**Faculdade de Engenharia da Universidade do Porto**



**CPD Project 1**

**Licenciatura em Engenharia Informática e Computação**

**Turma 6 - Grupo 16**

Index

**1.Problem Description**

**2.Algorithms**

# Problem Description

This project aimed to assess the impact of memory hierarchy on processor performance during extensive data access, utilizing matrix multiplication as a case study due to its computational intensity and relevance across various fields. We measured program performance using the Performance API (PAPI) across different programming languages and algorithm implementations. The tasks included:

* Comparing processing times of matrix multiplication in C/C++ and another chosen language (in our case, Java), with matrix sizes ranging from 600x600 to 3000x3000.
* Analyzing a line-by-line multiplication approach and a block-oriented strategy in the selected languages, including performance measurement for large matrices up to 10240x10240 in size.
* Exploring parallel processing techniques to enhance computational efficiency and assess their impact on performance.

# Algorithms

2.1 Simple Matrix Multiplication

We were given a basic C/C++ algorithm that multiplies two matrices, where one row from the first matrix is multiplied by each column of the second matrix. We opted to implement this algorithm in Java as our alternative programming language choice. Here is the pseudo code:

for(i=0; i<m\_ar; i++)

{ for( j=0; j<m\_br; j++)

{ temp = 0;

for( k=0; k<m\_ar; k++)

{

temp += pha[i\*m\_ar+k] \* phb[k\*m\_br+j];

}

phc[i\*m\_ar+j]=temp;

}

}

## 

2.2 Line Matrix Multiplication

The line multiplication is a variation of the basic multiplication, where the entries in the ith row of A are multiplied by the corresponding entries in the jth row of B and thus accumulated in the respective position of the resulting matrix. Here is the pseudocode:

for(i=0; i < m\_ar; i++)

for(k = 0; k < m\_ar; k++)

for(j = 0; j < m\_br; j++)

phc[i \* m\_ar + j] += pha[i \* m\_ar + k] \* phb[k \* m\_br + j];

2.3 Block Matrix Multiplication

The block matrix multiplication algorithm enhances computational efficiency by dividing the input matrices into smaller blocks, allowing for improved memory cache utilization. It iterates over the matrices in increments defined by bkSize, focusing on submatrices for each multiplication step. This approach ensures that boundary conditions are handled, avoiding out-of-bounds errors for matrices not evenly divisible by bkSize. By processing these blocks, the algorithm optimizes memory access patterns, reducing cache misses and accelerating computation, particularly beneficial for large matrices. Here is the pseudocode:

for (i = 0; i < m\_ar; i += bkSize)

for (k = 0; k < m\_ar; k += bkSize)

for (j = 0; j < m\_br; j += bkSize)

for (l = i; l < min(i + bkSize, m\_ar); l++)

for (n = k; n < min(k + bkSize, m\_ar); n++)

for (m = j; m < min(j + bkSize, m\_br); m++)

phc[l \* m\_ar + m] += pha[l \* m\_ar + n] \* phb[n \* m\_br + m];