# 

# Performance evaluation of a single core

### CPD Project 1

**Group T03\_G17**

* Guilherme Valler Moreira up202007036
* Tomás Pereira Maciel up202006845

## Problem Description and Algorithms Explanation

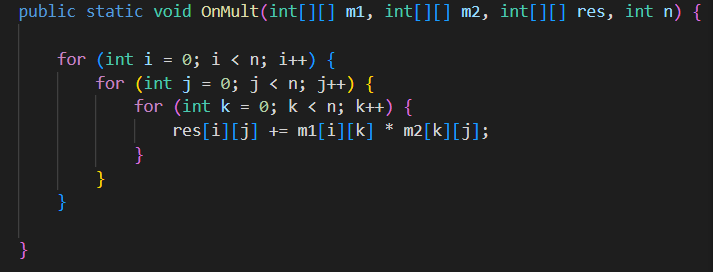
In this project, we aim to study the effect on the processor performance of the memory hierarchy when we access a large amount of data. To do this, we will use algorithms to multiply two matrices and Performance API (PAPI) to collect relevant performance measures of the program execution.

To achieve this goal, the project is divided in three different parts:

1. Download the example file from moodle that contains the basic algorithm in C/C++ that multiplies two matrices, i.e. multiplies one line of the first matrix by each column of the second matrix (matrixproduct.cpp). Implement the same algorithm in another programming language (just one), such as JAVA, C#, Fortran, etc, of your choice.
2. Implement a version that multiplies an element from the first matrix by the correspondent line of the second matrix, using the 2 programming languages selected in 1.
3. Implement a block-oriented algorithm that divides the matrices in blocks and uses the same sequence of computation as in 2, using C/C++.

* **Simple Multiplication**

The algorithm that we implemented to do the simple matrix multiplication consider that each matrix is represented by an array of arrays, since we thought it would be the best option to represent matrices and to access elements in a easy way.

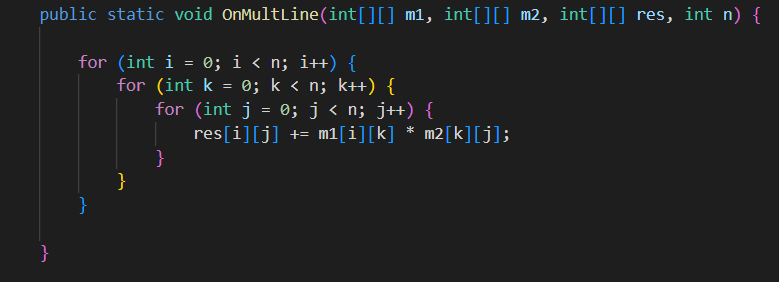


So, we did 3 for-cycles in which the variant i represents res’ and m1’s line, j represents res’ and m2’s column and k represents m1’s line and m2’s column (res = m1 x m2).

* **Line Multiplication**

This type of multiplication works by multiplying an element from the first matrix (m1) by the correspondent line in the second matrix (m2).

Basically, this type of multiplication is a variation of the simple multiplication (presented in point 1), so we reused the code with a small modification to implement this multiplication correctly.



As we can see, the only alteration made to this algorithm was change the 2nd loop with the 3rd of the first algorithm. With this small alteration, this algorithm has a huge increase in his efficiency when compared to the first algorithm.

* **Block Multiplication**

The third segment of this project will involve utilizing Block Matrix Multiplication. This particular method involves breaking a larger matrix into smaller matrices and then performing multiplication on them instead. To execute this algorithm, there are a total of 6 for loops.

The three outer loops are responsible for selecting the submatrices needed for computation, while the three inner loops perform the Multiline Multiplication on the selected submatrices.



The first if-condition verify if the size of the blocks size is valid, since it needs to be a divisor of the matrix’s size (for example, a matrix with size 4x4 can have blocks of size 2x2 but can’t have blocks of size 3x3).

* **Three first loops (outer loops):**
  + The variable of the first outer for loop determines the range of the fourth outer loop which specifies the lines to be extracted from matrices A and C to create their respective submatrices.
  + The variable of the second outer for loop determines the range of the fifth outer loop, which specifies the columns to be extracted from matrix A and the lines to be extracted from matrix B to create their respective submatrices.
  + The variable of the third outer for loop determines the range of the sixth outer loop (i.e., the third inner loop), which specifies the columns to be extracted from matrices B and C to create their respective submatrices.
* **Three last loops (inner loops):**
  + The variant i represents res’ and m1’s line, j represents res’ and m2’s column and k represents m1’s line and m2’s column (res = m1 x m2).

By splitting the matrix into smaller matrices, this algorithm is even more efficient than the second one (especially in larger matrices) as we will see in the results’ analysis.

## Performance metrics

These experiences were made all in the same machine, equipped with an Intel Core i7-9700 @ 3.00GHz processor and running on a Linux environment.

To measure the algorithm’s performance, besides the two different programming languages, we tested both with different sizes of the matrices as we will see next.

The metrics that we used to evaluate the performance of the algorithms are:

* Time – execution time of the algorithm
* L1\_DCM – Level 1 data cache misses
* L2\_DCM – Level 2 data cache misses
* L2\_DCA – Level 2 data cache acesses
* L3\_DCA – Level 3 data cache acesses