Matrix multiplication optimization

Jorge Hernández Hernández [1]

[1]jorge.hernandez12@alu.ulpgc.es

October 2022

Abstract

The intention of this study is to solve, by means of optimization, the execution time of a code base that performed the multiplication between different matrices. In order to make this possible, new implementation techniques have been implemented, among which we find Streams, Executor Service, Atomic, Threads... To see the performance, a benchmarking has been performed for both dense and sparse arrays of different sizes. The TDD (Test-Driven Development) technique has been used in the development, in addition to respecting the SOLID principles, separating the code functionalities in different classes. It can be seen that with the use of the above techniques both types of matrices, both dense and sparse, have a very high improvement compared to the methods that were already implemented previously. This clearly shows how parallel programming is much more efficient than standard programming, showing how the use of parallel programming is highly recommended for almost all types of programs.

Keywords— Parallelism, Java, TDD, Matrix Multiplication, Synchronization, Threads, Semaphores

1 Background

Matrix multiplication is something basic within linear algebra, it is an operation used in a large number of mathematical algorithms. Hence the importance of the search for an implementation of this problem as fast and efficient as possible. To reach the solution we have used Bench-marking, programming in Java.
This time we have implemented one of the most efficient forms of programming for large matrices as is the case of parallel programming, the goal is to use several processors or computers to perform the calculation in parallel, rather than sequentially on a single processor. This can significantly speed up the computation of large matrices, since the computation can be divided among several processors.
There are several ways to perform parallel matrix multiplication using Java. One option is to use the java.util.concurrent package, which provides a set of classes and interfaces that support parallel programming in Java. These classes include Executor,

ExecutorService, and ForkJoinPool, which allow you to submit tasks to be executed concurrently.
Another option is to use the Java Parallel Streams API, which was introduced in Java 8 and provides a high-level interface for parallel programming in Java. The Parallel Streams API allows you to easily parallelize operations on collections and arrays using the parallelStream() method.
There are also several third-party libraries that can be used for parallel matrix multiplication using Java, such as Apache Commons Math and Parallel Colt. These libraries provide optimized implementations of matrix multiplication and other linear algebra operations, and can be used to perform parallel computation using a variety of approaches, including multithreading and distributed computing.
With this study our intention is to provide a faster, more readable way to multiply the matrices for future projects, based on the advances of other great scientists in the sector.

2 Problem statement

The calculation can be very time consuming in the case of large matrices, a point we will focus on throughout this papper, it involves multiplying and adding a large number of elements. This can be a bottleneck in applications that rely on matrix multiplication, especially when the matrices are very large or the computation must be performed repeatedly.

To solve this problem, parallel programming techniques can be used to perform matrix multiplication in parallel, using multiple processors or computers. This can significantly speed up the calculation, since the work can be divided among several processors. But in order to be able to carry out this type of programming it is very necessary to understand the following

It must be ensured that the parallelized code is correct and works well. This may involve careful design of the parallel algorithm, optimization of resource usage (such as processors and memory), and handling problems such as data dependencies and race conditions.

3 Test environment

Once we have clarified which is the problem to solve or to treat in this paper we will give the specifications of the machine in which the different tests and the nbenchmarking have been made so that you can have an idea of what is due to the speed or delay of the processing time of one of the tasks in comparison to other pappers that are on the web. In this case we are looking at a ROG Zephyrus G14, and the specifications of this machine are as follows:

- Processor AMD Ryzen 9 4900HS with Radeon Graphics 3.00 GHz
- Installed RAM 16.0 GB (15.4 GB usable)
- System type 64-bit operating system, x64-based processor

As operating system we have a Windows on which we can see:

- Windows 11 Home Edition
- Version 21H2
- Operating system version 22000.1335

4 Methodology

Parallel programming is a programming technique that allows you to harness the processing power of multiple cores or processors to execute tasks simultaneously. This can significantly improve the performance of a program, especially in tasks that are computationally intensive or involve processing large amounts of data. In the context of this study, new implementations of multiplications focused on parallel programming have been created, divided into three types: standard, parallel, and synchronization.

Standard implementations include multiplications such as standard and transpose multiplication, which existed previously. These implementations are commonly used in programming and can be efficient for certain types of problems. However, in situations where large amounts of data need to be processed or many calculations need to be performed, it may be beneficial to use more advanced parallel programming approaches to improve performance.

Parallel implementations include multiplication using Streams and multiplication using Executor Service. These implementations use specialized programming techniques to facilitate code execution on multiple nodes or processors. This allows the program to run faster by leveraging the processing power of multiple cores.

Finally, synchronization implementations include atomic array multiplication along with semaphores and threads. These implementations use synchronization techniques to ensure that different parts of the program execute consistently and avoid shared data access conflicts.

All these new implementations have been developed for dense arrays, but not for sparse arrays, due to the high complexity of this implementation. However, in order to compare the different implementations with each other and to evaluate the impact of parallel programming on the performance of matrix multiplications, the different values of dense matrices of different dimensionalities and sparse matrices of medium and large size have been plotted. In this way, it is possible to visualize the differences between using

5 Experiment

Within the experimentation we can see two different sections of work on the one hand we have the experimentation with dense type matrices these are square, and on the other hand the experimentation with sparse type matrices of gan size and of a much smaller size.

If we focus on the dense type matrices, we can see how in this case as the number of elements of the matrices to be multiplied, i.e. n x n, the efficiency of the processes using parallel programming increases in comparison to those that had been applied at the beginning with a standard programming.

Translated with www.DeepL.com/Translator (free version)

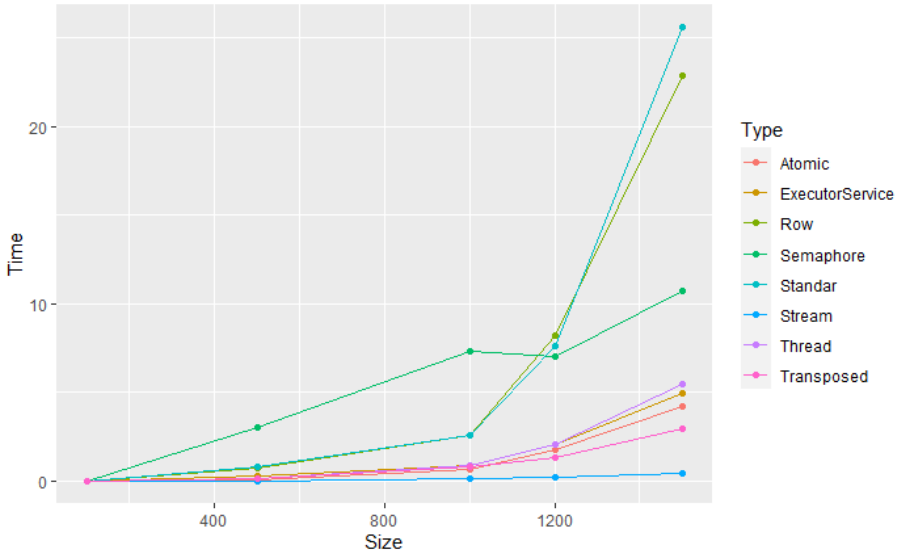

Figure 1: Time Compare Dense

How can you see in figure 1 we can clearly see how of the different methods used the most obvious is represented by blue color, referring to the operations performed using Streams, and the worst of all is represented in a lighter shade of blue, which refers to the way of standard multiplication. As expected, as the dimension increases, the computation time increases, but if we recall the parallel multiplication methods that have been discussed throughout the paper, we see that none of them exceeds 5 seconds of computation time, unlike the "standard" methods, three of which exceed 10 seconds in computation time. If we test with the implemented multiplication methods for sparse matrices we can see how we have used two types of sparse matrices, one of standard size and another one of much larger size of approximately 1M elements.

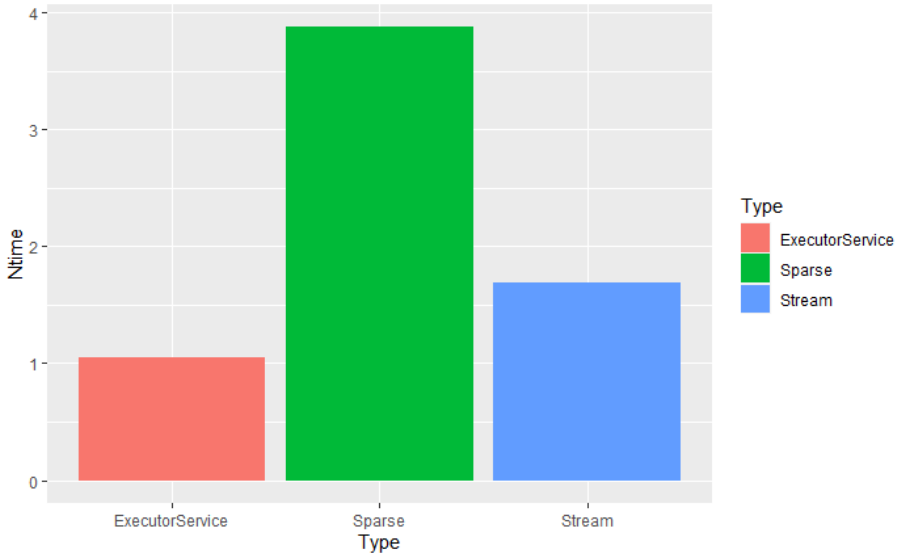

Figure 2: Time Compare Standar Sparse

How can you see in figure 2 we have this time the sparse matrix of normal size, in which we can clearly see the superiority of the methods that use parallel programming, of these two methods that implement it we see how it is faster the method in which Executor Service is used.

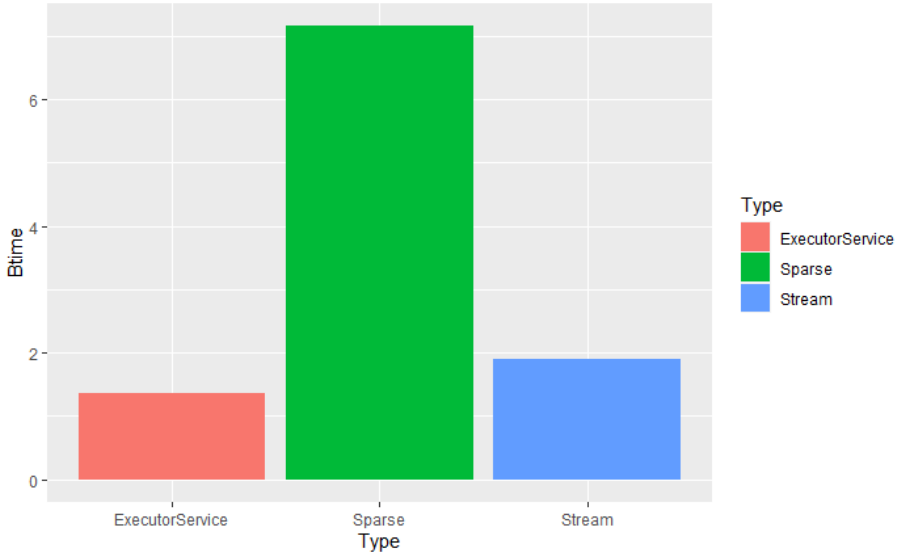

Figure 3: Time Compare Big Sparse

In this figure 2 we are looking at the results given for the high dimensional sparse matrix, we can see how they are very similar to the one we use with lower dimensionality, but the difference between the parallel and the classical methods is much larger, while the difference between Executor Service and Streams, although the former is still slightly faster than the latter.

6 Conclusions

In conclusion, parallel programming is an essential technique for matrix multiplication because of its ability to increase computational speed and improve computer efficiency by allowing several operations to be executed simultaneously instead of having to wait for them to complete one at a time. In addition, parallel programming also allows the use of high-capacity processing systems, such as computer clusters or supercomputers, which are especially useful for computationally intensive problems. Finally, parallel programming can also improve the scalability of an application by allowing more processing resources to be added to handle large data sets more efficiently. In summary, parallel programming is more efficient for matrix multiplication due to its ability to perform multiple operations simultaneously and increase computational speed. One of the main advantages of parallel programming is its ability to take advantage of the processing power available on systems with high processing power. This can be especially useful for problems involving large data sets, as seen in the data depicted in figure 2 and figure 3

7 Future work

Due to the lack of time for the development of the project we have been forced to postpone a large number of tests and experiments. Next we will say some of these points to improve. First of all, We will seek to apply the range of possibilities for multiplying, using parallel programming, sparse matrices of all dimensionality. It would also be a plus if the sparse matrices could be created randomly to see the results of more specific matrices rather than relying on the matrices that have been downloaded.

References

[1] https://GitHub.com/