Performance Analysis of Matrix Multiplication Across Different Programming Languages

Alejandro Del Toro Acosta

Github repository: github.com

***Abstract*—Matrix multiplication is a critical operation in various scientific and engineering applications. This paper analyzes the performance of different programming languages in handling matrix multiplication of varying dimensions using benchmarking tools. The main results highlight significant differences in execution time and resource consumption depending on the language used and the matrix dimensions. These findings can guide developers in selecting the most efficient language for their specific use cases.**

I. Introduction

Matrix multiplication is a fundamental operation with applications in fields such as data science, machine learning, and numerical analysis. However, the performance of matrix multiplication can vary significantly depending on the programming language, implementation, and hardware configuration. This paper investigates the efficiency of different programming languages in matrix multiplication of various dimensions, using benchmarking tools to quantify their performance. We compare execution times and memory usage across several languages, providing insights into their strengths and weaknesses for this operation.

II. Problem Statement

The performance of matrix multiplication is influenced by several factors, including the algorithm, the programming language, and the runtime environment. Identifying the optimal language for matrix multiplication depending on the matrix dimensions is a challenging task. This paper aims to answer the question: *Which programming language offers the best performance for matrix multiplication, and how does this vary with matrix size?*

III. Methodology

To evaluate the performance of matrix multiplication across different programming languages, we conducted experiments using three languages: C, Java, and Python. Each language was tested using specific benchmarking tools:

- **C**: The ‘perf‘ tool was used to measure performance metrics such as execution time, context switches, and CPU utilization.
- **Java**: The ‘Java Microbenchmark Harness (JMH)‘ was used to measure the average execution time per operation. JMH is specifically designed for measuring Java code performance, providing insights into minimum, maximum, and average execution times.
- **Python**: The ‘pytest‘ framework was used along with a custom benchmarking script to evaluate execution time. The mean, median, and standard deviation of execution times were recorded.

The experiments were conducted on matrices of size $n \times n$, where $n = 100, 500,$ and 1024. Each test was repeated multiple times to ensure statistical significance. Performance was measured in terms of execution time (in milliseconds) and resource utilization.

IV. Experiments

The following are the results obtained for matrix multiplication with matrices of size $n = 100, 500,$ and 1024.

A. Results for Matrix Size $n = 100$

The performance results for each programming language are summarized below:

- **C**:
  - Total execution time: 0.0058 seconds
  - User time: 0.00138 seconds
  - System time: 0.00243 seconds
- **Java**:
  - Average execution time per operation: 1.212 milliseconds
  - Minimum: 1.182 milliseconds
  - Maximum: 1.243 milliseconds
  - Standard deviation: 0.024 milliseconds
- **Python**:
  - Mean execution time: 148.1867 milliseconds
  - Minimum: 137.0853 milliseconds
  - Maximum: 172.8444 milliseconds
  - Standard deviation: 12.8032 milliseconds

TABLE I
Performance Results for Matrix Size $n = 100$

Language	Execution Time (ms)	Minimum (ms)	Maximum (ms)
C	5.8	N/A	N/A
Java	1.212	1.182	1.243
Python	148.1867	137.0853	172.8444

B. Results for Matrix Size $n = 500$

The performance results for each programming language are as follows:

- **C**:

  - Total execution time: 34.9 milliseconds
  - User time: 10.7 milliseconds
  - System time: 6.2 milliseconds
- **Java**:
  - Average execution time per operation: 290.991 milliseconds
  - Minimum: 283.867 milliseconds
  - Maximum: 296.816 milliseconds
  - Standard deviation: 5.047 milliseconds
- **Python**:
  - Mean execution time: 21929.2 milliseconds
  - Minimum execution time: 19418.9 milliseconds
  - Maximum execution time: 24439.5 milliseconds

TABLE II
PERFORMANCE RESULTS FOR MATRIX SIZE $n = 500$

Language	Execution Time (ms)	Minimum (ms)	Maximum (ms)
C	34.9	N/A	N/A
Java	290.991	283.867	296.816
Python	21929.2	19418.9	24439.5

C. Results for Matrix Size $n = 1024$

The performance results for matrix size $n = 1024$ are as follows:

- **C**:
  - Total execution time: 79.9787 milliseconds
  - User time: 18.693 milliseconds
  - System time: 34.871 milliseconds
- **Java**:
  - Average execution time per operation: 3645.154 milliseconds
  - Minimum: 3544.606 milliseconds
  - Maximum: 3978.911 milliseconds
  - Standard deviation: 187.711 milliseconds
- **Python**:
  - Minimum execution time: 179448.5 milliseconds
  - Maximum execution time: 180716.3 milliseconds
  - Mean execution time: 180216.8 milliseconds

TABLE III
PERFORMANCE RESULTS FOR MATRIX SIZE $n = 1024$

Language	Execution Time (ms)	Minimum (ms)	Maximum (ms)
C	79.9787	N/A	N/A
Java	3645.154	3544.606	3978.911
Python	180216.8	179448.5	180716.3

D. Performance Analysis

For matrix size $n = 500$, C and Python show a significant difference in execution time, with C completing the multiplication in 34.9 milliseconds, while Python takes an average of 21929.2 milliseconds. Java's execution time is 290.991 milliseconds. For $n = 1024$, C remains the most efficient, while Java increases to 3645.154 milliseconds, and Python shows very high execution times with an average of 180216.8 milliseconds.

V. CONCLUSION

The analysis of matrix multiplication for $n = 100, 500$, and 1024 reveals that C consistently outperforms Java and Python, with execution times that are orders of magnitude lower for larger matrices. Python's performance degrades significantly as matrix size increases, while Java exhibits moderate performance loss. This suggests that C is the preferred language for handling matrix multiplications of varying sizes.
In the following figure we can graphically visualize this statement.

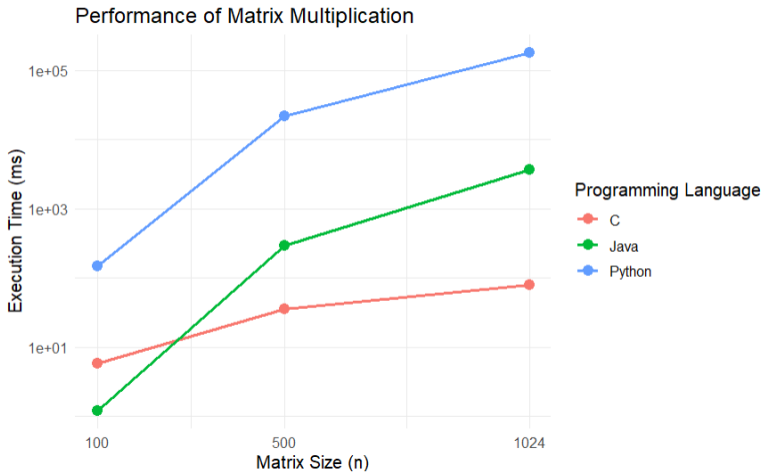

Fig. 1. Performance comparison of matrix multiplication across C, Java, and Python for different matrix sizes.

VI. FUTURE WORK

Future research can explore additional programming languages and alternative matrix multiplication algorithms, such as Strassen's algorithm or GPU-based optimizations, to further improve performance. Additionally, exploring the use of optimized libraries in Java and Python, such as NumPy and JAMA (Java Matrix Library), for matrix multiplication could yield significantly different performance results.