Lab 2: Threads

16th November 2020

Name: Karim Ahmed Elhawaty ID: 48

Code Organization and Main functions:

- Code is split into 3 mains files:
  - main.c
    - `void parse(int num_args, char** args)`

      Parses execution flags and sets correct file input and output names and sets the computation method

    - `void main(int num_args, char** args)`

      Calls methods in correct ordering from parsing arguments - in format of ./multp a b c {0 or 1} where 0 or 1 is the execution method and it is optional - to reading matrices from files and calling the correct computation method then outputting it to file with time spent and num_thread used.

  - compute.c
    - `void* compute_element(void *point)`

      Routine to compute_element (point(i,j)) for a given thread and store the result in Matrix out;

    - `void compute_element_with_thread()`

      Calling this method will multiply the external Matrix a and Matrix b and store the result in Matrix out by calculating each element with a separate thread.

    - `void* compute_row(void* row)`

      Routine to compute_row (row) for a given thread and store the result in Matrix out.

    - `void compute_row_with_thread()`

Calling this method will multiply the external Matrix a and Matrix b and store the result in Matrix out by calculating each row with a separate thread.

- io_utils.c
  - `Matrix* new_matrix(int n, int m)`

    Allocates a matrix of size n and m and returns a pointer to the allocated struct.

  - `Matrix* readMatrixFromFile(char* fileName)`
  - `void writeMatrixToFile(char* fileName, Matrix* mat)`

How to compile code:

1. Install new version of CMake on Ubuntu: https://askubuntu.com/a/1157132.
2. Run these two commands in project directory.
   a. `$ cmake -H. -Bbuild`
   b. `$ cmake --build build -- -j3`
3. Executable will be in the Build file.

How to run code:

- Run executable with either of the following 3 formats:
  - **./multp**

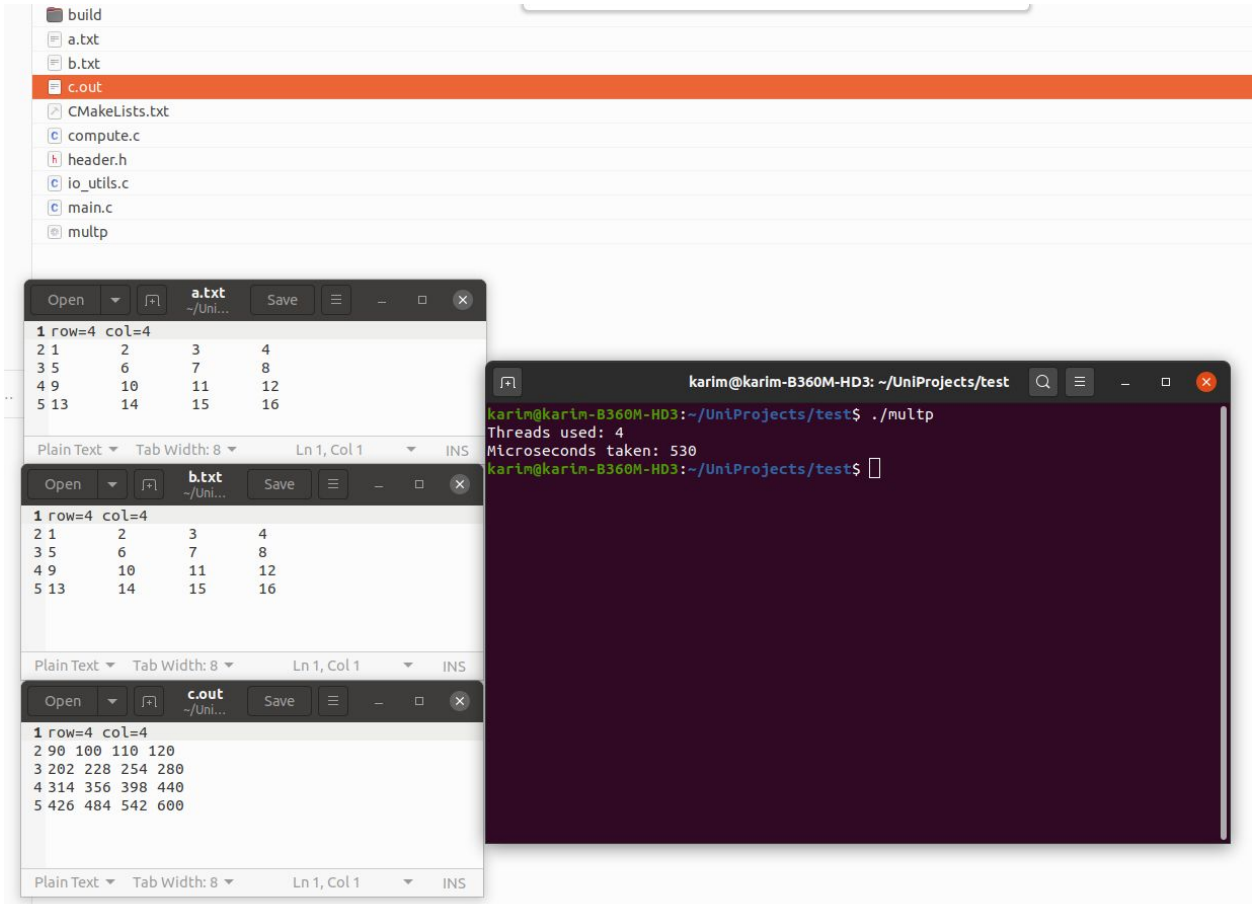
    It will default to a.txt and b.txt and store the result in c.out
  - **./multp file_A file_B file_output**

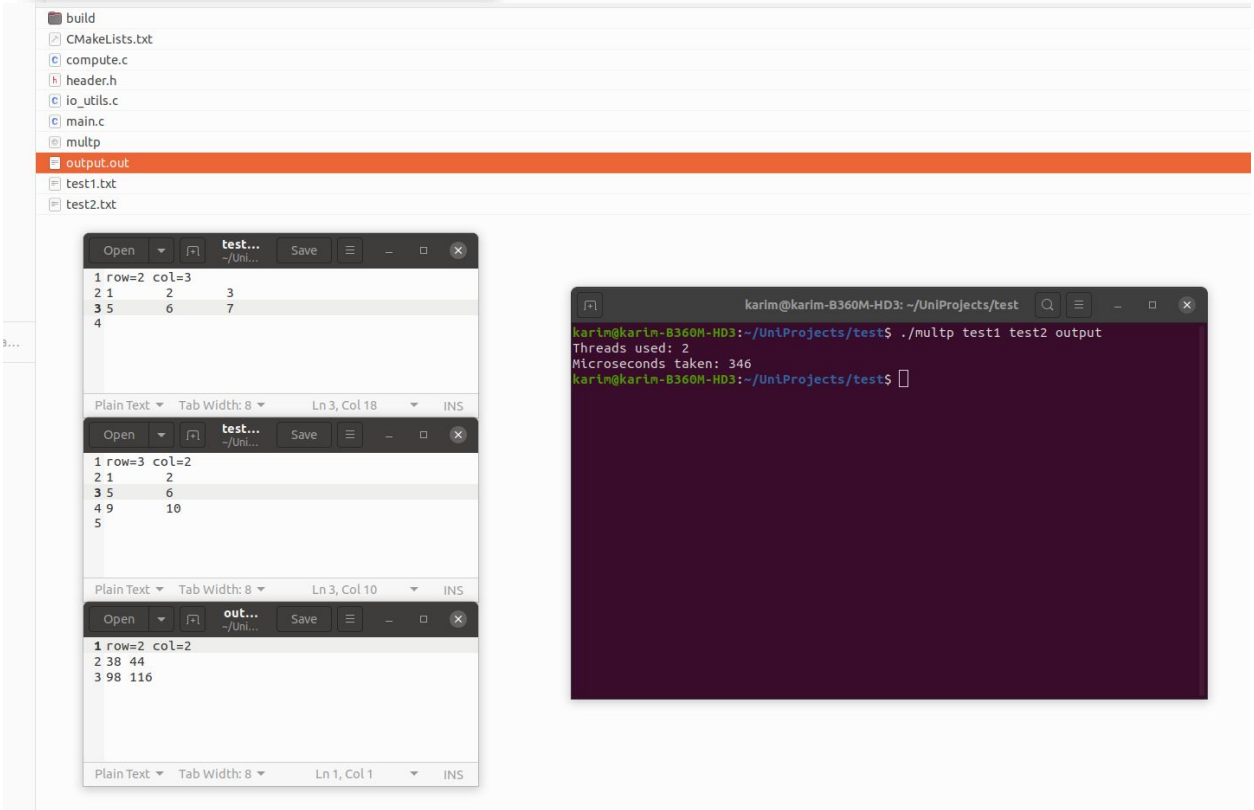
    It will read from file_A.txt and file_B.txt and store the result in file_output.out
  - **./multp file_A file_B file_output 0**

    Similar to previous but it will compute using a thread for each row, which is default
  - **./multp file_A file_B file_output 1**

    Similar to previous but it will compute using a thread for each element

Sample runs:


build
a.txt
b.txt
c.out
CMakeLists.txt
compute.c
header.h
io_utils.c
main.c
multp
Open
a.txt
~/Uni...
Save
1 row=4 col=4
2 1 2 3 4
3 5 6 7 8
4 9 10 11 12
5 13 14 15 16
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
Open
b.txt
~/Uni...
Save
1 row=4 col=4
2 1 2 3 4
3 5 6 7 8
4 9 10 11 12
5 13 14 15 16
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
Open
c.out
~/Uni...
Save
1 row=4 col=4
2 90 100 110 120
3 202 228 254 280
4 314 356 398 440
5 426 484 542 600
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
karim@karim-B360M-HD3: ~/UniProjects/test
karim@karim-B360M-HD3:~/UniProjects/test$ ./multp
Threads used: 4
Microseconds taken: 530
karim@karim-B360M-HD3:~/UniProjects/test$


build
CMakeLists.txt
compute.c
header.h
io_utils.c
main.c
multp
output.out
test1.txt
test2.txt
Open
test...
~/Uni...
Save
1 row=2 col=3
2 1 2 3
3 5 6 7
4
Plain Text
Tab Width: 8
Ln 3, Col 18
INS
Open
test...
~/Uni...
Save
1 row=3 col=2
2 1 2
3 5 6
4 9 10
5
Plain Text
Tab Width: 8
Ln 3, Col 10
INS
Open
out...
~/Uni...
Save
1 row=2 col=2
2 38 44
3 98 116
Plain Text
Tab Width: 8
Ln 1, Col 1
INS
karim@karim-B360M-HD3: ~/UniProjects/test
karim@karim-B360M-HD3:~/UniProjects/test$ ./multp test1 test2 output
Threads used: 2
Microseconds taken: 346
karim@karim-B360M-HD3:~/UniProjects/test$

Comparison between methods:

n	Threads_per_row		Thread_per_element	
	Time (microsec)	Threads used	Time (microsec)	Threads used
1	372	1	772	1
10	895	10	7037	100
100	11127	100	191874	10000
200	21545	200	Memory_limit	40000