Performance Analysis of Matrix Multiplication across Programming Languages

Gabriel Munteanu Cabrera

October 22, 2025

Abstract

This paper presents a comparative performance analysis of the standard matrix multiplication algorithm implemented in C, Java, and Python. The experiments were conducted across various matrix sizes to evaluate the impact of different programming paradigms and execution environments—compiled, Just-In-Time (JIT) compiled, and interpreted—on computational efficiency. The results demonstrate that compiled languages like C offer the best performance for smaller workloads, while Java's JIT compiler shows remarkable optimization capabilities for larger matrices. Python, as an interpreted language, consistently exhibits the highest execution time, highlighting the performance trade-offs for its high-level abstractions.

1 Introduction

Matrix multiplication is a fundamental operation in linear algebra and a core component of countless algorithms in science, engineering, and data analysis. Its computational complexity, typically $O(n^3)$, makes its performance a critical factor in application efficiency. The choice of programming language can significantly influence this performance due to differences in their execution models, memory management, and optimization capabilities.

This study aims to quantify these performance differences by implementing and benchmarking a standard matrix multiplication algorithm in three distinct languages: C, representing low-level compiled languages; Java, representing JIT-compiled languages running on a virtual machine; and Python, representing high-level interpreted languages.

2 Methodology

To ensure a fair and reproducible comparison, a consistent methodology was applied across all three languages. This section details the implementation, verification, and experimental setup.

2.1 Implementation in C

The C implementation was developed to serve as a performance baseline. The core logic (ijk variant) was isolated in a `matmul_ijk` function operating on one-dimensional arrays. The benchmarking driver handled command-line arguments (`--size`, `--repeats`), dynamic memory allocation with `malloc`, and time measurement using the high-precision `gettimeofday` function. The code was compiled with GCC using the `-O2` optimization flag.

2.2 Implementation in Java

The Java implementation followed an object-oriented approach. The multiplication algorithm was encapsulated in a static method, `MatrixOps.matmul`. The main driver class, `MatrixCLI`, managed experiment parameterization and used `System.nanoTime()` for high-resolution timing. The code was executed on a standard JDK, relying on the Just-In-Time (JIT) compiler for runtime optimizations.

2.3 Implementation in Python

The Python version was structured with the `matmul` function in its own module. The main script utilized standard libraries: `argparse` for command-line arguments, `time.perf_counter()` for accurate timing, and the `csv` module for data collection. As an interpreted language, this implementation serves as a baseline for dynamically-typed languages.

2.4 Code Verification

Before benchmarking, the correctness of each implementation was verified through a suite of unit tests. These tests included validation against known results for a 2x2 matrix and checks for mathematical properties, such as multiplication by the identity and zero matrices. This ensures that the performance measurements are valid and reliable.

3 Results

The experiments were executed for matrix sizes of 64x64, 128x128, 256x256, and 512x512, with each test being repeated 3 times to obtain an average. The collected data is summarized in Table 1 and visualized in Figure 1.

Table 1: Average execution time (in seconds) for matrix multiplication.

Matrix Size	C	Java	Python
64x64	0.000528	0.002857	0.051600
128x128	0.002952	0.026322	0.367444
256x256	0.033309	0.038695	3.439129
512x512	0.365053	0.330681	29.999540

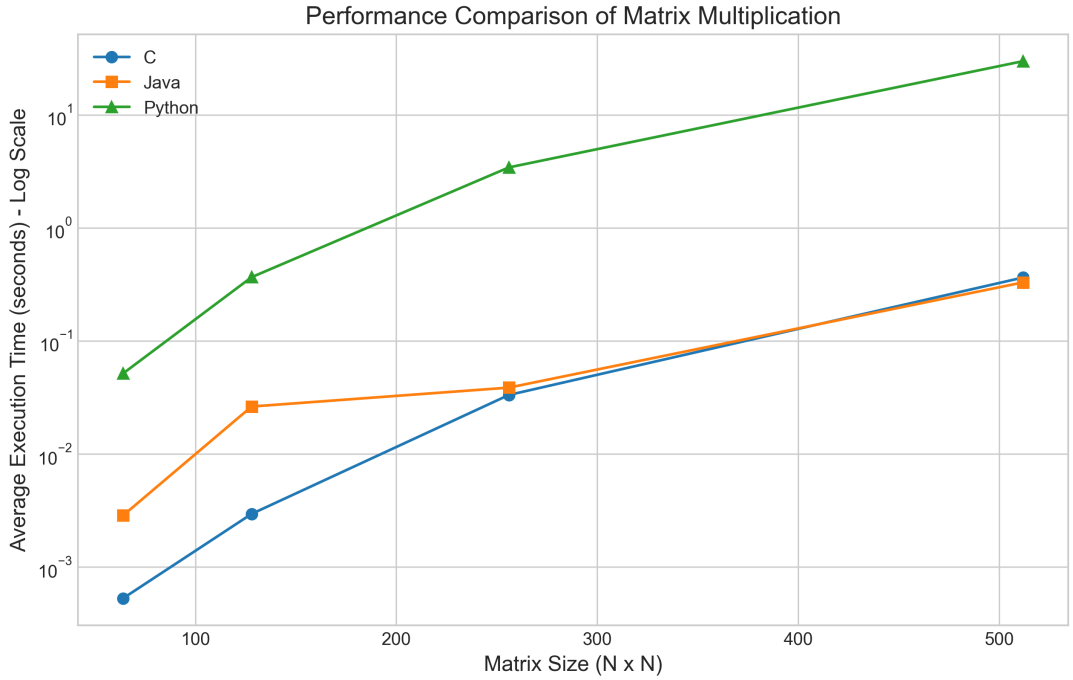

Figure 1: Execution time vs. Matrix Size for C, Java, and Python. Note the logarithmic scale on the y-axis.

4 Discussion

The results align with theoretical expectations but also reveal interesting nuances. Python's performance is several orders of magnitude slower than C and Java, which is attributable to its interpreted nature.

The most notable result is Java's performance surpassing C's for the 512x512 matrix. This is a classic demonstration of the power of the JVM's Just-In-Time (JIT) compiler. During execution, the JIT can perform profile-guided optimizations on "hot" code paths, sometimes achieving a level of optimization that surpasses the static, ahead-of-time compilation of C.

Finally, the execution times grow consistently with the $O(n^3)$ complexity of the algorithm, which is particularly visible in the steep curve of the performance plot.

5 Conclusion

This study successfully benchmarked matrix multiplication in C, Java, and Python. The findings confirm that for raw computational performance, compiled languages like C remain a top choice. However, modern execution environments like the JVM in Java can provide competitive, and sometimes superior, performance for long-running, intensive tasks due to advanced runtime optimizations. Python, while significantly slower, offers development speed and simplicity, representing a clear trade-off between programmer productivity and computational efficiency.

A Repository Link

The complete source code, test suites, and raw data for this project are publicly available on GitHub: github.com/Gabrii8/matrix-assignment.