# $1^{st}$ CPD Project

Informatics and Computing Engineering, FEUP

3LEIC10, Group 11

Adriano Machado, up202105352@up.pt.
André Rodrigues, up202108721@up.pt.
Daniel Dória, up202108808@up.pt.

## 1 Problem Description

This project aims to analyze the impact on CPU performance during the processing of extensive datasets using matrix multiplication. Divided into two parts, the study will evaluate performance metrics in both single and multi-core environments. By implementing diverse algorithms in C++ and Julia for matrix multiplication and leveraging the PAPI API to gather relevant performance data directly linked to CPU activity, we aim to provide valuable insights into the effectiveness of different techniques on processor performance.

## 2 Algorithms Explanation

In the initial phase of this project, three distinct matrix multiplication algorithms were employed:

1. Column Matrix Multiplication
2. Line Matrix Multiplication
3. Block Matrix Multiplication

For both the **Column Multiplication** and **Line Multiplication** algorithms, a comparative analysis between their C++ implementations and those in another programming language was required. Consequently, we chose Julia, renowned for its suitability in numerical computation and high-performance computing tasks.

In the second phase, we were tasked with developing two parallel versions of the **Line Multiplication** algorithm.

## 2.1 Single-Core Algorithm Variants

### 2.1.1 Column Multiplication

This algorithm closely resembles the conventional manual matrix multiplication method. It involves multiplying each element of one matrix row by each corresponding element of the other matrix column. The initial C++ implementation was provided by the teaching staff, and we subsequently implemented it in Julia.

```
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;
    }
}
```

### 2.1.2 Line Multiplication

The Line Multiplication algorithm differs from Column Multiplication in the order of its nested for loops. Despite this seemingly minor distinction, it offers notable advantages in terms of efficiency, particularly by mitigating cache misses as we shall explore.

```
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];
        }
    }
}
```

### 2.1.3 Block Multiplication

The **Block Multiplication** algorithm breaks down matrices into smaller blocks for multiplication. The algorithm involves six nested for loops. The three outer loops handle the selection of submatrices for the computation, while the three inner loops execute the multiplication operations within these chosen submatrices.

This method offers performance benefits, particularly with large datasets. By dividing matrices into blocks, it reduces cache-related issues. Large matrices can exceed cache capacity, leading to frequent cache misses and slower computation. By using smaller blocks, this algorithm improves cache utilization, minimizing cache misses and enhancing performance.

```
for (int rbs = 0; rbs < m_ar; rbs += bkSize){
    for (int cbs = 0; cbs < m_ar; cbs += bkSize){
        for (int ibs = 0; ibs < m_ar; ibs += bkSize){
            for (int r = rbs; r < rbs + bkSize; r++){
                for (int c = cbs; c < cbs + bkSize; c++){
                    for (int i = ibs; i<ibs + bkSize; i++){
    phc[r*m_ar + i] +=  pha[r*m_ar + c] * phb[c*m_br + i];
                    }
                }
            }
        }
    }
}
```

## 2.2 Multi-Core Algorithm Variants

### 2.2.1 First Implementation

The first parallel implementation employs OpenMP directives for straightforward parallelization. Using the omp parallel for directive, the outer loop is parallelized, distributing loop iterations among multiple threads. This allows concurrent computation of matrix multiplication within the nested loops, enhancing performance.

```
int n_threads = omp_get_max_threads();

#pragma omp parallel for num_threads(n_threads) private(j,k)
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];
        }
    }
}
```

### 2.2.2 Second Implementation

The second implementation of the parallel matrix multiplication algorithm presents a significant departure from the first approach. While both methods utilize OpenMP directives for parallelization, the second variant introduces nested parallelism. In this approach, a parallel region is established, within which the innermost loop responsible for matrix multiplication is targeted by a pragma omp for directive. This nesting of parallelism facilitates finer workload distribution among threads, enhancing resource utilization and potentially improving overall performance.

```
int n_threads = omp_get_max_threads();

#pragma omp parallel private(i, j, k) num_threads(n_threads)
for(i=0; i<m_ar; i++){
    for( k=0; k<m_ar; k++){
        #pragma omp for
        for( j=0; j<m_br; j++){
            {
             phc[i*m_ar+j] += pha[i*m_ar+k] * phb[k*m_br+j];
            }
        }
    }
}
```

## 3 Performance metrics

To ensure consistency and reliability in our algorithm evaluations, we conducted all tests on a single machine, minimizing external influences on the results. Additionally, the machine remained connected to a power outlet throughout testing to prevent interruptions caused by CPU battery management. Below are the specifications of the machine used for our evaluations:

| **Component** | **Specification** |
|---|---|
| Processor | Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz |
| RAM | 12.0 GB (11.9 GB usable) |
| Operating System | Ubuntu 5.15.146.1-microsoft-standard-WSL2 |

**Table 1** Machine Specifications

## 4 Results and Analysis

## 5 Conclusions

Tables can be inserted via the normal table and tabular environment. To put footnotes inside tables you should use `\footnotetext[]{...}` tag. The footnote appears just below the table itself (refer Tables 2 and 3). For the corresponding footnotemark use `\footnotemark[...]`
The input format for the above table is as follows: