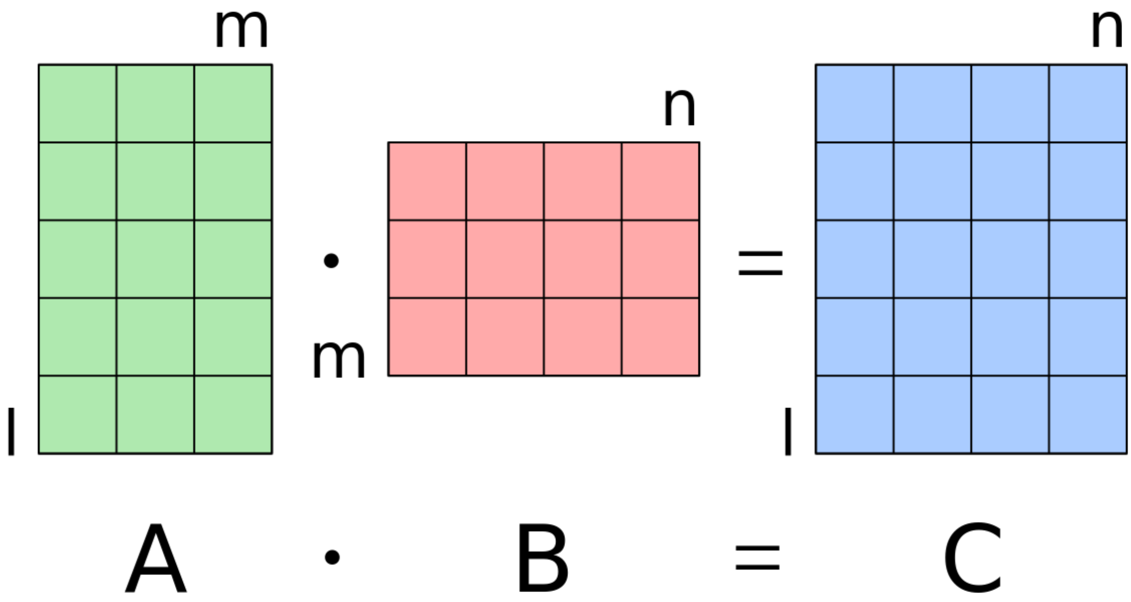
Operating System

Lab 2

Matrix Multiplication (Multi-Threading)



Name : Mohamed Mohamed Hawas

ID : 18011595

1. Objectives

- To get familiar with thread programming using the Pthread library.
- To better understand processes and threads.

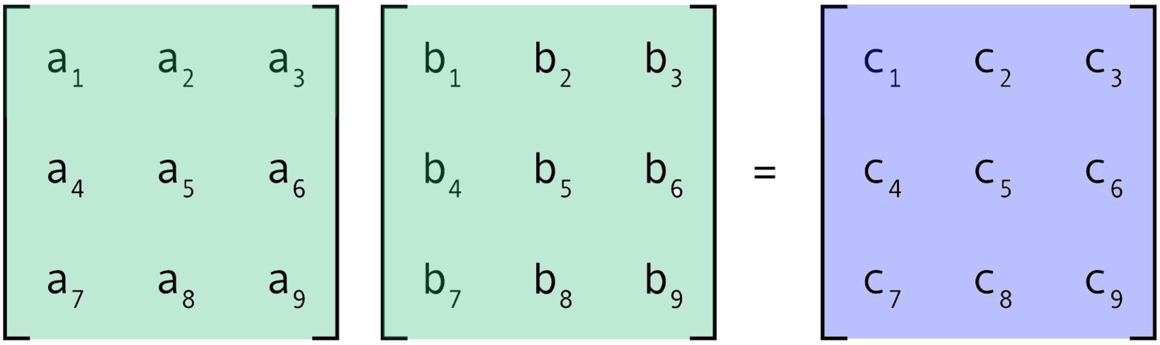
2. Overview

You are required to implement a multithreaded matrix multiplication program.

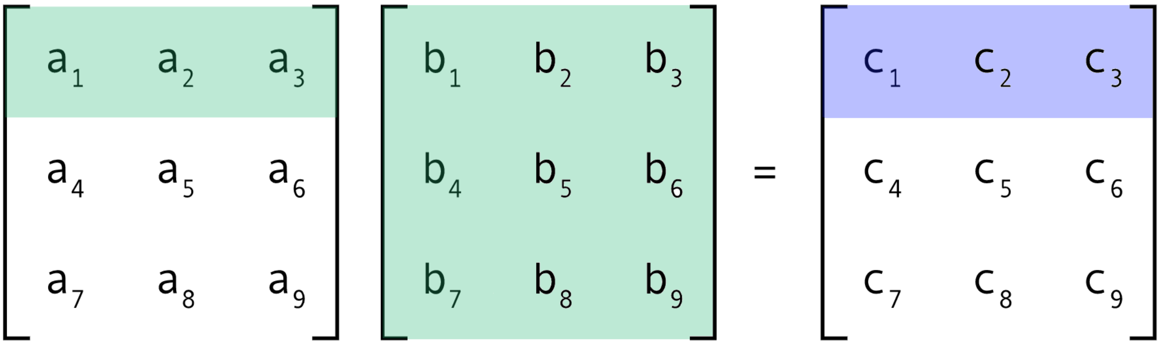
The input to the program is two matrixes A(x*y) and B(y*z) that are read from corresponding text files. The output is a matrix C(x*z) that is written to an output text file.

A parallelized version of matrix multiplication can be done using one of these three methods:

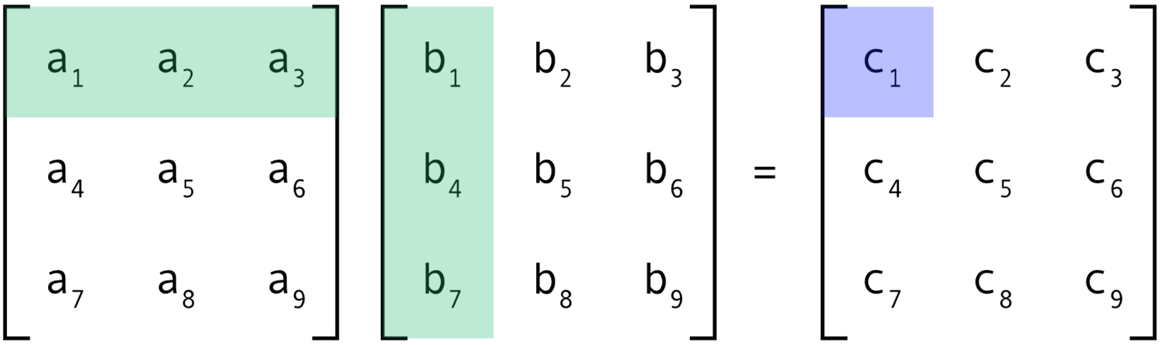
1. A thread computes the output C matrix i.e. without multi-threading. (A thread per matrix).



2. A thread computes each row in the output C matrix. (A thread per row).



3. A thread computes each element in the output C matrix. (A thread per element).



3. Code Organization

The program reads two matrices from files, multiplies them using three different methods, and writes the resulting matrices to output files. The program uses “pthreads” to parallelize the multiplication process. Here is an overview of how the code is organized:

1. The program defines a Matrix struct that holds the dimensions and elements of a matrix.
2. It defines three instances of Matrix: matrix_a, matrix_b, and matrix_c.
3. The program defines three functions to read the contents of the matrices from files and populate the Matrix structs. These functions are read_file, construct, and write_file.
4. The program defines three multiplication functions:
   - mult_per_matrix: multiplies matrix_a and matrix_b using the matrix multiplication algorithm and stores the result in matrix_c_per_matrix.

   - mult_per_row: multiplies matrix_a and matrix_b using the matrix multiplication algorithm and stores the result in matrix_c_per_row. This function is designed to be run by multiple threads, with each thread computing one row of the result matrix.

   - mult_per_element: multiplies matrix_a and matrix_b using the matrix multiplication algorithm and stores the result in matrix_c_per_element. This function is designed to be run by multiple threads, with each thread computing one element of the result matrix.

5. The main function reads the matrices from files using read_file, constructs the matrix_c matrices using construct, and creates the threads to perform the multiplications using pthread_create.

6. After the threads are created, the main function waits for all threads to complete using pthread_join.

7. Finally, the main function writes the resulting matrices to output files using write_file.

4. Code main functions

1. **construct(Matrix* matrix, int rows, int cols)**: A function that constructs a matrix with the given number of rows and columns.

2. **void* read_file(void* arg, Matrix* matrix)**: A thread function that reads the matrix data from a file and stores it in the given matrix.

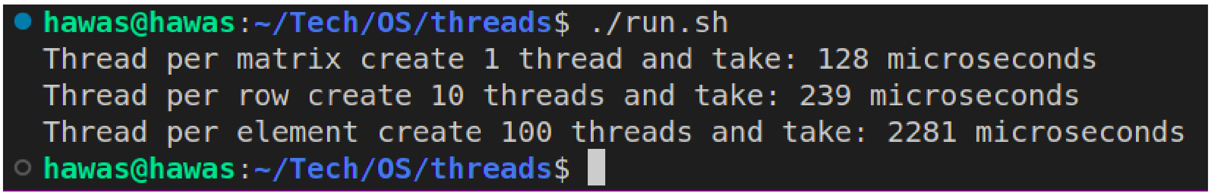
3. **void write_file(Matrix* matrix, char* filename)**: A function that writes the matrix data to a file.

4. **void* mult_per_matrix():** A thread function that multiplies two matrices (matrix_a and matrix_b) and stores the result in matrix_c_per_matrix.

5. **void* mult_per_row(void* arg)**: A thread function that multiplies two matrices (matrix_a and matrix_b) row by row and stores the result in matrix_c_per_row.

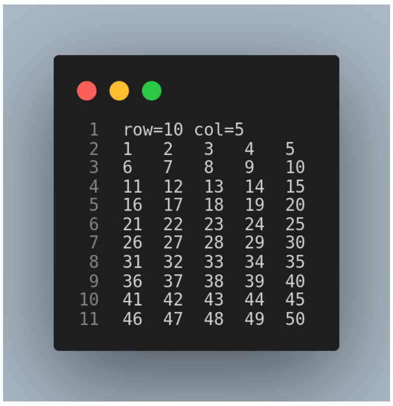
6. **void* mult_per_element(void* arg)**: A thread function that multiplies two matrices (matrix_a and matrix_b) element by element and stores the result in matrix_c_per_element.

3. Sample Runs and comparison



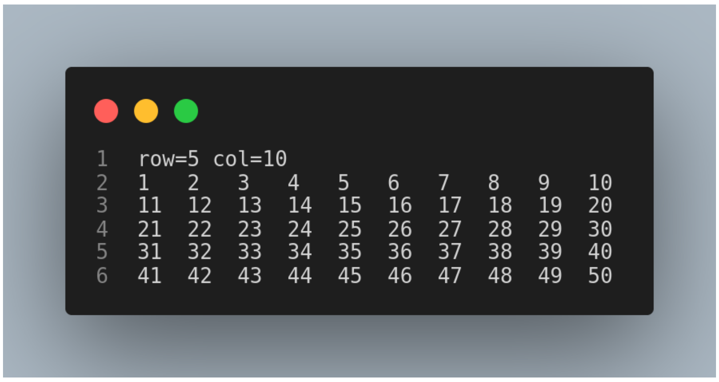
```
hawas@hawas:~/Tech/OS/threads$ ./run.sh
Thread per matrix create 1 thread and take: 128 microseconds
Thread per row create 10 threads and take: 239 microseconds
Thread per element create 100 threads and take: 2281 microseconds
hawas@hawas:~/Tech/OS/threads$ 
```

- **a_txt**



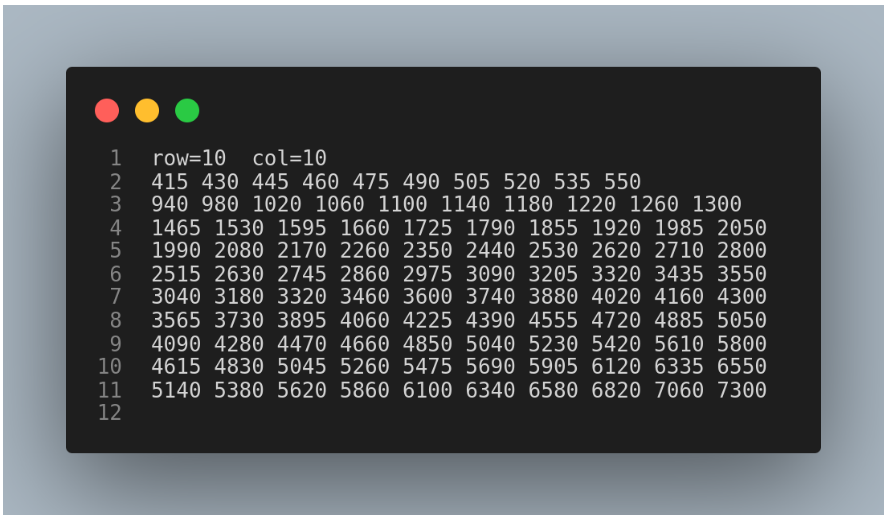
```
row=10 col=5
1   2   3   4   5
6   7   8   9   10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
26  27  28  29  30
31  32  33  34  35
36  37  38  39  40
41  42  43  44  45
46  47  48  49  50
```

- **b_txt**



```
row=5 col=10
1   2   3   4   5   6   7   8   9   10
11  12  13  14  15  16  17  18  19  20
21  22  23  24  25  26  27  28  29  30
31  32  33  34  35  36  37  38  39  40
41  42  43  44  45  46  47  48  49  50
```

- **c_txt (For all)**



```
row=10  col=10
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 3600 3740 3880 4020 4160 4300
3565 3730 3895 4060 4225 4390 4555 4720 4885 5050
4090 4280 4470 4660 4850 5040 5230 5420 5610 5800
4615 4830 5045 5260 5475 5690 5905 6120 6335 6550
5140 5380 5620 5860 6100 6340 6580 6820 7060 7300
```