Middle East Technical University Department of Computer Engineering

CENG 334

Introduction to Operating Systems

Spring 2022-2023

Homework 2 - Parallel Matrix Operation

Due date: 14 05 2023, Sunday, 23:59

1 Objective

This assignment aims to familiarize you with the development of multi-threaded applications and synchronization using C or C++. Your task is to implement two different matrix operations where multiple threads will perform them on the elements of four matrices and print the results. Towards this end, you will be using mutexes, semaphores and condition variables for synchronization between them.

Keywords— Thread, Semaphore, Mutex, Condition Variable

2 Problem Definition

In this homework, you are going to create a program that perform three operations on four matrices using threads without losing the integrity of the values. You will be given four matrices. First two matrices and the last two matrices will be added. The result of the first addition will be multiplied by the result of the second. These operations will be completed by threads. Each thread will be responsible for one row of results on all matrices (first two additions and final result). There are two rules to take note of. The first rule is that the operations must be thread-safe and run in parallel. The second rule is that there should be no unnecessary waiting. This means that one thread should not wait any longer than it needs to, especially not to wait for operations that does not affect its result. Here is an example of an addition operation:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} + \begin{bmatrix} 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \end{bmatrix} = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 11 & 13 & 15 & 17 \\ 19 & 21 & 23 & 25 \\ 27 & 29 & 31 & 33 \end{bmatrix}$$

Each row should be completed by a separate thread, with the final matrix resulting from the addition of the individual rows. The highlighted cells show the results of each row addition, which are then combined to form the final matrix. In this example, the yellow cells represent the result of the first row addition, the remaining follow the same idea.

Here is an example of a multiplication operation:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \\ 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \end{bmatrix} = \begin{bmatrix} 100 & 110 & 120 & 130 \\ 228 & 254 & 280 & 306 \\ 356 & 398 & 440 & 482 \\ 484 & 542 & 600 & 658 \end{bmatrix}$$

Each row in the resulting matrix can be computed by a separate thread, with the final matrix resulting from the multiplication of the individual elements. The highlighted cells show the results of each row multiplication, which are then combined to form the final matrix.

In this example, the yellow cells represent the result of the multiplication of the first row of the first matrix and all of the columns of the second matrix, the remaining colors follow the same idea.

3 Implementation Details

You will be given four matrices. Two $N \times M$ matrices and two $M \times K$ matrix. First two matrices will be added together to form the matrix J and the last two matrices will be added together to form L. Finally J and L will multiplied together to get the final result. To this end, you will create $N + M + N$ number of threads. The first N threads will be responsible for the first addition operation. They each will be responsible for the addition of a single row. The second M threads will be responsible for the second addition operation. Similarly, they will be responsible for a single row. The final N threads will be responsible for the final multiplication operation. They will multiply one row of the first result from the matrix J with every column of the matrix L to fill a single row. In short, they will be responsible for the calculation of a single row in the final result.

If we consider the matrices are named A, B, C, and D all of the operations would correspond to the following formula:

$$A + B = J$$
$$C + D = L$$
$$J \times L = R$$

where R is the final result. Please note that $\times$ is used for matrix multiplication.

All threads should start at the beginning and the threads responsible for the final result should wait until the row they need from the matrix J and each column they need from the matrix L is available. They should not wait for all of the matrix to be ready, just their row in J and each column in L. They should calculate each element in the row as the row and column they require for multiplication is ready. During their execution, the threads should print each element they calculated using the provided function. After each thread finishes its execution, your program should join them and not leave them hanging. The threads should also **NOT** be detached. When the final matrix is ready, your program should print the result and exit.

4 Input Format

You will read the input from that standard input. The input consists of four matrices with their dimensions preceding them:

The first matrix is an $r_1 \times c_1$ matrix A, where r_1 is the number of rows and c_1 is the number of columns. The elements of the matrix are represented as integers and are provided row-wise in the following format:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,c_1} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,c_1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_1,1} & a_{r_1,2} & \cdots & a_{r_1,c_1} \end{bmatrix}$$

The second matrix is an $r_2 \times c_2$ matrix B, where r_2 is the number of rows and c_2 is the number of columns. The elements of the matrix are represented as integers and are provided row-wise in the following format:

$$\begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,c_2} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,c_2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{r_2,1} & b_{r_2,2} & \cdots & b_{r_2,c_2} \end{bmatrix}$$

The third matrix is an $r_3 \times c_3$ matrix C, where r_3 is the number of rows and c_3 is the number of columns. The elements of the matrix are represented as integers and are provided row-wise in the following format:

$$\begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,c_3} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,c_3} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r_3,1} & c_{r_3,2} & \cdots & c_{r_3,c_3} \end{bmatrix}$$

The fourth matrix is an $r_4 \times c_4$ matrix D, where r_4 is the number of rows and c_4 is the number of columns. The elements of the matrix are represented as integers and are provided row-wise in the following format:

$$\begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,c_4} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,c_4} \\ \vdots & \vdots & \ddots & \vdots \\ d_{r_4,1} & d_{r_4,2} & \cdots & d_{r_4,c_4} \end{bmatrix}$$

Please note that the first two matrices A, B and the last two matrices C, D must have identical size and the number of columns of A and B must be equal to the number of rows of C and D in order to perform the addition and multiplication operations. This means that $c_1 = c_2 = r_3 = r_4$. Additional reminder that, the three matrices will only contain integers. There will **NOT** be any floating point values.

4.1 Example Input

```
4 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
4 4
2 0 0 1
0 2 0 0
0 0 2 0
0 0 0 2
4 3
1 2 3
4 5 6
7 8 9
10 11 12
4 3
0 0 3
0 5 0
7 0 0
1 1 1
```

In this example, we have:

- A 4×4 matrix A, where $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$
- A 4×4 matrix B, where $B = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$
- A 4×3 matrix C, where $C = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$
- A 4×3 matrix D, where $D = \begin{bmatrix} 0 & 0 & 3 \\ 0 & 5 & 0 \\ 7 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

5 Output Format

For output you will use a provided function `hw2_write_output` to indicate when a matrix element is calculated. It should be called inside the threads when the calculation of an element is complete. It is given in the files provided to you. There is also `hw2_init_output` function that needs to be called at the start of your main program only once. It is used to record the start time of your program. The functions are given below:

```
void hw2_init_output(void);

void hw2_write_output(unsigned matrix_id, unsigned row, unsigned col, int value);
```

The first argument is the matrix id. It is used to indicate which of the results that have been calculated. It should be 0 for the result of the first addition operation, 1 for the result of the second addition operation, and 2 for the final result. The `row` and `col` parameters indicate the location of the element that has been calculated and `value` should be the element. After the calculation of the final matrix has completed, your program should print the final result it. The format of the matrix is:

```
<Row 1 of result matrix>
<Row 2 of result matrix>
...
<Row n of result matrix>
```

And for the given example, the output would be:

```
108 110 122
223 242 245
363 366 387
477 502 531
```

6 Specifications

- Your code must be written in C or C++ on Linux. No other platforms and languages will be accepted.
- You are allowed to use `pthread.h`, `semaphore.h` libraries for the threads, semaphores, condition variables and mutexes. Your solution should not employ busy wait. Your Makefile should not contain any library flag other than `-lpthread`. It will be separately controlled.
- Submissions will be evaluated with black box technique with different inputs.
- There will be penalty for bad solutions. Non terminating (deadlocked or otherwise) solutions will get zero from the corresponding input.
- Your submission will be evaluated on lab computers (ineks).
- Everything you submit should be your own work. Usage of binary source files and codes found on internet is strictly forbidden.
- Please follow the course page on ODTUClass for updates and clarifications.
- Please ask your questions related to homework through ODTUClass instead of emailing directly to teaching assistants if your question does not contain solution or code.

7 Submission

Submission will be done via ODTUClass. Create a tar.gz file named `hw2.tar.gz` that contains all your source code files together with your Makefile. Your tar file should not contain any folders. Your code should be able to compile and your executable should run using this command sequence.

```
> tar -xf hw2.tar.gz
> make all
> ./hw2
```

If there is a mistake in any of the 3 steps mentioned above, you will lose 10 points.