Untangling the maze of massive data: A mosaic-based approach.

Alba Martín Lorenzo[1]
[1] *alba.martin112@alu.ulpgc.es*

Abstract

The importance of minimizing memory space and improving computational speed has become a priority, especially when dealing with large masses of data, as we could save a lot of time. This paper presents a new technique for multiplying matrices using mosaics instead of the traditional way. A detailed analysis of the performance of this technique is provided and examples of its application in different contexts are presented. This technique is expected to have a significant impact on the speed and efficiency of various applications and algorithms involving matrix multiplications, especially those requiring the processing of large amounts of data or in systems with limited computational resources. In addition, the MapReduce technique is used to parallelize the matrix multiplication process with mosaics, which further increases efficiency in handling large volumes of data. We discuss how this combination of techniques optimizes performance and reduces computational time compared to traditional methods. Possible limitations and future improvements of this technique and its application in different fields are also discussed. In summary, the technique of matrix multiplication with mosaics using Map Reduce is presented as a highly efficient solution for managing large volumes of data and its application has great potential in various fields.

Key words: Speed, Storage, Execution time, Mosaics, MapReduce, Matrix Multiplication

1 Introduction

Matrix multiplication is a fundamental operation in various areas of mathematics and computer science, and is used in a wide variety of contexts, from numerical computation to the transformation of coordinates into 3D graphics.

Although there are several ways to perform this operation, the traditional one has been the most widely used due to its simplicity and ease of implementation. However, in situations where it is necessary to perform matrix multiplications quickly and efficiently, this way can be inefficient. For example, when multiplying large matrices or in systems with limited computational resources, the traditional way may require too much time or resources to complete.

This tiled matrix multiplication technique is a valuable tool that can have a significant impact on the speed and efficiency of various applications and algorithms involving matrix multiplications.

A new technique for data processing in distributed systems using the MapReduce technique is also presented. The technique consists of dividing data into small fragments or blocks and processing them independently on different nodes or machines, and then combining the results obtained. This technique allows processing large volumes of data in an efficient, scalable and fault-tolerant manner.

2 Problem statement

The MapReduce process is divided into two steps: the "Map" phase and the "Reduce" phase.

The "Map" phase is the first phase of the MapReduce process. During this phase, each element of the input data set is processed and an intermediate set of key-value pairs is produced. For example, if the input data set is a set of lines of text, the "Map" phase might process each line and produce a key-value pair for each unique word found on that line, where the key is the word and the value is a counter of 1.

The "Reduce" phase is the second and final phase of the MapReduce process. During this phase, the intermediate key-value pairs produced by the "Map" phase are taken and combined to produce a final set of results. For example, if the "Map" phase has produced an intermediate set of key-value pairs such as (word1, 1), (word2, 1), (word1, 1), (word3, 1), the "Reduce" phase might combine those pairs to produce a final set of results such as (word1, 2), (word2, 1), (word3, 1), indicating that word "word1" appeared twice in the input data set, while word "word2" and word "word3" appeared once each.

Matrix multiplication with mosaics is done as follows:

Suppose we have two square matrices A and B, and we want to compute the product of both matrices using MapReduce and the tiled multiplication technique. Matrix A has 3 rows and 3 columns, and is represented as follows: A = [[a11 a12 a13], [a21 a22 a23], [a31 a32 a33]] Matrix B has 3 rows and 3 columns, and is represented as follows: B = [[b11 b12 b13 b13], [b21 b22 b23], [b31 b32 b33]]. The product of matrices A and B is a matrix C of 3 rows and 3 columns, which is represented as follows: C = [[c11 c12 c13 c13], [c21 c22 c23], [c31 c32 c33]], where each element cij of matrix C is calculated as the product of row i of matrix A by column j of matrix B. For example, element c11 is calculated as c11 = a11 * b11 + a12 * b21 + a13 * b31.

To use the tiled multiplication technique, we divide matrices A and B into square tiles of 2 rows and 2 columns each. Matrix A would be divided as follows: A1 = [[a11 a12], [a21 a22]] A2 = [[a13 0], [a23 0]] A3 = [[a31 a32], [0 0]] A4 = [[a33 0], [0 0]] And matrix B would be partitioned as follows: B1 = [[b11 b12], [b21 b22]]; B2 = [[b13 0], [b23 0]]; B3 = [[b31 b32], [0 0]]; B4 = [[b33 0], [0 0]].

Then, we could use MapReduce to calculate the product of each pair of tiles (A1,B1), (A1,B2), (A2,B3), (A2,B4), (A3,B1), (A3,B2), (A4,B3), (A4,B4). For example, the sum of the product of the tiles (A1,B1), (A1,B2), (A2,B3), (A2,B4) would give us the elements c11, c12, c21 and c22 of the C matrix.

Once we have calculated the product of all the tiles, we could combine the results to obtain the complete C matrix. For example, we could obtain a final set of results as follows: (((1,1), c11), ((1,2), c12), ((2,1), c21), ((2,2), c22), ((3,1), c31), ((3,2), c32), ((1,3), c13), ((2,3), c23), ((3,3), c33)), where each key-value pair consists of a pair of coordinates (i,j) indicating the row and column of the corresponding C matrix, and a value cij which is the corresponding element of the C matrix.

3 Method

3.1 Analysis and design

To implement the code, a class diagram was first created in StarUML, which represents the fundamental objects of the system that will be part of the solution and the relationships between the system elements, in order to see what components the system needs and how they influence each other.

For this purpose, the SOLID methodology, object orientation, interfaces and class polymorphism have been taken into account.

We have three main interfaces: Matrix, Builder and MatrixMultiplication, structured as shown in Figure 1. From them we generalize and create the rest of the classes. We have a DenseMatrix class for initializing dense matrices, a child of the Matrix interface, with its own DenseMatrixBuilder constructor, which generalizes from the Builder interface.

We will have a child class of the MatrixMultiplication interface to be able to make different types of multiplications: TiledMultiplication.

Finally, we will have four classes on which TiledMultiplication will depend: Reducer, Mapper, Partitioner and EdgeZerosEliminator. Another point to take into account prior to the implementation has been the use of unit tests, to check the correct operation of the main classes.

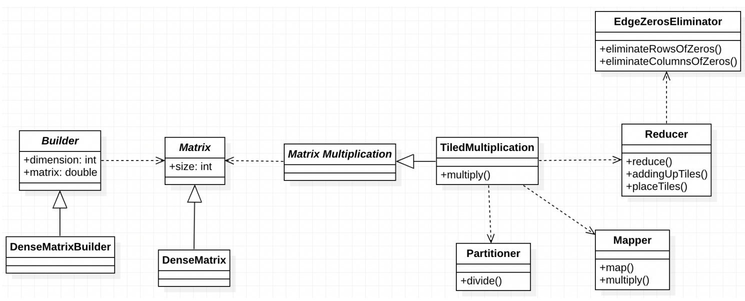

Figure 1: Class diagram for Matrix Multiplication implementation

3.2 Test environment

Once the representation of the system in the UML class diagram is done, we start the implementation.

The implementation and testing is done in the Java programming language (version 11), which is a compiled language widely used for coding web applications. It has been a popular choice among developers for over two decades, with millions of Java applications in use today. The tests will be run in the IntelliJ integrated development environment (IDE), specifically for the Java programming language, developed by the company JetBrains.

* Note that the tests will be run on an Legion 5 15IMH05, with a Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz and 16 GB of RAM, so the execution program under other conditions will vary.

Unit tests to check the performance of the classes have been performed with different use cases in the multiplication functions, checking mathematically (by multiplying a matrix by a random vector by multiplying it by A and B). All this has been executed under JMH, a Java harness to build, execute and analyze nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM. All of them have been developed under the same circumstances and the same type of computer, described above.

4 Conclusion

Matrix multiplication is a fundamental task in numerical computation and is used in a wide variety of applications, such as artificial intelligence, data science, simulation and optimization. However, when matrices are large, computing their multiplication can be costly in terms of time and resources.

In this context, matrix multiplication with mosaics using the MapReduce technique has been shown to be an efficient and scalable method for solving large-scale problems. The use of tiling allows matrices to be divided into smaller blocks, which reduces the size of the matrices that must be stored in memory and processed at each processing node.

The MapReduce technique, on the other hand, makes it possible to distribute the matrix multiplication calculation among several nodes in a cluster, which makes it possible to take full advantage of the available processing capacity. This results in a great reduction in computation time, since the operations are performed in parallel.

However, a possible disadvantage of this approach is that it can require a large amount of memory to store the tiles, since each block must be stored on a processing node. In addition, the process of dividing the matrices into tiles and distributing them among the nodes can be complex and require a large amount of pre-configuration.

In summary, matrix multiplication with tiling using the MapReduce technique is an efficient and scalable method for solving large-scale problems, but it can require a large amount of memory and preconfiguration.

5 Future Work

A possible future work from this paper could be to investigate the use of the Hadoop technique in Java to further improve the efficiency and scalability of array multiplication. Hadoop is an open source software framework used to store and process large data sets in a network of distributed nodes. By combining Hadoop with the tiling technique and MapReduce, even better results could be obtained in terms of runtime and scalability. In addition, one could investigate how the configuration of Hadoop nodes affects the efficiency of matrix multiplication. In summary, using Hadoop together with Mapreduce and mosaics could be a possible way to further improve the efficiency and scalability in matrix multiplication in Java.