Basic Matrix Multiplication in Different Languages

Luis Perera Pérez
Grado en Ciencia e Ingeniería de Datos

Academic year 2024/25

Abstract

In this paper, we compare the performance of matrix multiplication implemented in Python, Java, and C. The experiments were conducted using different matrix sizes (from 10×10 to 1000×1000), and the execution times were measured. The results show that C outperforms both Python and Java significantly for large matrices, highlighting the advantages of lower-level languages for computationally intensive tasks. This study provides insights into the trade-offs between ease of development and execution efficiency in these programming languages.

1 Introduction

Matrix multiplication plays a fundamental role in various computer applications, such as scientific computing and machine learning. As the size of data sets increases, it becomes essential to optimize this operation in different programming languages. This article benchmarks basic matrix multiplication in Python, Java, and C, three languages commonly used in both academic and industrial environments.

2 Problem Statement

The multiplication of two matrices of size $n \times n$ has a time complexity of $O(n^3)$. As matrix size increases, this operation becomes computationally expensive, making language efficiency crucial for applications involving large data sets. This paper addresses the problem of how different programming languages—Python, Java, and C—handle matrix multiplication, both in terms of execution speed and memory usage, and which language provides the best performance as matrix sizes increase.

3 Methodology

The matrix multiplication algorithm was implemented in Python, Java, and C using a standard approach with $O(n^3)$ complexity. The code was separated into production and test sections to ensure accurate benchmarking. To measure performance, we used *pytest-benchmark* in Python, *JMH* in Java, and *perf* in C. Each experiment was run multiple times to account for variance. We tested matrix sizes of 10×10, 100×100, and 1000×1000, recording both execution time and memory usage.

4 Experiments

We conducted our experiments on matrix sizes of 10×10, 100×100, and 1000×1000. The results are summarized in Tables 1 and 2.

4.1 How to Run the Code from GitHub

To replicate these experiments, the code for matrix multiplication in Python, Java, and C is available in the following GitHub repository:
https://github.com/LuisPereraPerez/BigData
Follow these steps to download and execute the code:

- **Clone the repository**: Clone the repository by running the following command:

```
git clone https://github.com/LuisPereraPerez/BigData
```

- **Requirements**:
  - For **Python**, install the necessary packages using:

```
pip install pytest-benchmark memory-profiler
```

  - For **Java**, ensure JDK 8 or higher is installed.
  - For **C**, install a C compiler (e.g., `gcc`) and tools like `perf` and `/usr/bin/time`.
- **Run the Benchmarks**:
  - **Execution time (Table 1)**:
    - **Python**: Run the benchmark with:

```
pytest --benchmark-only matrix_multiplication.py
```

    - **Java**: Compile and run the benchmarks:

```
javac MatrixMultiplication.java
java -jar target/benchmarks.jar
```

* **C**: Compile and run the program:

```
gcc -o matrix_multiplication matrix_multiplication.c
perf stat ./matrix_multiplication
```

– **Memory usage (Table 2)**:

* **Python**: Run the memory profiling script:

```
python -m memory_profiler memory_test.py
```

* **Java**: Run the benchmark with GC profiling:

```
java -jar target/benchmarks.jar MyBenchmark -prof gc
```

* **C**: Measure memory usage with `/usr/bin/time`:

```
/usr/bin/time -v ./matrix_multiplication
```

4.2 Results

Matrix Size	Python	Java	C
10×10	0.18 ms	0.007 ms	0.001 ms
100×100	0.436 s	2.00 ms	0.54 ms
1000×1000	189.93 s	5.23 s	2.31 s

Table 1: Execution times for matrix multiplication across different programming languages.

As shown in Table 1, C consistently outperformed both Java and Python, particularly for larger matrix sizes. Python's performance significantly deteriorated with increasing matrix size, while C maintained a clear advantage due to its low-level memory management and optimization capabilities.

In addition to execution times, we measured the peak memory usage during the matrix multiplication process. The results are shown in Table 2.

Matrix Size	Python (MB/op)	Java (MB/s)	C (MB/op)
10×10	23.49 MB	393.61 MB/s	1.35 MB
100×100	24.58 MB	120.27 MB/s	1.92 MB
1000×1000	142.68 MB	2.68 MB/s	27.50 MB

Table 2: Peak memory usage for matrix multiplication across different programming languages.

The results in Table 2 show that Python used the most memory, especially for larger matrix sizes, while C was more efficient in terms of memory consumption.

Java's memory usage was characterized by a high memory allocation rate due to frequent memory allocations during the computation process.

5 Analysis

- **Python:** Exhibits the highest execution time and memory usage, particularly for larger matrices. This is due to Python being an interpreted language that introduces significant computational overhead, along with a lack of optimizations in its list-based implementation for matrix multiplication.
- **Java:** Performs better than Python, thanks to its compiled nature and the efficiency of loop execution. However, memory usage in Java is notably high due to the necessity of allocating memory for temporary objects during the computation process, resulting in a high memory allocation rate.
- **C:** Demonstrates the fastest execution times and the lowest memory usage. This highlights the efficiency of low-level memory management in C. The ability of C to manage memory manually, coupled with the absence of runtime overhead, makes it the ideal choice for computationally intensive tasks.

6 Conclusion

This paper compared the performance of matrix multiplication across Python, Java, and C, focusing on both execution time and memory usage for different matrix sizes. The results clearly indicate that C provides superior performance, especially for large matrix sizes, due to its lower-level memory management and optimization potential. Python, while the slowest and most memory-intensive, offers simplicity and ease of development, making it more suitable for smaller-scale tasks where performance is less critical. Java sits between the two, providing reasonable performance but with higher memory allocation rates.

These findings can assist developers in choosing the right tool for the right task, balancing performance with ease of development.

7 Future Work

Future research could explore optimized implementations, such as Strassen's algorithm, to further improve performance. Additionally, parallelization in C and Java or using optimized libraries like NumPy in Python could provide more comprehensive insights. Expanding the benchmarks to include modern programming languages such as Rust or Go would also offer a broader performance comparison.

GitHub Repository

The source code for the matrix multiplication implementations in Python, Java, and C can be found at:https://github.com/LuisPereraPerez/BigData.