for Different Sizes in Python, Java, and C (Windows)

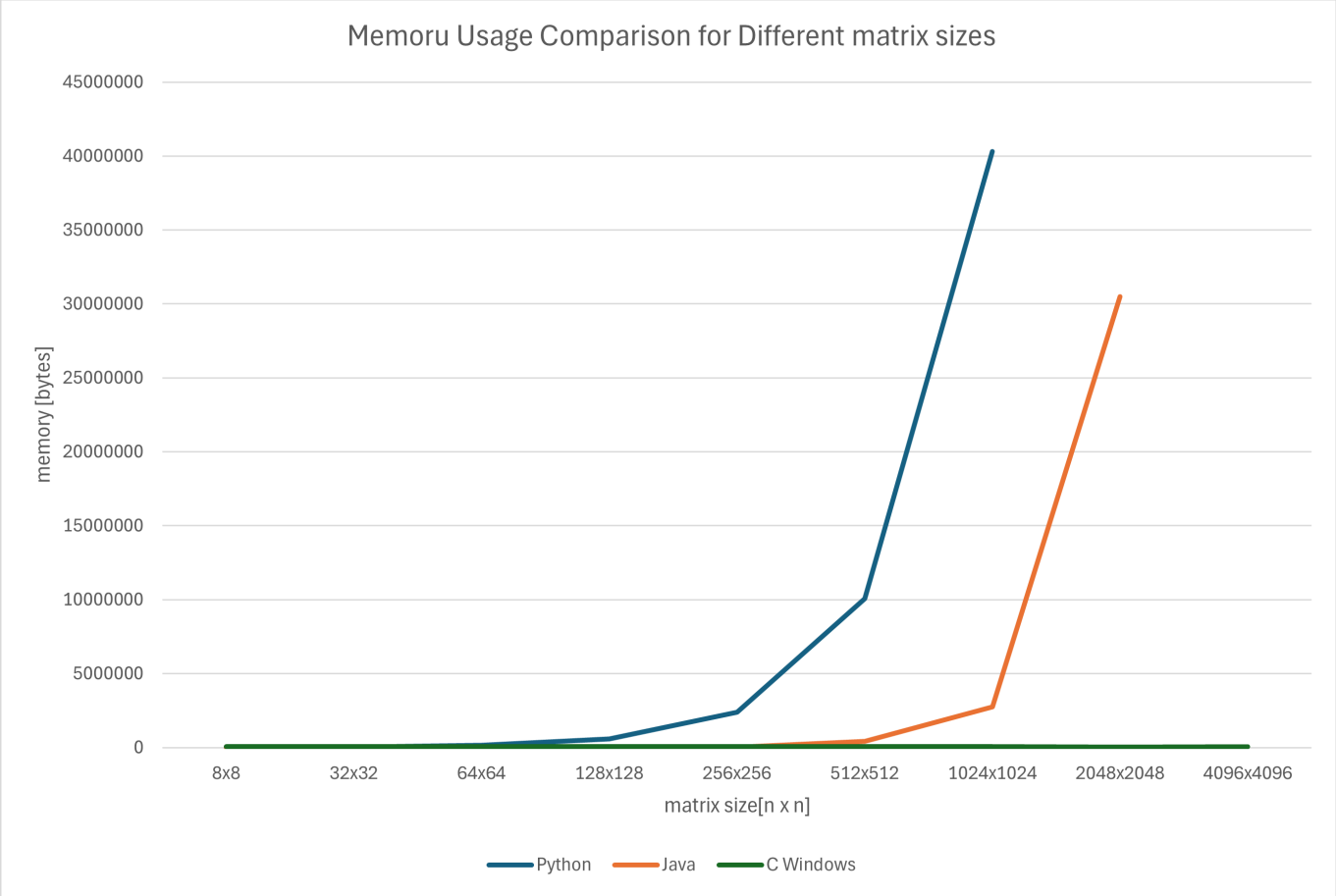

Figure 2: Memory usage Comparison for Different Sizes in Python, Java, and C (Windows)

Memory usage results highlight language-specific efficiency:

- Python: Memory usage in Python scales sharply with matrix size, particularly after 256x256, where memory usage becomes very high. This is likely due to Python's memory management and lack of low-level optimization for handling large data structures.
- Java: Java shows relatively efficient memory usage for smaller matrices, but the memory consumption increases significantly from 512x512 upwards. This increase reflects Java's handling of object overhead and garbage collection but remains better optimized than Python for larger sizes.
- C: Memory usage in C on Windows is stable and efficient across all matrix sizes, consistently using less memory than both Python and Java. This stability is likely due to C's low-level memory management and lack of overhead from runtime systems or garbage collection.

5.3 CPU Utilization

The results of the CPU Utilization experiment are presented in the Table 3 and in the chart Figure 3 below.

CPU Time [ms]			
Size	**Python**	**Java**	**C (Windows)**
8x8	1046.88	0.001016	0
32x32	1140.62	0.00957	0
64x64	1203.12	0.07259	15
128x128	2500	0.7008	15
256x256	15765.62	7.6729	31
512x512	109468.75	70.606	468
1024x1024	613843.75	2568.88	2718
2048x2048	-	25966.8	23140
4096x4096	-	-	190531

Table 3: CPU Time Usage for Different Sizes in Python, Java, and C (Windows)

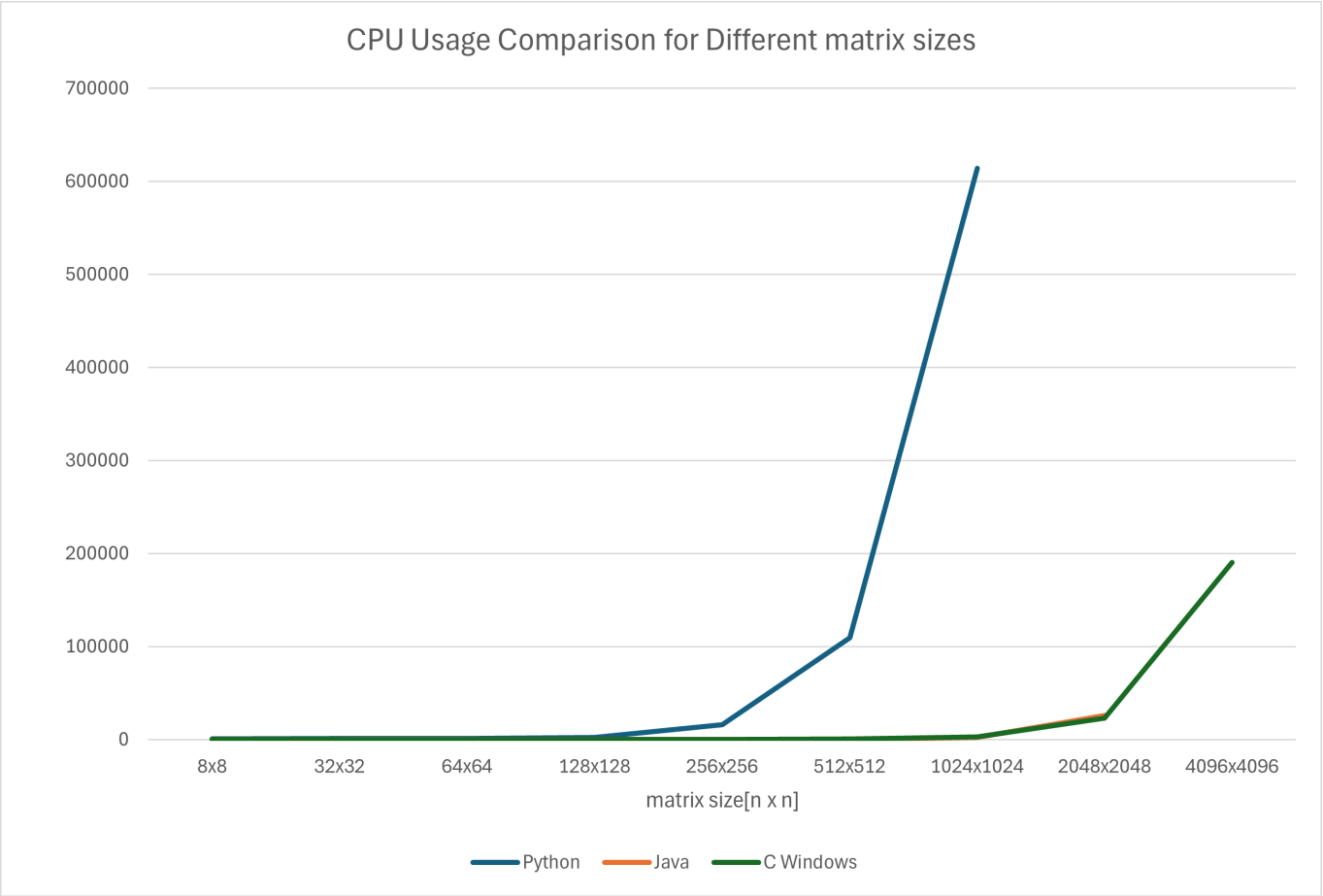

Figure 3: CPU Usage Comparison for Different Sizes in Python, Java, and C (Windows)

- Python: The increase in CPU time is steep, especially for larger matrices, highlighting Python's performance limitations with computationally intensive tasks
- Java: Java demonstrates a more efficient CPU usage pattern than Python, especially for smaller matrices. It's almost the same as in C.
- C: Highest CPU utilization, demonstrating effective performance.

6 Conclusion

This project highlights the performance differences in matrix multiplication across Python, Java, and C, focusing on execution time, memory usage, and CPU efficiency. The C solution proved to be the most effective for large-scale matrix operations, outperforming Python and Java. Java offered balanced performance, while Python, although being the slowest, stands out for its usability.

7 Future work

Future experimentation will explore the following topics:

- Optimized Matrix Multiplication Approaches
- Sparse Matrices
- Vectorized and Parallel Matrix Multiplication
- Distributed Matrix Multiplication