# Multi-core Programming
## Assignment 2

Ali Gholami

Department of Computer Engineering & Information Technology
Amirkabir University of Technology

*https://aligholamee.github.io*
*aligholami7596@gmail.com*

**Abstract**

OpenMP is an implementation of multithreading, a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them. The threads then run concurrently, with the runtime environment allocating threads to different processors. In this assignment, we are going to implement the parallelization of the matrix multiplication.

**Keywords.** *Heterogeneous Programming, OpenMP, C Programming, C++ Programming, Parallelization, Multi-thread Programming.*

# 1 Matrix Multiplication

## 1.1 What's the goal?

In this assignment, we'll be parallelizing the matrix multiplication using *OpenMP*. The goal is to speed up the matrix multiplication by implementing the parallelization in two axis (*1D* & *2D*). Below the serial code for the matrix multiplication. Sources for this assignment is available in the repository merged with this report.

```
void multiply(DataSet dataSet){
	int i, j, k, sum;
	for(i = 0; i < dataSet.n; i++){
		for(j = 0; j < dataSet.p; j++){
			sum = 0;
			for(k = 0; k < dataSet.m; k++){
				sum += dataSet.A[i * dataSet.m + k] * dataSet.B[k * dataSet.p + j];
			}
			dataSet.C[i * dataSet.p + j] = sum;
		}
	}
}
```

## 1.2 1D Parallelization

The following figures are provided from the problem description by *Dr. Ahmad Siavashi*. Each of the highlighted areas show a job for a thread. Figure 1.1 shows how the multiplication is done by each thread.
Assuming each *integer* as 4 bytes, we'll be filling the table 1.1 using the average time computed after 6 times of running the program. Note that in the dimensions of these matrices is assumed to be same

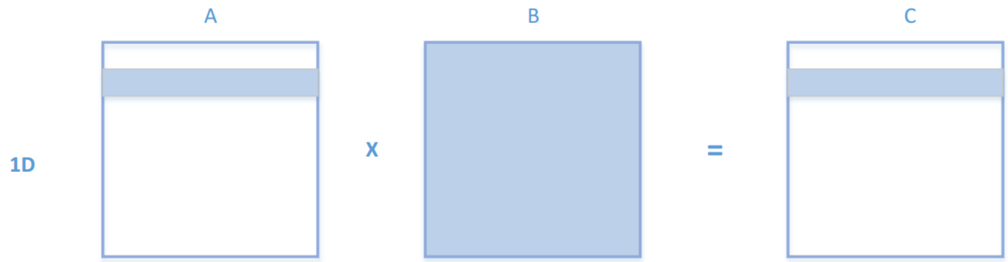


Figure 1.1: Matrix Multiplication Parallelization on Horizontal Axis.

in different axises and we are dealing with squared matrices. According to this assumption, each dimension can be computed as below:

- **1 KB**: $d = \sqrt{\frac{10^3}{4}} = 16$
- **10 KB**: $d = \sqrt{\frac{10^4}{4}} = 50$
- **100 KB**: $d = \sqrt{\frac{10^8}{4}} = 160$
- **1000 KB**: $d = \sqrt{\frac{10^9}{4}} = 505$

### 1.2.1 Serial Time

The average serial time for this multiplication is $0.239668$ seconds.

### 1.2.2 Parallelized For Loop

In order to make things a bit faster, we'll use *pragma* like so:

```
void multiply(DataSet dataSet){
	int i, j, k, sum;

	#pragma omp parallel for private(i)
	for(i = 0; i < dataSet.n; i++){
		for(j = 0; j < dataSet.p; j++){
			sum = 0;
			for(k = 0; k < dataSet.m; k++){
				sum += dataSet.A[i * dataSet.m + k] * dataSet.B[k * dataSet.p + j];
			}
			dataSet.C[i * dataSet.p + j] = sum;
		}
	}
}
```

| Total Size of Each Matrix | | | | | |
|---|---|---|---|---|---|
| Num of Threads | 1 KB | 10 KB | 100 KB | 1000 KB | Speedup |
| 1 | 2.39668 | 3.098899 | 56.76889 | 309.10345 | - |
| 2 | 0.32538 | 3.138752 | 31.108587 | 310.84936 | 2.3 |
| 4 | 0.35203 | 3.188132 | 32.936455 | 310.43297 | 1 |
| 8 | 0.339939 | 3.162757 | 31.435501 | 308.047638 | 1.1 |

Table 1.1: Results of 1-Dimensional Parallelization.

## 1.3 2D Parallelization

The results for the *2D* parallelization is given in the table 1.2.

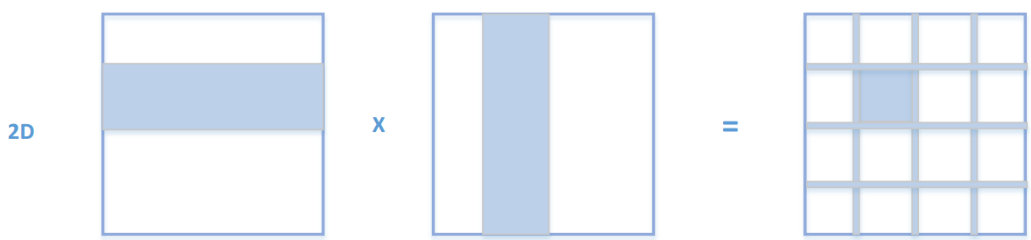


Figure 1.2: Matrix Multiplication Parallelization on Horizontal and Vertical Axis.

## 1.4 2D Parallelized For Loop

Each of the threads does the job in the highlighted area. In order to make things a bit faster, we'll use *pragma* like so:

```
void multiply(DataSet dataSet){
        int i, j, k, sum;

        #pragma omp parallel for private(i)
        for(i = 0; i < dataSet.n; i++){

                #pragma omp parallel for private(j)
                for(j = 0; j < dataSet.p; j++){
                        sum = 0;
                        for(k = 0; k < dataSet.m; k++){
                                sum += dataSet.A[i * dataSet.m + k] * dataSet.B[k * dataSet.p + j];
                        }
                        dataSet.C[i * dataSet.p + j] = sum;
                }
        }
}
```

| Total Size of Each Matrix | | | | | |
|---|---|---|---|---|---|
| Num of Threads | 1 KB | 10 KB | 100 KB | 1000 KB | Speedup |
| 1 | 0.335665 | 3.202322 | 30.957708 | 295.218567 | - |
| 2 | 0.279533 | 2.812544 | 30.707558 | 300.334747 | 1.2 |
| 4 | 0.268334 | 2.400637 | 30.366821 | 302.839111 | 1.04 |
| 8 | 0.294931 | 2.543862 | 30.529339 | 304.471771 | 0.95 |

Table 1.2: Results of 2-Dimensional Parallelization.