Comparison of Matrix Multiplication Performance Across Java, Python, and C

Javier González Benítez
Benchmark's GitHub repository.

Abstract

This paper presents a comparative analysis of the performance of matrix multiplication across three programming languages: Java, Python, and C. The study benchmarks the execution times and resource utilization of each language to determine which offers the most efficient performance under similar conditions.

1 Introduction

Matrix multiplication is a fundamental operation in many computational tasks. Its performance can vary significantly depending on the programming language and the underlying implementation. In this paper, we evaluate the execution time and system resource usage during matrix multiplication using three different languages: Java, Python, and C.

2 Methodology

The tests were conducted on the same hardware to ensure consistency. The matrix multiplication was implemented in Java, Python, and C, and benchmarked using appropriate tools for each language.

2.1 Java

For Java, the matrix multiplication was benchmarked using JMH (Java Microbenchmark Harness). The score represents the average time taken per operation (ms/op) across different matrix sizes.

Benchmark	Size (n)	Mode	Count	Score (ms/op)	Error (ms/op)	Units
testMethod	10	avgt	5	0.001	0.001	ms/op
testMethod	100	avgt	5	0.621	0.361	ms/op
testMethod	1000	avgt	5	1812.668	446.505	ms/op

Table 1: Java Benchmark Results

2.2 Python

In Python, matrix multiplication was benchmarked using the `timeit` module. The table below shows the minimum, maximum, mean, standard deviation, and the median.

Name	Min (μs)	Max (μs)	Mean (μs)	Std Dev (μs)	Median (μs)
test_my_function	72.800	79.200	74.530	1.7951	74.100

Table 2: Python Benchmark Results

2.3 C

For C, the `perf` tool was used to measure CPU cycles, context switches, page faults, task-clock time, and CPU core value. The table below summarizes the results.

Metric	Value	Unit
Task-Clock	98.45	msec
Context-Switches	11	/sec
CPU Migrations	0	/sec
Page Faults	6.206	K/sec
CPU Utilization	0.804	CPUs
Time Elapsed	0.122	sec
User Time	0.045	sec
System Time	0.049	sec

Table 3: C Benchmark Results

3 Results and Discussion

3.1 Performance Comparison

When comparing execution times across all three languages, the following trends emerge:

- **Java**: The average execution time for small matrices (n = 10) was competitive at 0.001 ms/op. However, for larger matrices (n = 1000), the execution time increased significantly to 1812.668 ms/op, indicating a steep performance drop as the matrix size grows.
- **Python**: For small matrices (n = 10), Python's execution time was faster in absolute terms (median of 74.1 s), but the standard deviation indicates greater variability compared to Java. As matrix sizes increase, Python is expected to show performance degradation due to its interpreted nature and reliance on high-level abstractions.

- **C**: C demonstrated the best overall performance. The total elapsed time of 0.122 seconds, combined with low CPU usage and minimal overhead, highlights C's superiority in handling computationally intensive tasks like matrix multiplication. Its execution time scales much more efficiently with increasing matrix sizes.

3.2 Resource Utilization

The resource utilization measurements provide further insight into the performance differences:

- **Java**: The JVM introduces additional resource overhead due to memory management and garbage collection, contributing to the increased execution time for large matrix sizes.
- **Python**: The GIL and Python's interpreted runtime add overhead, particularly when dealing with larger matrices. Although the use of NumPy mitigates some of these effects, Python remains slower compared to C for larger datasets.
- **C**: C's direct interaction with hardware, coupled with its lack of runtime overhead, ensures minimal resource usage. The low context-switches, page-faults, and CPU utilization underscore its efficiency.

4 Conclusion

In conclusion, the benchmarking results clearly show that C outperforms Java and Python in terms of both execution time and resource utilization for matrix multiplication tasks. Java, though structured and easier to use for large-scale applications, exhibits significant performance drops as the matrix size grows. Python, while fast for small matrices, is hampered by runtime overhead and variability. C, with its direct-to-hardware execution and minimal overhead, remains the most efficient language for computationally intensive tasks like matrix multiplication.