Lab 2
Operating System

Sama Tarek Anwar Zayed

20010698

How my code is organized.

1.readfilea is a function to read first matrix in first file and assign number of rows and columns and this matrix to its global variables

2.readfileb is a function to read second matrix in second file and assign number of rows and columns and this matrix to its global variables

3.threadpermatrix is a function to multiply two matrices using just one thread

4.threadperrow is function to take index of row and get the row result

5.threadsperrow is a function to create threads of threadperrow equal to number of row

6.threadperelement is a function that take the index of row and column (in form of struct) of element to get it after multiplying .

7.threadsperelement is a function that creates multiple threads of threadperelement in order to get the each element after multiplying

8.running function is a function where the previous functions are called and where the time of each method is calculated and the resulted matrices are written in the output files

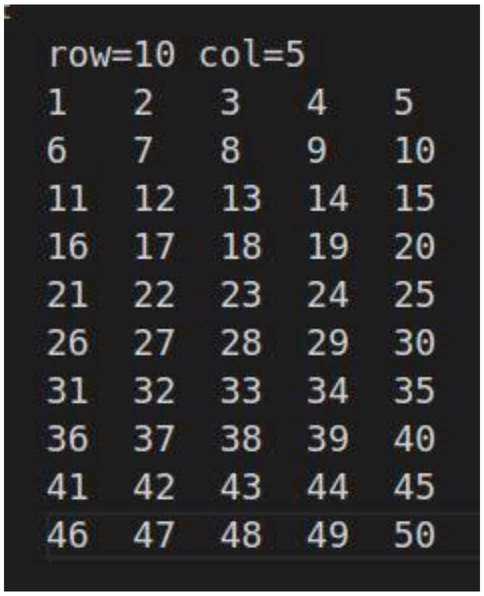
My code main functions.

Main function takes args and if this arg is less than 4 arg then the program will read from the default files if not then the program will read from the given files and then readfile functions are called then running function is called.

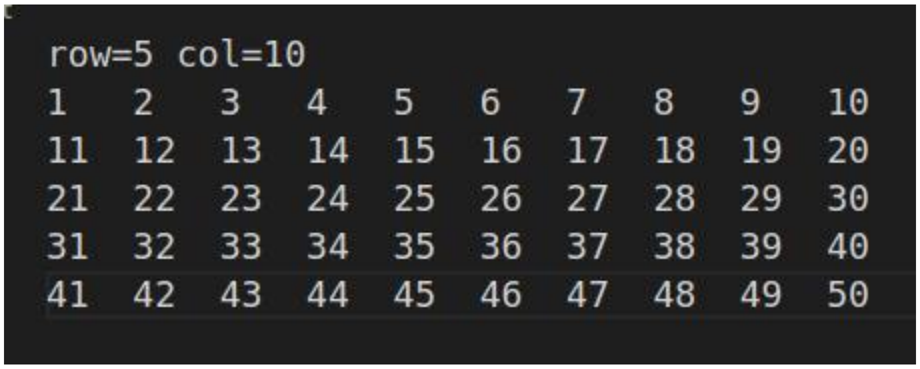
How to compile and run your code.

To run code with default files just run it if you want to specify the files then you should write in terminal gcc main.c -o matMultp then ./matMultp a b c ,a b c are the names of files you want to work on.

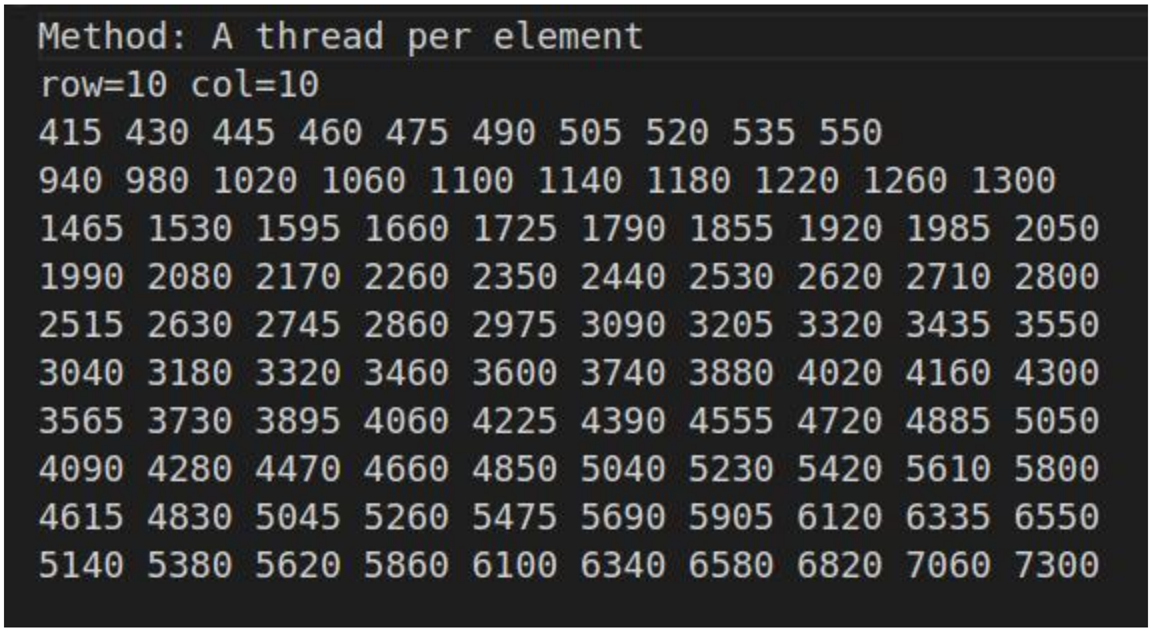
Sample runs.



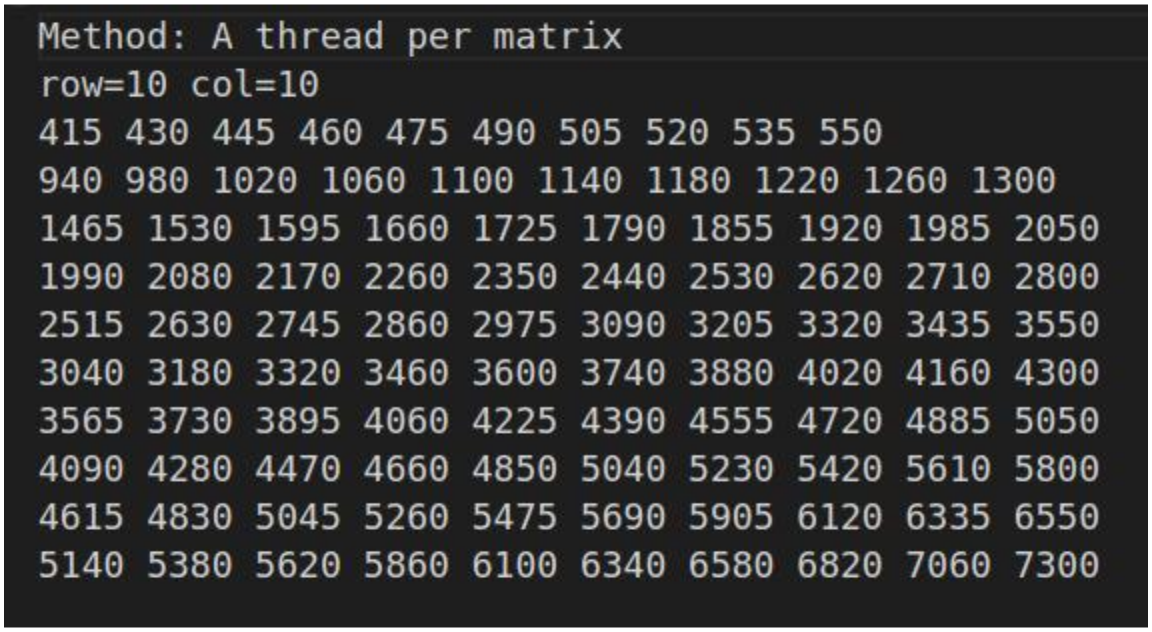
```
row=10 col=5
1   2   3   4   5
6   7   8   9   10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
26  27  28  29  30
31  32  33  34  35
36  37  38  39  40
41  42  43  44  45
46  47  48  49  50
```



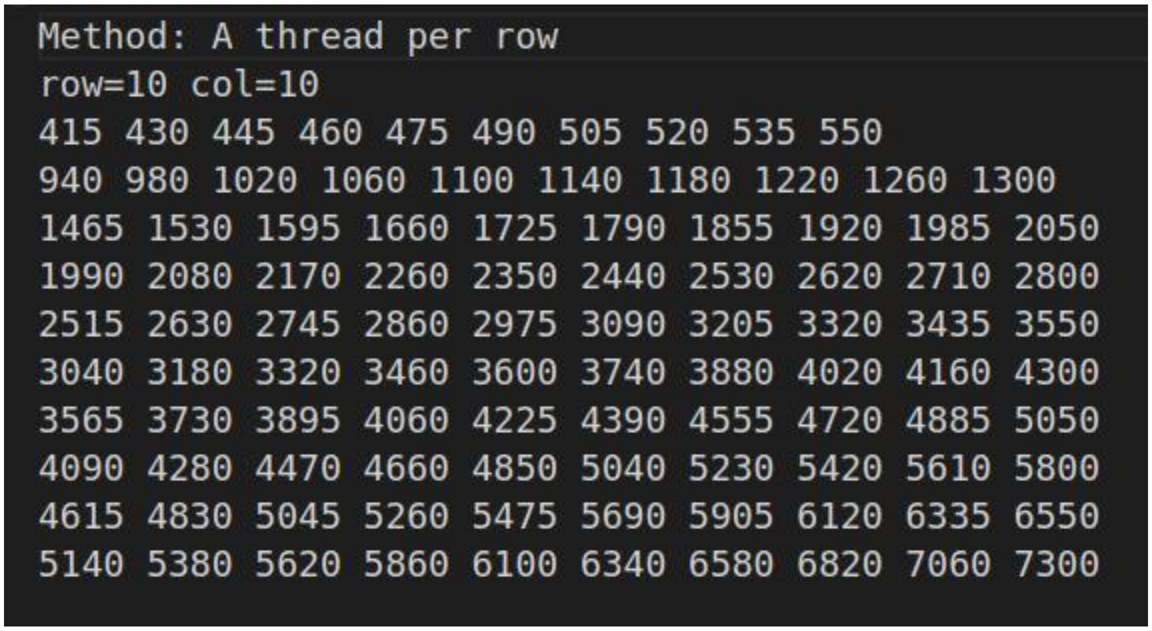
```
row=5 col=10
1    2    3    4    5    6    7    8    9    10
11   12   13   14   15   16   17   18   19   20
21   22   23   24   25   26   27   28   29   30
31   32   33   34   35   36   37   38   39   40
41   42   43   44   45   46   47   48   49   50
```



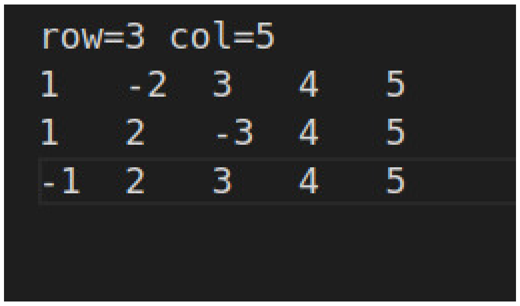
```
Method: A thread per element
row=10 col=10
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 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
```



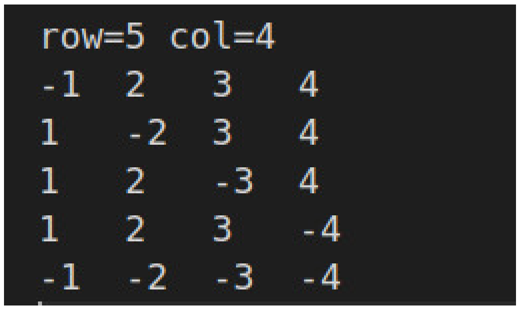
```
Method: A thread per matrix
row=10 col=10
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 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
```



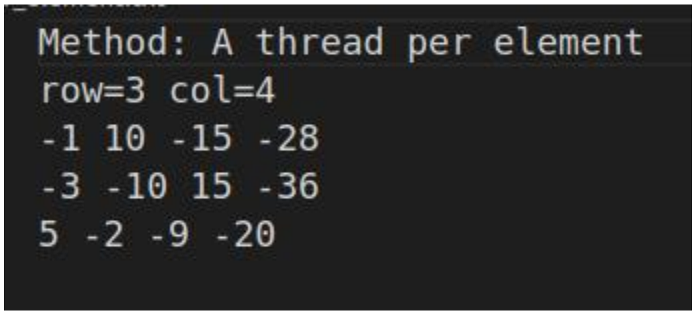
```
Method: A thread per row
row=10 col=10
415 430 445 460 475 490 505 520 535 550
940 980 1020 1060 1100 1140 1180 1220 1260 1300
1465 1530 1595 1660 1725 1790 1855 1920 1985 2050
1990 2080 2170 2260 2350 2440 2530 2620 2710 2800
2515 2630 2745 2860 2975 3090 3205 3320 3435 3550
3040 3180 3320 3460 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
```



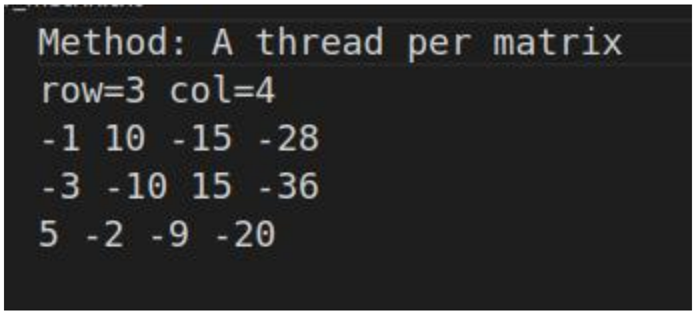
```
row=3 col=5
1   -2  3   4   5
1   2   -3  4   5
-1  2   3   4   5
```



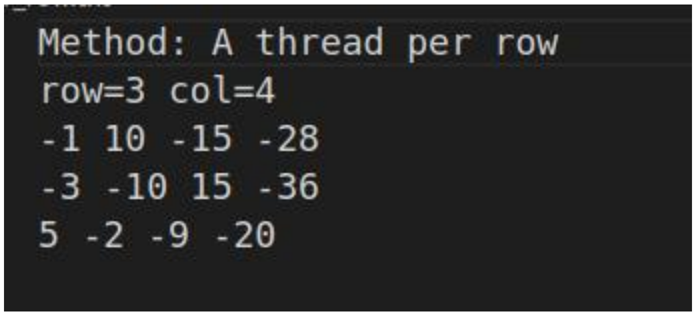
```
row=5 col=4
-1  2   3   4
1   -2  3   4
1   2   -3  4
1   2   3   -4
-1  -2  -3  -4
```



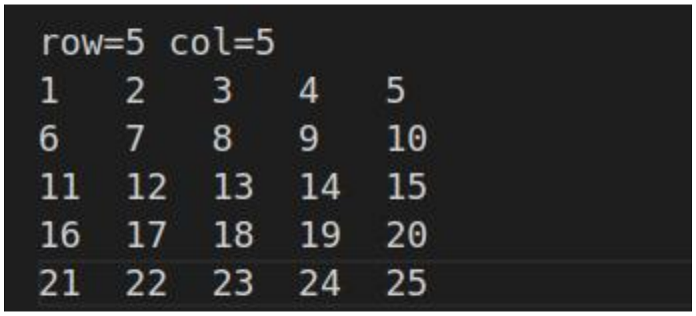
```
Method: A thread per element
row=3 col=4
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
```



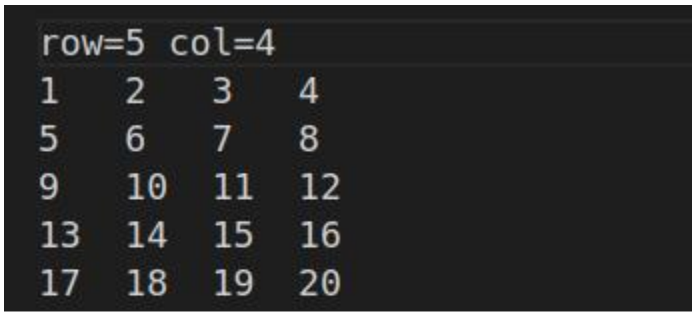
```
Method: A thread per matrix
row=3 col=4
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
```



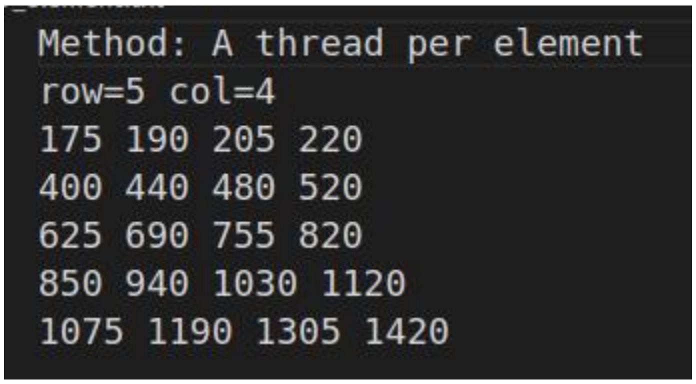
```
Method: A thread per row
row=3 col=4
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
```



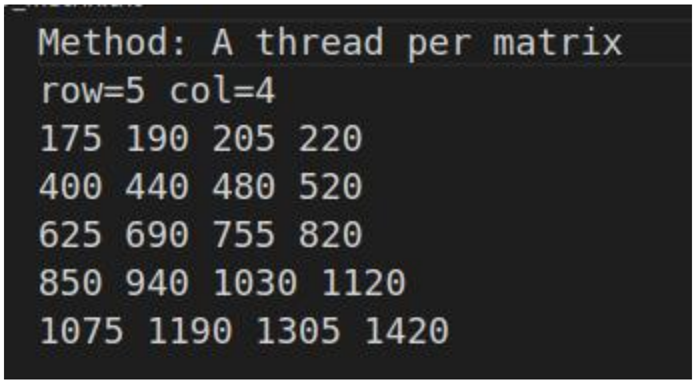
```
row=5 col=5
1   2   3   4   5
6   7   8   9   10
11  12  13  14  15
16  17  18  19  20
21  22  23  24  25
```



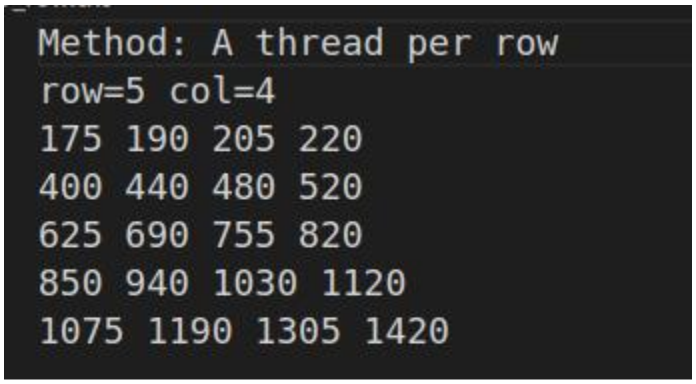
```
row=5 col=4
1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  16
17  18  19  20
```



```
Method: A thread per element
row=5 col=4
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120
1075 1190 1305 1420
```



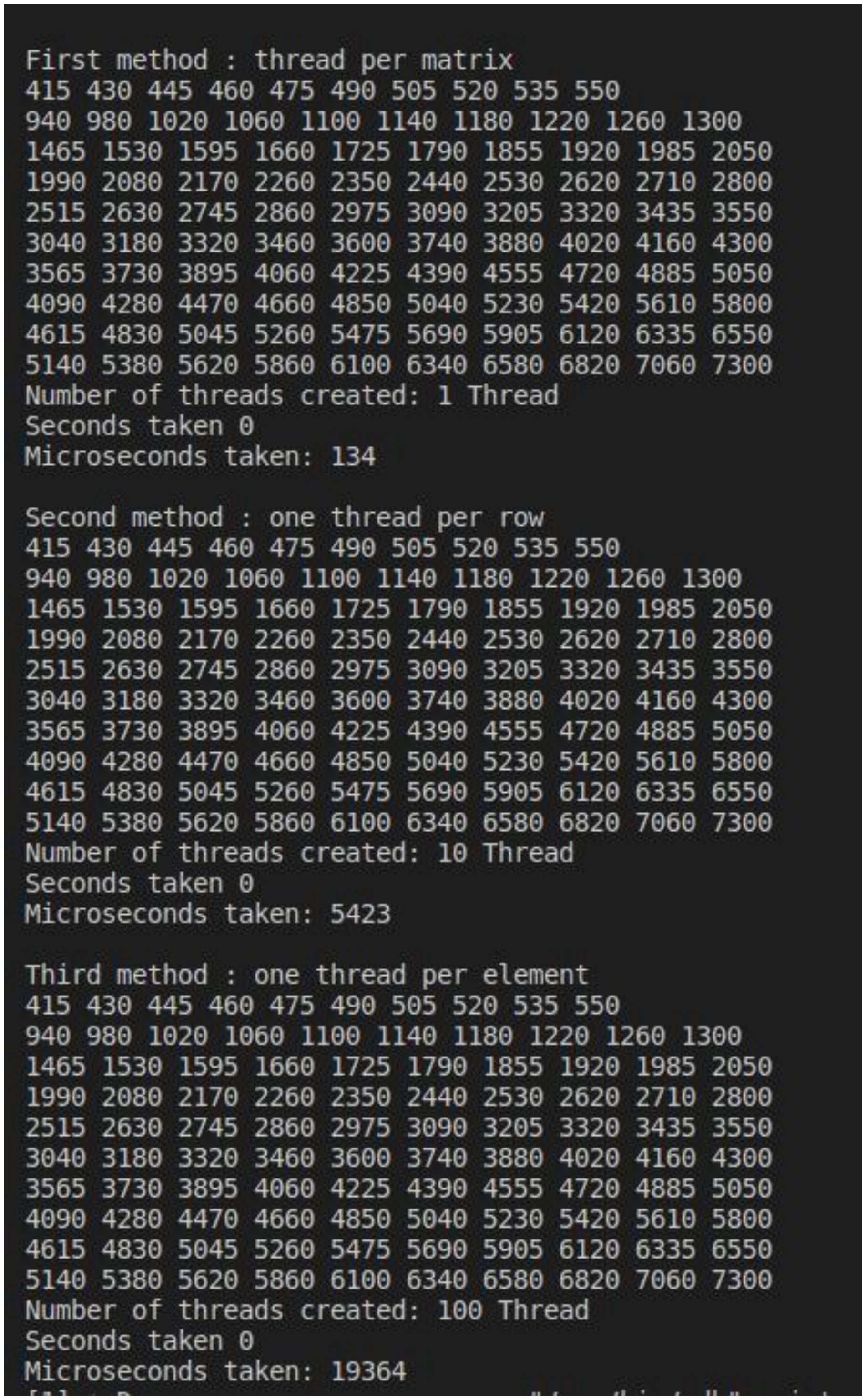
```
Method: A thread per matrix
row=5 col=4
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120
1075 1190 1305 1420
```



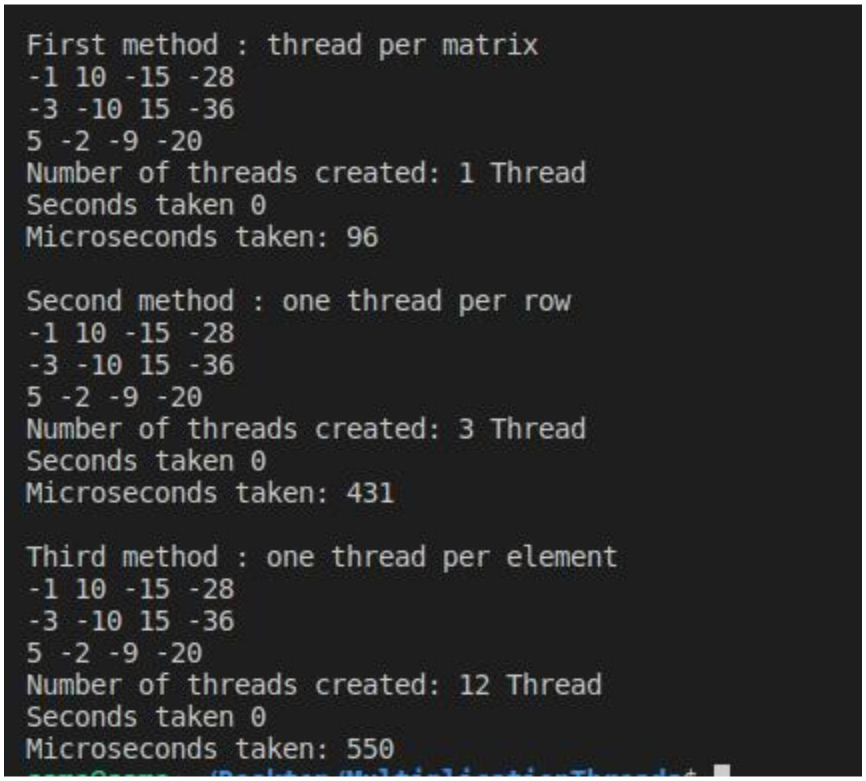
```
Method: A thread per row
row=5 col=4
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120
1075 1190 1305 1420
```

A comparison between the three methods of matrix multiplication.

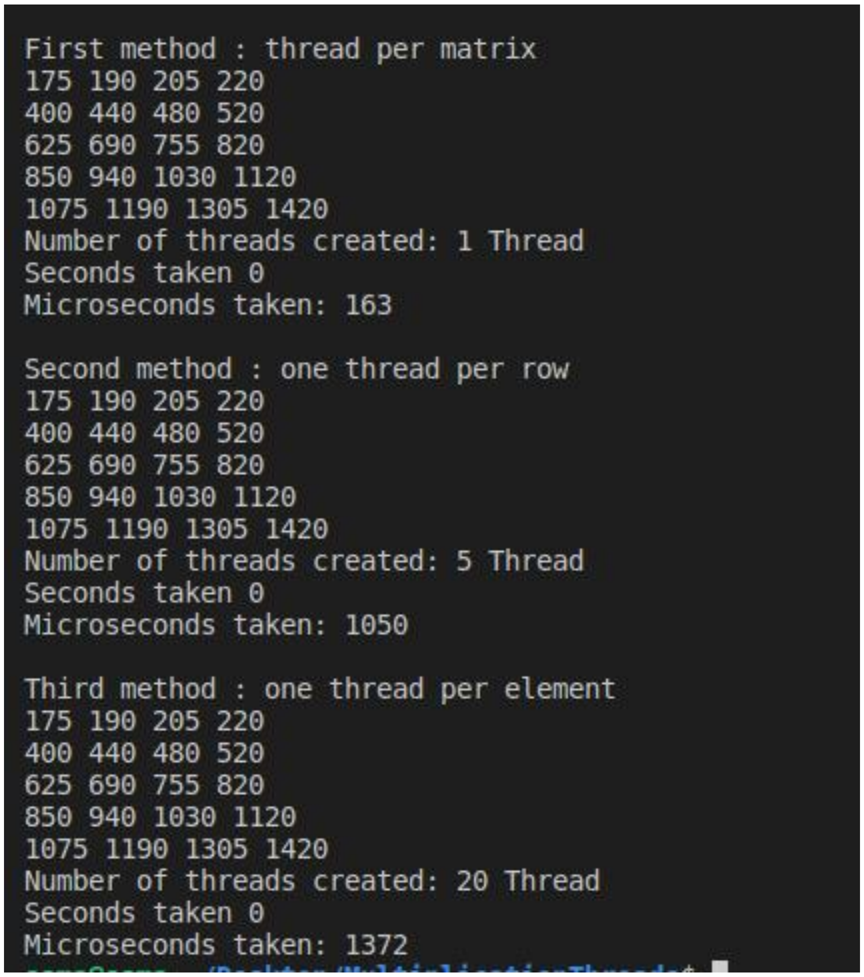
In case of thread per matrix it needs 1 thread while second method needs thread equals number of rows and the third methods needs number of thread equals rows*columns so as we can see in the photos the first method take less time because the thread creation had an overhead and we will see that the second and third methods are better when the thread creation overhead become negligible with respect to the calculations.



```
First method : thread per matrix
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
Number of threads created: 1 Thread
Seconds taken 0
Microseconds taken: 134

Second method : one thread per row
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
Number of threads created: 10 Thread
Seconds taken 0
Microseconds taken: 5423

Third method : one thread per element
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
Number of threads created: 100 Thread
Seconds taken 0
Microseconds taken: 19364
```



```
First method : thread per matrix
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
Number of threads created: 1 Thread
Seconds taken 0
Microseconds taken: 96

Second method : one thread per row
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
Number of threads created: 3 Thread
Seconds taken 0
Microseconds taken: 431

Third method : one thread per element
-1 10 -15 -28
-3 -10 15 -36
5 -2 -9 -20
Number of threads created: 12 Thread
Seconds taken 0
Microseconds taken: 550
```



```
First method : thread per matrix
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120
1075 1190 1305 1420
Number of threads created: 1 Thread
Seconds taken 0
Microseconds taken: 163

Second method : one thread per row
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120
1075 1190 1305 1420
Number of threads created: 5 Thread
Seconds taken 0
Microseconds taken: 1050

Third method : one thread per element
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120
1075 1190 1305 1420
Number of threads created: 20 Thread
Seconds taken 0
Microseconds taken: 1372
```

Running test cases video

https://drive.google.com/file/d/1U28lGCRjpCaru4SnITEelmwIhTZflz7o/view?usp=sharing