Comparative Analysis of Matrix Multiplication Implementations in Python, Java, and C

Gabriel Felipe Bernal Pinto

October 20, 2024

Abstract

Matrix multiplication is a fundamental operation in various scientific and engineering applications. This paper presents a comparative analysis of matrix multiplication implementations in Python, Java, and C. Benchmarks are conducted to evaluate the performance in terms of execution time and memory usage for different matrix sizes. The results provide insights into the trade-offs between these programming languages in computational performance and resource utilization.

1 Introduction

Matrix multiplication is a key operation in many computational fields, such as machine learning, computer graphics, and numerical analysis. Efficient implementation of this operation can significantly impact the performance of applications. This study compares three implementations of matrix multiplication, each written in Python, Java, and C, to evaluate their execution time and memory usage across different matrix sizes. These implementations are tested using various benchmarking tools, and the results are discussed.

2 Implementations

All of this tests where conducted on a Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz and 16 GB of RAM on a Windows 10 machine.

2.1 Python

The Python implementation was developed using Visual Studio Code and tested with `pytest` for benchmarking. The multiplication was performed using nested loops, and memory usage was tracked with `tracemalloc`. The implementation multiplies two matrices by iterating through their elements and accumulating the products.

Python Code:

```
import random

def multiplication(A, B):
    n = len(A)
    C = [[0 for _ in range(n)] for _ in range(n)]

    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] += A[i][k] * B[k][j]

    return C
```

2.2 Java

The Java implementation was developed using IntelliJ and benchmarked with the Java Microbenchmark Harness (JMH). Similar to Python, the matrix multiplication was implemented with three nested loops. JMH was used to measure execution time, and the memory usage was tracked with the `Runtime` class.

Java Code:

```
public class Main {
    public static void main(String[] args) {
        int[][] A = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
```

```
        int[][] B = {
            {9, 8, 7},
            {6, 5, 4},
            {3, 2, 1}
        };

        int[][] result = multiplyMatrices(A, B);
    }

    public static int[][] multiplyMatrices(int[][] A, int[][] B) {
        int n = A.length;
        int[][] C = new int[n][n];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }

        return C;
    }
}
```

2.3 C

The C implementation was developed using Visual Studio Code, and the benchmarking was done with the `gettimeofday` function for execution time and the `GetProcessMemoryInfo` function for memory usage. Similar to Python and Java, the matrix multiplication was implemented using three nested loops.

C Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
```

```
void multiply_matrix(double **a, double **b, double **c, int size) {
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            for (int k = 0; k < size; ++k) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

3 Results

3.1 Execution Time

The following table summarizes the execution time for matrix multiplication in each language. Tests were conducted for matrix sizes of 10x10, 100x100, 500x500, and 1024x1024.

Matrix Size	Python (s)	Java (ms)	C (ms)
10x10	0.000284	0.026	0.00
100x100	0.000268	0.941	0.99
500x500	34.738	163.216	131.44
1024x1024	362.814	2140.572	2677.41

Table 1: Execution Time Comparison

Python demonstrates faster execution for smaller matrices, while C performs better for larger matrices. Java falls in between but shows increasing execution time as the matrix size grows. It should be noted that both Python and Java results are average scores of multiple iterations, while C's are scores of just one iteration

3.2 Memory Usage

The table below shows the memory usage during matrix multiplication.

In terms of memory usage, Java consistently uses less memory compared to Python and C. Python's memory usage grows significantly with matrix

Matrix Size	Python (MB)	Java (MB)	C (MB)
10x10	0.008	1	4.01
100x100	0.617	1	4.33
500x500	15.21	1	10.32
1024x1024	64.19	4.36	29.11

Table 2: Memory Usage Comparison

size, whereas C maintains more efficient memory usage compared to Python but higher than Java.

4 Conclusion

This comparative analysis of matrix multiplication implementations in Python, Java, and C highlights significant differences in performance. Python is highly efficient for small matrix sizes but suffers from longer execution times and higher memory consumption as the matrix size increases. Java offers balanced performance but tends to be slower for larger matrices, although it remains highly memory-efficient. C demonstrates the best performance for larger matrices, with faster execution times but higher memory usage. The choice of programming language for matrix operations should be based on specific application needs, balancing speed and resource consumption.