TJAA Vol. **0**, *Issue* 0, *p.*(0000). *Research Article*

Comparing different programming languages for matrix multiplication

Eduardo López Fortes
Universidad de Las Palmas de Gran Canaria
Escuela de Ingeniería Informática
https://github.com/eduardolfULPGC2003/First-MatrixMultiplication

Accepted: XXX. Revised: YYY. Received: ZZZ.

Abstract
Planning a new project always involve a step in which a programming language must be chosen: sometimes this decision is direct since the task to be performed is very specific or the developer wants to specifically develop the software in a certain programming language. However, other times the programming language may not be that important, so the developer explores different approaches and chooses the best option based on different factors such as speed, performance (which not always synonymous with speed), memory usage, whether if it is object-oriented or not, etc. This study focused on measuring the speed of three different programming languages (Python, Java and C) by comparing them while doing a dummy task such as matrix multiplication: we used benchmarking frameworks, which includes pytest-benchmarking for python; Java Microbenchmark Harness for Java and Perf for C. Results have shown that Python is the slowest overall, followed by Java, and the fastest, C. More specifically, Python performs well for small matrix but struggles with larger ones; Java works well for medium-sized matrix; and C performs very well for larger matrix, even 10.000x10.000.

Key words: benchmark – matrix – multiplication

1 Introduction

"Benchmarking" is a widely used word in economics and other industries and consists of comparing business processes and performance metrics to industry bests and best practices. This concept first appeared in the 1980s when the company Xerox was interested in comparing its own performance with competitors'. Benchmarking is just an abstraction of measuring different processes and comparing them: when trying to know which route is faster, measuring the time it takes to reach the destination on every route and comparing them; that is basically benchmarking.

This concept has been adapted so that it can be used in computer science, but the main idea remains the same: benchmarking different programming languages and comparing them under the same conditions. Unlike benchmarking in the real world, benchmarking programming languages allows to always have almost the same conditions during the experiments: initial state can be set, and the hardware will remain the same during the experiment; with the exception, of course, of each programming language nature.

Benchmarking is a very common task in the field of computer science because is the only way in which the better solution (methodology, framework, programming language, etc) can be chosen objectively. However, people tend to benchmark running the code once, which can lead to biased results. This paper is going to deep on it on the methodology part.

In this study three programming languages were compared: C, Java and Python. The task chosen for benchmarking them is the matrix multiplication. Matrix multiplication consists in the multiplication of two matrix and despite it may seem a dummy task, if matrix are large enough, this task becomes really expensive computationally talking. Therefore, choosing the right programming language can make in the future our software faster and more efficient.

2 Problem Statement

In the process of choosing a programming language, speed is going to be surely on the most important measures, so making the right decision is critical for the software to succeed or not. Since developing the software in three different programming languages is not sustainable, the need of finding a way to compare them with its advantages and disadvantages arise.

Some of the characteristics of the programming languages can be compared in a qualitative way, such as whether if it is a low-level or high-level programming language, compiled or translated, script-based or not... But some other characteristics not, and one of them is, obviously, speed.

There are many studies that have addressed this issue, but since speed relies in the hardware specifications of the machine, it is a good idea to make a test on the machine where the software is going to run and with its own hardware specifications.

As stated before, writing the entire software three times is a waste of time, so a task that can be compared to the original software must be found. A candidate for that is the matrix multiplication: a simple task for small sizes of the matrix , but as we expand the matrix, the task becomes more expensive.

Performing this task on the three programming languages will give an idea of which of them is better for a certain situation.

Matrix multiplication has always been a study topic in mathematics because of this reason and therefore, many ways

to optimize this task has been developed. However, this topic goes beyond the scope of this paper.

3 Methodology

3.1 Baseline

The first step was to write the code that we were to use to benchmark the programming languages. The code is different for each programming language, but the pseudo-code, which is the same for the three programming languages is the following:

Algorithm 1 Matrix Multiplication

Input: x ▷ Size of the matrices
Initialize A and B as $x \times x$ matrices with random values
Initialize C as a $x \times x$ matrix with zeros
for $i = 1$ to x **do**
    for $j = 1$ to x **do**
        for $k = 1$ to x **do**
            $C[i][j] \leftarrow C[i][j] + A[i][k] \times B[k][k]$
        end for
    end for
end for
return C

There are two different approaches for this experiment depending on the grade of precision:

a. Taking into account the process of creating both matrix as a part of the whole experiment
b. Treating the creation of the matrix as a pre-processing and therefore, leave it apart from the experiment.

3.2 Test Developing

The experiment is more precise if we consider the process of creation is inherited from the matrix multiplication. As a result, when benchmarking, it was not treated as setup.

For this experiment we tested different sizes of matrix: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 y 2048. However, there were some complications that are going to be exposed later on.

For every size in every programming language, the performance would be sized by measuring the time it took to the program to be executed (including the process of creating the matrix) in milliseconds.

In addition, in order to have consistent and reliable results, the following procedure was followed:

a. An arbitrary number of warm-ups rounds would be set to run before testing the program
b. The test would be ran several times (5)
c. This two steps would form a complete round. This process would be completed 3 times before moving to testing the next size

3.3 Software used

Intellij Idea along with a maven project were chosen for running and editing Java code, and then Java Microbenchmarking Harness for the testing purposes - added as a dependency in the pom.xml file:

- `Java 11`
- `Java Microbenchmarking Harness 1.35`

Visual Studio Code was the code editor chosen for editing Python code and the library Pytest were used for running the tests:

- `Python 3.10.9`
- `Pytest 8.3.3`

In order to run C code and install libraries, the best option was to create a Virtual Machine and install any Linux distribution on it. For the sake of simplicity, Ubuntu was chosen. Perf was the library used for testing the code:

- `Virtual Box 7.0.6`
- `Ubuntu 24.10`
- `gcc 14.2.0`
- `Perf 6.11.0`

3.4 Machine Specifications

Hardware specifications influences notably on the testing results since many factors could change the way in which the program is executed: amount of RAM, type of processor - including number of cores -, graphic card (if applicable), etc.

For that reason, the specifications of the machine in which the test were ran are the following:

- `Machine: AMD64`
- `Processor: AMD Ryzen 7 5700U with 8 cores, 1.80 GHz`
- `RAM: 8GB`
- `SSD storage: 512GB`
- `Operating System: Windows 10 Home`
- `Version: 22H2`

The specifications for the virtual machine are:

- `Processor: 5 cores`
- `RAM: 4GB`
- `SSD storage: 25GB`
- `Operating System: Ubuntu`
- `Version: 24.10`

4 Experiments

The very first idea was to try to measure all matrix sizes with the above described procedure. However, 1024 was the maximum size it could be tested since large matrix would take a long time to be executed, making it even worse with the testing procedure. Therefore, tests regarding Python and matrix sizes 2048 and 4096 are going to be shown as NM (not measured).

As shown in Table 1 the results of the benchmarking can be appreciated. The table shows the average runtime for every size on each programming language. Times for medium-sized matrix are much better in Java rather than in Python or C, but for size 4096 the tendency changes and C gave the best performance. We can expect that for bigger matrix (10000, 20000, etc) C should have best performance than Java.

The execution times for Python are not bad for small matrix until size 32, but after that, results compared with Java and C are extremely bad. Despite 2048 and 4096 sizes were not tested, based on the previous results and based on the fact that we are testing an $O(n^3)$ algorithm, we should expect high running times and absolutely, not affordable.

C results are mediocre for medium-sized matrix, but the situation changes when encountering extremely large matrix.

Table 1. Tiempos Promedio de Ejecución para Diferentes Tamaños de Matrices medido en milisegundos

Size	Python	C	Java
1	0.0044	1.22	**0.0001**
2	0.0087	1.07	**0.0001**
4	0.029	1.06	**0.0001**
8	0.15	1.11	**0.001**
16	1.02	1.31	**0.004**
32	7.39	1.46	**0.0024**
64	56.30	2.91	**0.182**
128	438.40	11.48	**1.449**
256	3442.57	59.34	**15.708**
512	28116.11	2274.06	**201.711**
1024	232314.48	16028.60	**3852.485**
2048	NM*	97595.77	**81518.070**
4096	NM*	**588432.24**	777841.935

*NM: Not Measured

5 Conclusions

Taking a look on the results it can be appreciated that, besides no clear solution has been found, we can conclude the following:
- Python is the lowest one over all of them but works fine for small sizes.
- I expected C to be the fastest of the three programming languages and maybe the fact of running the code in a virtual machine biased the results, but at the end, despite we can not conclude that it is slow (Python is slow, C actually performs well), for medium-sized matrix C is not a programming language you would use unless you have to. However, C will be the programming language to be chosen if your matrix are extremely large.
- Java is the closest programming language to be considered "the winner" of this experiment, as it has shown that works pretty well for most of matrix sizes.

Despite that this work seems very simple, throughout the developing many problems have been encountered and the most significant are going to be exposed.

While doing the benchmarkin in Java, the jar file was not created properly and I therefore could not follow the instructions on the virtual campus. I had to install the Intellij pluggin for jmh and only then worked.

Benchmarking in Java was very challenging since running a benchmark for a 4096x4096 matrix takes a lot of time. So the only option that worked for me was to let the benchmarking running through the entire night.

I have to admit that benchmarking Python was the easiest part and no problems were faced in this part.

My first idea to run C code was using CLion and running it directly on my PC, but the I realized that it was simpler to just create a virtual machine and install a Linux distribution on it. It took longer because of the downloading and installation process, but simpler. It was a trade-off that I was committed to take.

Finally, I would like to apologize for the absence of graphics: I could not manage to place them in the text, overleaf would place them in an arbitrary order. I tried with the tag [h], [H] and multiple things, but none of them worked. I will upload the graphics to the Github repository in case the reader wants to take a look of them.

6 Future Work

In the section of "Problem Stating" I stated that the optimization of matrix multiplication goes beyond the scope of this paper and therefore, one option for future work is to try to implement some of the algorithms and try to to optimize this process.

As I exposed before in this paper, the matrix multiplication is nothing new, people has always tried to develop new methods for doing this operation cheaper. Some of them involves algortihms that makes less multiplication operations such as the Strassen algorithm, which is able to transform the complexity of the problem from $O(n^3)$ to $O(n^2.807)$. There exists other algorithms called super-methods which in practice are not applicable.

Another way to face the problem of optimizing the process of matrix multiplication is to change the way in which the matrix is stored: if the matrix is likely to contain many 0's, it is a better option to save just the position where the value is not 0 and, of course, its value.

Multithreading or distributed programming is an interest approach that instead of changing the algorithm or the way in which the matrix is stored, faces the process of the matrix multiplication itself.