Islam Yasser Mahmoud Mohamed – 20010312

Lab2 OS

How code is organized:

- A pool of global variables declared so as to be defined among all the program functions
- 10 functions each perform a set of related actions also to maximize code usability
- Only on global array C is used among the three methods of multiplications where it is cleared before every time it is reused again

Code main functions:

- mainThread(): is considered the main thread of the program and calls other functions
- readInputFromFiles(): changes file names accordingly and calls readFile() function
- readFile(): read columns and rows number from the input files then calls readArray() function
- readArray(): read the matrices from the files and store them in the global arrays A and B
- ThreadPerMatrix(): performs matrix multiplication normally and sequentially without creating threads
- ThreadPerRow(): creates a number of threads equal to the number of rows in matrix A where each thread is responsible to compute its row in the output matrix C
- ThreadPerElement: creates a number of threads equal to the number of elements in matrix C where each thread is responsible for multiplying a row from Matrix A and a column from Matrix B to compute a single element in C

How to run the code:

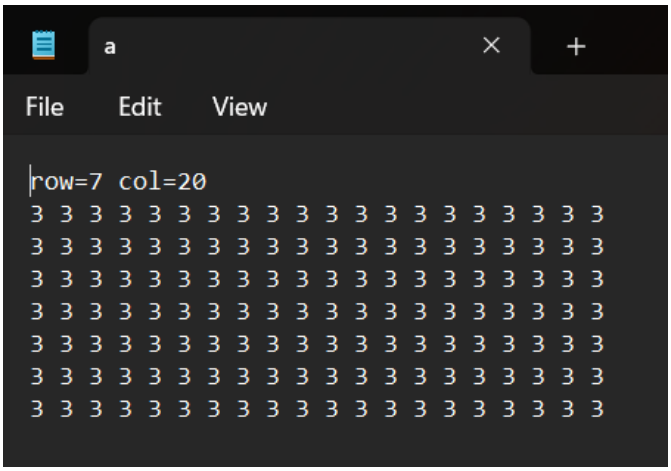
- Open the command Prompt
- Go to the path of the executable file using cd
- write gcc -o matMultp main.c

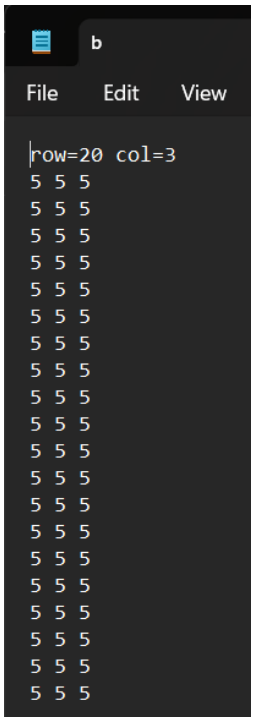
- Now you can write .\matMultp without parameters so file names will be by default a b c
- Or you can write three parameters with it so file names will be these parameters

Sample Runs:

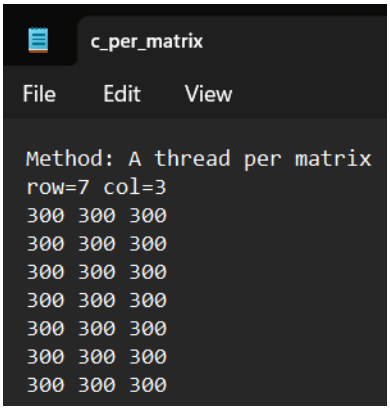
```
C:\Users\Islam\Desktop\Lab 2 OS>.\matMultp
thread_per_matrix method created 0 threads
Seconds taken 0
Microseconds taken: 0

thread_per_row method created 7 threads
Seconds taken 0
Microseconds taken: 1026

thread_per_element method created 21 threads
Seconds taken 0
Microseconds taken: 1967
```

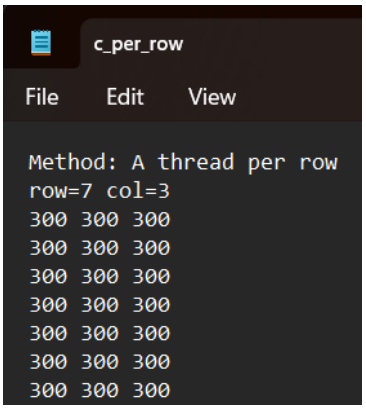
input files a & b

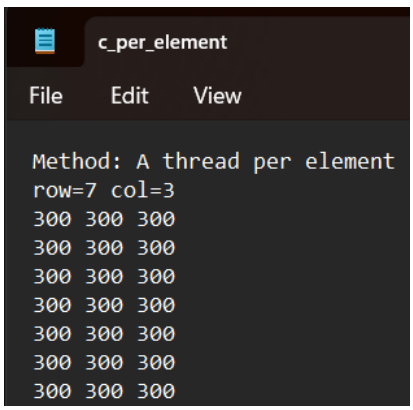




Output files







```
C:\Users\Islam\Desktop\Lab 2 OS>.\matMultp f s o
thread_per_matrix method created 0 threads
Seconds taken 0
Microseconds taken: 0

thread_per_row method created 10 threads
Seconds taken 0
Microseconds taken: 1020

thread_per_element method created 100 threads
Seconds taken 0
Microseconds taken: 3986
```

Input files:

f

File Edit View

```
row=10 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
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
```

s

File Edit View

```
row=10 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
51 52 53 54 55 56 57 58 59 60
61 62 63 64 65 66 67 68 69 70
71 72 73 74 75 76 77 78 79 80
81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100
```

Output files

o_per_matrix

File Edit View

```
Method: A thread per matrix
row=10 col=10
3355 3410 3465 3520 3575 3630 3685 3740 3795 3850
7955 8110 8265 8420 8575 8730 8885 9040 9195 9350
12555 12810 13065 13320 13575 13830 14085 14340 14595 14850
17155 17510 17865 18220 18575 18930 19285 19640 19995 20350
21755 22210 22665 23120 23575 24030 24485 24940 25395 25850
26355 26910 27465 28020 28575 29130 29685 30240 30795 31350
30955 31610 32265 32920 33575 34230 34885 35540 36195 36850
35555 36310 37065 37820 38575 39330 40085 40840 41595 42350
40155 41010 41865 42720 43575 44430 45285 46140 46995 47850
44755 45710 46665 47620 48575 49530 50485 51440 52395 53350
```

o_per_row

File Edit View

```
Method: A thread per row
row=10 col=10
3355 3410 3465 3520 3575 3630 3685 3740 3795 3850
7955 8110 8265 8420 8575 8730 8885 9040 9195 9350
12555 12810 13065 13320 13575 13830 14085 14340 14595 14850
17155 17510 17865 18220 18575 18930 19285 19640 19995 20350
21755 22210 22665 23120 23575 24030 24485 24940 25395 25850
26355 26910 27465 28020 28575 29130 29685 30240 30795 31350
30955 31610 32265 32920 33575 34230 34885 35540 36195 36850
35555 36310 37065 37820 38575 39330 40085 40840 41595 42350
40155 41010 41865 42720 43575 44430 45285 46140 46995 47850
44755 45710 46665 47620 48575 49530 50485 51440 52395 53350
```

o_per_element

File Edit View

```
Method: A thread per element
row=10 col=10
3355 3410 3465 3520 3575 3630 3685 3740 3795 3850
7955 8110 8265 8420 8575 8730 8885 9040 9195 9350
12555 12810 13065 13320 13575 13830 14085 14340 14595 14850
17155 17510 17865 18220 18575 18930 19285 19640 19995 20350
21755 22210 22665 23120 23575 24030 24485 24940 25395 25850
26355 26910 27465 28020 28575 29130 29685 30240 30795 31350
30955 31610 32265 32920 33575 34230 34885 35540 36195 36850
35555 36310 37065 37820 38575 39330 40085 40840 41595 42350
40155 41010 41865 42720 43575 44430 45285 46140 46995 47850
44755 45710 46665 47620 48575 49530 50485 51440 52395 53350
```

Comparison:

Thread_per_matrix:

- Single thread is used which is the main thread of the program to compute the result matrix by multiplying each row of the first matrix by all the columns in the second matrix
- it is sequential and no threads are created
- no overhead for threads creation and is considered efficient for small matrices

Thread_per_row:

- A thread is created for each row of the first matrix where each thread multiply one row of the first matrix by all columns in the second matrix
- Threads creation causes more overhead and requires memory management and synchronization
- Can be efficient for large matrices

Thread_per_element:

- A thread is created for each element of the result matrix where each thread multiply one row from first and one column from second matrix
- Threads creation causes more and more overhead and requires more memory management and more synchronization
- Can be efficient for very large matrices