Name: Mahmoud Mohammed Fathallah abdulrazik

ID: 19016582

Lab #2 report

Code organization:

- The main function first uses the read_data_mat() function to read the input matrices, then multiplies them using mult_matrix_onethread() function and then creates threads with the number of rows of first matrix where every thread is responsible for calculating a row using mult_matrix_element() function, then it creates threads with the #rows in first matrix * #columns in second matrix where every thread is responsible for calculating an element using mult_matrix_element() function, then the resulting matrices are written into files using write_data_res() function. At the end the program prints out the execution time of the three methods.

Code Main functions:

- read_data_mat(filename, num, matrix): this function takes as parameters: a file name to be opened, a number 1 or 2 to specify whether the matrix is the first or second one, and a matrix, it reads the input matrix from the file into the matrix.
- write_data_res(filename, num, matrix): this function takes as parameters: a file name to be created, a number 1 or 2 or 3 to specify whether the matrix is the first or second or third one, and a matrix, it writes the resulting matrix to the file.
- mult_matrix_onethread(): this function multiplies the global matrices mat1 and mat2 and writes the result to res1 matrix.

- mult_matrix_row(row): this function takes the row index as a parameter and multiplies the specified row of mat1 by all columns of mat2 and writes the result to res2.
- mult_matrix_element(roc): this function takes an integer array containing the two indices I,J of an element and multiplies the ith row of mat1 by the jth column of mat2 and stores the result in res3.

Compiling and running the code:

- To compile the code open the project folder in a terminal and enter the command make.
- To run the code after compilation enter the command "./matMultp x y z" where x is the name of the txt file containing the first matrix, y is the name of the file containing the second matrix and z is the first part of the names of the three output files of the code.
- Note that if x y z where not entered the default values will be a b c.

Sample runs:

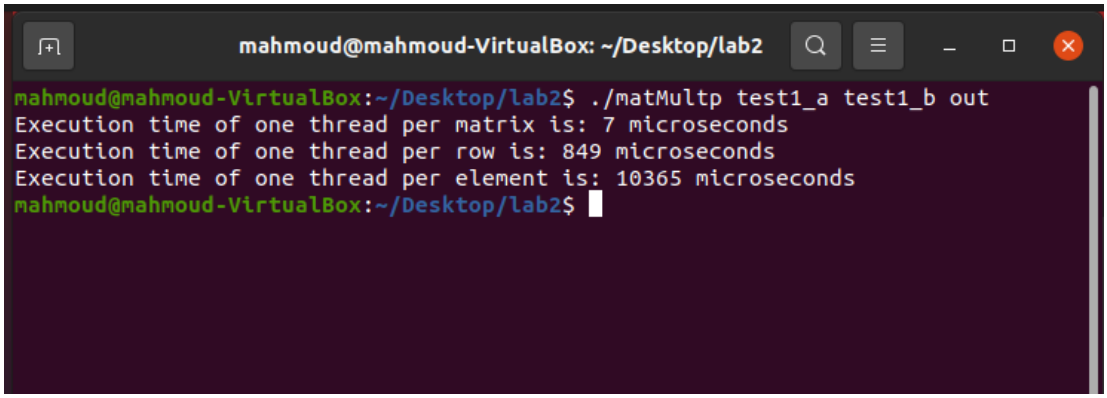
- First run:-
  First matrix:

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

Second matrix.

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

Run.



Output matrices.

```
1 method: one thread per element.
2 row=10 col=10
3 415   430   445   460   475   490   505   520   535   550
4 940   980   1020  1060  1100  1140  1180  1220  1260  1300
5 1465  1530  1595  1660  1725  1790  1855  1920  1985  2050
6 1990  2080  2170  2260  2350  2440  2530  2620  2710  2800
7 2515  2630  2745  2860  2975  3090  3205  3320  3435  3550
8 3040  3180  3320  3460  3600  3740  3880  4020  4160  4300
9 3565  3730  3895  4060  4225  4390  4555  4720  4885  5050
10 4090  4280  4470  4660  4850  5040  5230  5420  5610  5800
11 4615  4830  5045  5260  5475  5690  5905  6120  6335  6550
12 5140  5380  5620  5860  6100  6340  6580  6820  7060  7300
```

```
method: one thread per matrix.
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
```

```
method: one thread per row.
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
```

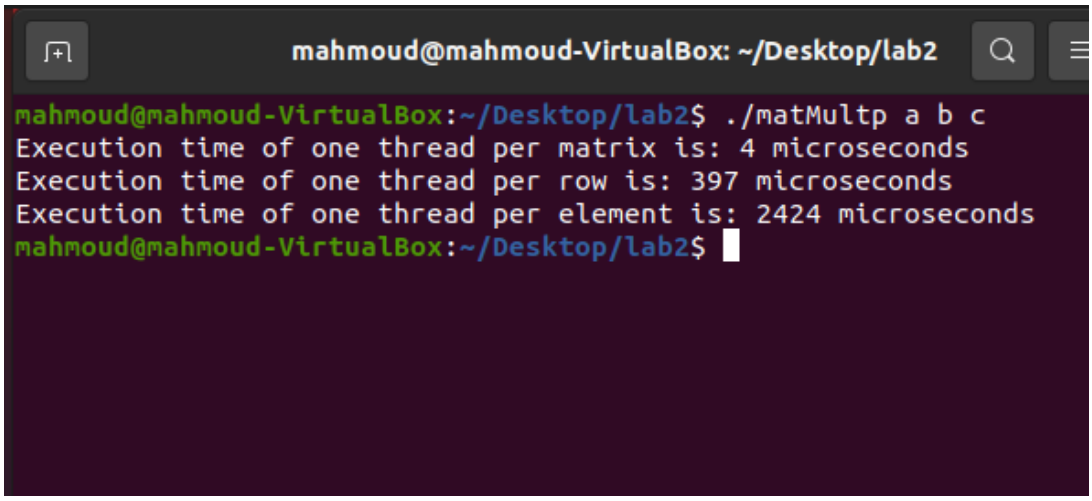
- Second run:-
  First matrix.

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

Second matrix.

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

Run.



Output matrices.

```
method: one thread per element.
row=6 col=6
55   6   107   115   182   33
161   117   252   276   307   30
347   372   525   597   320   -101
463   501   686   778   431   -120
144   37   291   320   329   -123
-299   -412   -317   -351   -61   160
```

```
method: one thread per matrix.
row=6 col=6
55   6   107   115   182   33
161   117   252   276   307   30
347   372   525   597   320   -101
463   501   686   778   431   -120
144   37   291   320   329   -123
-299   -412   -317   -351   -61   160
```

```
1 method: one thread per row.
2 row=6 col=6
3 55  6  107  115  182  33
4 161  117  252  276  307  30
5 347  372  525  597  320  -101
6 463  501  686  778  431  -120
7 144  37  291  320  329  -123
8 -299  -412  -317  -351  -61  160
```

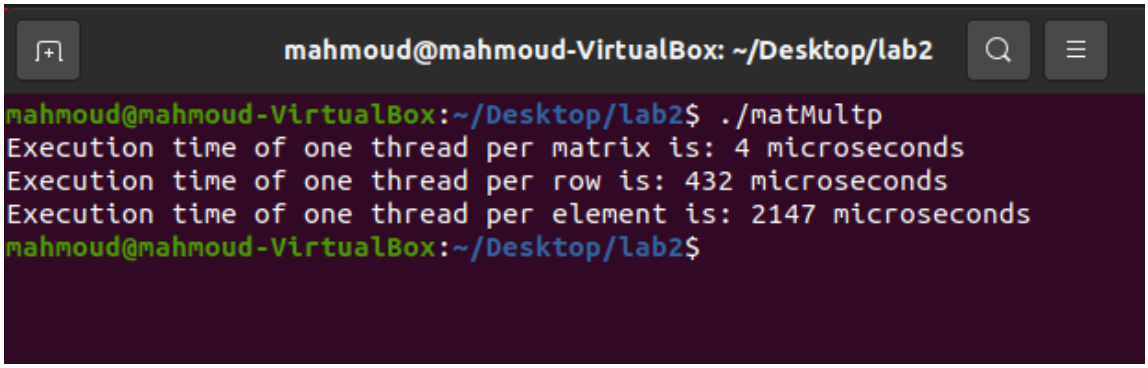
- Third run:
  First matrix.

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

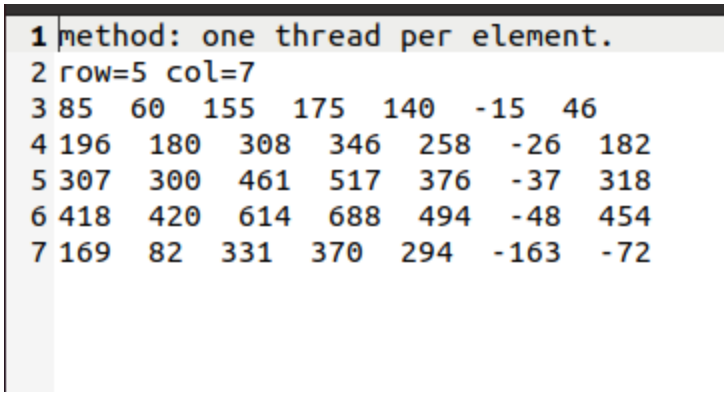
Second matrix.

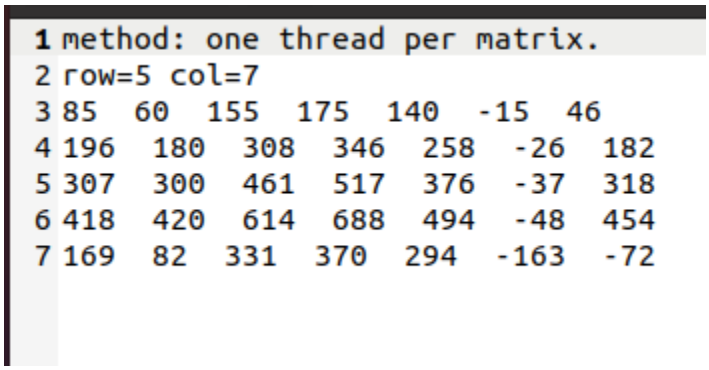
```
1 row=5 col=7
2 1      2      3      4      5      6      5
3 5      6      7      8      6      7      14
4 9      10     11     12     7      -8     15
5 13     14     15     16     8      -9     2
6 -1     -8     9      11     14     5      -8
```

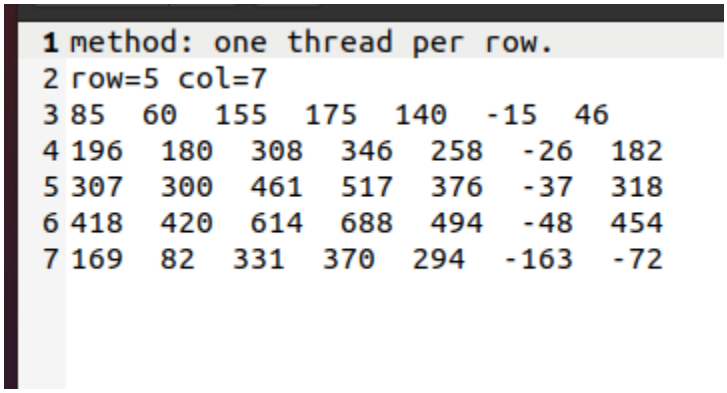
Run.



Output matrices.







Comparison:

Table:

Run	Point of comparison	Thread per matrix	Thread per row	Thread per element
First10*5 Second5*10	time taken	7	849	10365
	#of threads	1	10	100
First6*6 Second6*6	time taken	4	397	2424
	#of threads	1	6	36
First5*5 Second5*7	time taken	4	432	2147
	#of threads	1	5	35
First3*5 Second5*4	time taken	4	300	998
	#of threads	1	3	12

Conclusion:

- We can conclude from the above that the number of threads created for third method is greater than second method and the second method greater than first method.
- We can also conclude that the execution time for first method is the least and the second is greater and third is greater than both, this is due to the overhead of creating the threads.

Graph:

The following graph represents the relationship between number of threads and execution time taken.