Lab 2
OS

Name: البدري محمد السيد البدري
ID: 20010329

- Problem Statement:

Implement a multi-threaded matrix multiplication program, there are three cases:

1. A thread per matrix.
2. A thread per row.
3. A thread per element.

- Organization:

The program is composed of 7 important function including the main function:

1. void Take_Input(char *ar[]).
2. void *CaseA().
3. void *CaseB(void *arg).
4. void *CaseC(void *arg).
5. Void print_Output(double mat[][MAX_LENGTH],char *methodName, __u_long t, __u_long ts).
6. void free_Resources(matrix_minimized *m, matrix_minimized **m3).
7. int main(int argc, char *argv[]).

- Functions’ Description:

1. void Take_Input(char *ar[]):

   This function takes the name of the files that contains the input arrays, then store the array elements and number of rows and columns in global variables.

2. void *CaseA():

   This function implements matrix multiplication, which is used by a single thread to compute the whole new matrix.

3. void *CaseB(void *arg):

   this function implement matrix multiplication, which is used by number of threads, where each thread compute a single row in the new matrix. The parameter includes a struct contains the row number.

4. void *CaseC(void *arg):

   This function implements matrix multiplication, used by multiple threads, where each thread computes each single element in the new matrix. The parameter includes a struct contains the row and column number.

5. Void print_Output(double mat[][MAX_LENGTH],char *methodName, __u_long t, __u_long ts):

   This method is responsible for printing the content of the new matrix produced from each method(by matrix, by row, by

element), as well as the number of rows , columns , threads and execution time, it takes the matrix as a parameter , method name and the execution time.

6. void free_Resources(matrix_minimized *m, matrix_minimized **m3):

   This method is used to free the resources located in the dynamic heap by the structs.

7. int main(int argc, char *argv[]):
   The main function contains the thread creation and calling all the above functions.

- How to compile and run code:

1. Open terminal and cd to project file.
2. Write “make” in the terminal.
3. Write “./matMultp a b c” where “a” and “b” are the files that contain the input arrays, while “c” is the output file.

- Sample runs:

1.

a

```
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

```
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_per_matrix

```
Method: Case A (per matrix)
row = 10 col = 10
threads = 1 time in microseconds= 62us, time in seconds = 0s
415.000   430.000   445.000   460.000   475.000   490.000   505.000   520.000   535.000   550.000
940.000   980.000   1020.000   1060.000   1100.000   1140.000   1180.000   1220.000   1260.000   1300.000
1465.000   1530.000   1595.000   1660.000   1725.000   1790.000   1855.000   1920.000   1985.000   2050.000
1990.000   2080.000   2170.000   2260.000   2350.000   2440.000   2530.000   2620.000   2710.000   2800.000
2515.000   2630.000   2745.000   2860.000   2975.000   3090.000   3205.000   3320.000   3435.000   3550.000
3040.000   3180.000   3320.000   3460.000   3600.000   3740.000   3880.000   4020.000   4160.000   4300.000
3565.000   3730.000   3895.000   4060.000   4225.000   4390.000   4555.000   4720.000   4885.000   5050.000
4090.000   4280.000   4470.000   4660.000   4850.000   5040.000   5230.000   5420.000   5610.000   5800.000
4615.000   4830.000   5045.000   5260.000   5475.000   5690.000   5905.000   6120.000   6335.000   6550.000
5140.000   5380.000   5620.000   5860.000   6100.000   6340.000   6580.000   6820.000   7060.000   7300.000
```

$ c_per_row

```
Method: Case B (per row)
row = 10 col = 10
threads = 10 time in microseconds= 184us, time in seconds = 0s
415.000   430.000   445.000   460.000   475.000   490.000   505.000   520.000   535.000   550.000
940.000   980.000   1020.000   1060.000   1100.000   1140.000   1180.000   1220.000   1260.000   1300.000
1465.000   1530.000   1595.000   1660.000   1725.000   1790.000   1855.000   1920.000   1985.000   2050.000
1990.000   2080.000   2170.000   2260.000   2350.000   2440.000   2530.000   2620.000   2710.000   2800.000
2515.000   2630.000   2745.000   2860.000   2975.000   3090.000   3205.000   3320.000   3435.000   3550.000
3040.000   3180.000   3320.000   3460.000   3600.000   3740.000   3880.000   4020.000   4160.000   4300.000
3565.000   3730.000   3895.000   4060.000   4225.000   4390.000   4555.000   4720.000   4885.000   5050.000
4090.000   4280.000   4470.000   4660.000   4850.000   5040.000   5230.000   5420.000   5610.000   5800.000
4615.000   4830.000   5045.000   5260.000   5475.000   5690.000   5905.000   6120.000   6335.000   6550.000
5140.000   5380.000   5620.000   5860.000   6100.000   6340.000   6580.000   6820.000   7060.000   7300.000
```

$ c_per_element

```
Method: Case C (per element)
row = 10 col = 10
threads = 100 time in microseconds= 1486us, time in seconds = 0s
415.000   430.000   445.000   460.000   475.000   490.000   505.000   520.000   535.000   550.000
940.000   980.000   1020.000   1060.000   1100.000   1140.000   1180.000   1220.000   1260.000   1300.000
1465.000   1530.000   1595.000   1660.000   1725.000   1790.000   1855.000   1920.000   1985.000   2050.000
1990.000   2080.000   2170.000   2260.000   2350.000   2440.000   2530.000   2620.000   2710.000   2800.000
2515.000   2630.000   2745.000   2860.000   2975.000   3090.000   3205.000   3320.000   3435.000   3550.000
3040.000   3180.000   3320.000   3460.000   3600.000   3740.000   3880.000   4020.000   4160.000   4300.000
3565.000   3730.000   3895.000   4060.000   4225.000   4390.000   4555.000   4720.000   4885.000   5050.000
4090.000   4280.000   4470.000   4660.000   4850.000   5040.000   5230.000   5420.000   5610.000   5800.000
4615.000   4830.000   5045.000   5260.000   5475.000   5690.000   5905.000   6120.000   6335.000   6550.000
5140.000   5380.000   5620.000   5860.000   6100.000   6340.000   6580.000   6820.000   7060.000   7300.000
```

2.

x

```
row=3 col=5
1   -2  3   4   5
1   2   -3  4   5
-1  2   3   4   5
```

y

```
row=5 col=4
-1  2   3   4
1   -2  3   4
1   2   -3  4
1   2   3   -4
-1  -2  -3  -4
```

d_per_matrix

```
Method: Case A (per matrix)
row = 3 col = 4
threads = 1 time in microseconds= 63us, time in seconds = 0s
-1.000   10.000   -15.000   -28.000
-3.000   -10.000   15.000   -36.000
5.000   -2.000   -9.000   -20.000
```

d_per_row

```
Method: Case B (per row)
row = 3 col = 4
threads = 3 time in microseconds= 73us, time in seconds = 0s
-1.000   10.000   -15.000   -28.000
-3.000   -10.000   15.000   -36.000
5.000   -2.000   -9.000   -20.000
```

d_per_element

```
Method: Case C (per element)
row = 3 col = 4
threads = 12 time in microseconds= 272us, time in seconds = 0s
-1.000   10.000   -15.000   -28.000
-3.000   -10.000   15.000   -36.000
5.000   -2.000   -9.000   -20.000
```

3.

z

```
row=5 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
```

w

```
row=5 col=4
1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  16
17  18  19  20
```

e_per_matrix

```
Method: Case A (per matrix)
row = 5 col = 4
threads = 1 time in microseconds= 37us, time in seconds = 0s
175.000   190.000   205.000   220.000
400.000   440.000   480.000   520.000
625.000   690.000   755.000   820.000
850.000   940.000   1030.000   1120.000
1075.000   1190.000   1305.000   1420.000
```

e_per_row

```
Method: Case B (per row)
row = 5 col = 4
threads = 5 time in microseconds= 127us, time in seconds = 0s
175.000   190.000   205.000   220.000
400.000   440.000   480.000   520.000
625.000   690.000   755.000   820.000
850.000   940.000   1030.000   1120.000
1075.000   1190.000   1305.000   1420.000
```

e_per_element

```
Method: Case C (per element)
row = 5 col = 4
threads = 20 time in microseconds= 338us, time in seconds = 0s
175.000   190.000   205.000   220.000
400.000   440.000   480.000   520.000
625.000   690.000   755.000   820.000
850.000   940.000   1030.000   1120.000
1075.000   1190.000   1305.000   1420.000
```

- Comparison between the three methods of matrix multiplication:

	Method 1	Method 2	Method 3
Test 1	62us	184us	1486us
Test 2	63us	73us	272us
Test 3	37us	127us	338us

From the previous table we can observe that:

method 1 has the least time, while method 2 in the middle, and method 3 has the highest execution time

We can conclude that:

increasing number of threads increase the execution time as the thread creation takes much time, that’s why we need to know when to use multi-threading, to avoid wasting the processor time.