Lab Assignment 2: Matrix Multiplication

I. Code Organization

- Each module is separated to header file and its implementation where any function that will get used by other modules will be defined in the header file and the header file contains the full documentation of the function. An implementation file might contain helper function where their documentation is above their prototype in the implementation file.

- Modules in the project:
    - ***matrix:*** module responsible about initializing, destroying and defining the matrix object structure.
    - ***matrix_operations:*** module responsible about the micro operations like calculating a single element given its coordinates in the output matrix or calculating a specific row in the output matrix. Meaning they are operations that will be given for individual threads to execute.
    - ***matrix_multiplier:*** module responsible about calculating the output of two matrixes using different methods.
    - ***file_processing:*** module responsible about handling with the files, meaning opening , closing and checking if they exist.
    - ***matrix_file_processor:*** module responsible about reading and parsing a file into a matrix or outputting a matrix to a file.
    - ***matrix_multiplicator_app:*** our top level module combining the different modules and generating the requested output.
    - ***default_values:*** Only a header which contains some constants used throughout the code.
    - ***matrix_stat_formatter:*** an extra module responsible about generating some statistiques file about the time taken in each method for different calculation methods in order to be able to compare them.

II. Main Functions

- init_matrix :

Implemented in matrix.c and defined in matrix.h. This function is responsible for looking initializing a matrix with the given dimensions given to it it..

- read_matrix :

Implemented in matrix_file_processor.c and defined in matrix_file_processor.h . This function is responsible of reading from a given file name, and parsing the data read into a matrix object and returning that matrix.

- write_matrix :

Implemented in matrix_file_processor.c and defined in matrix_file_processor.h. This function is responsible of writing a matrix object in a suitable format in an output file.

- calculate_element :

Implemented in matrix_operations.c and defined in matrix_operations.h. Given three matrices a,b and c and two coords x and y. This function calculates the value that will normally be in the matrix resulting from the multiplication of a * b at coords x,y and saves it in c(x,y).

- calculate_row :

Implemented in matrix_operations.c and defined in matrix_operations.h. Given three matrices a,b and c and a coordinate x. This function calculates the values that will normal be in the matrix resulting from the multiplication of a * b in row x and stores these value in row x in matrix c.

- multiply_threaded_elements :

Implemented in matrix_multiplier.c and defined in matrix_multiplier.h. This function is responsible of calculating the output of the multiplication of two matrices by calculating each element in the output matrix in an independent thread and combining their output and returning the number of threads created.

- multiply_threaded_row :

Implemented in matrix_multiplier.c and defined in matrix_multiplier.h. This function is responsible of calculating the output of the multiplication of two matrices by calculating each row in the output matrix in an independent thread and combining their output and returning the number of threads created.

- run_matrix_mul_app :

Implemented in matrix_multiplicator_app.c and documented in matrix_multiplicator_app.h. This is considered the top level interface of the program as it's where we give the file names of the input matrices and the output file name. It will be responsible of reading matrices from the files using and combining the other modules, handling any error at the parsing stage, then start calculating the output, time taken to calculate it and the number of threads created and output them in a suitable format for each method we have for multiplying matrices all while detecting if an error happens and reacting suitably.

III. Compilation & Running Guide

- Compilation :
    - Go to the folder of the project <lab2_45>. You should see src folder, include folder and makefile.
    - Open a terminal in this directory.
    - Type the command 'make' and run it.
    - It should end successfully having created two folders in the directory obj and bin.
    - The matmult.out should be in bin folder.
    - If for any reason you want to clean the directory and compile again, enter the command 'make clean' which deletes all the object files and the output program. Then rerun 'make'.

- Running the matrix multiplicator program :
    - After compiling the project, go to the bin folder.
    - Open a terminal in this directory.
    - To run with default file values of a.txt, b.txt and output file c.txt to calculate the output of a * b and save it in c_1.txt which indicates row threaded output and c_2.txt that indicates element threaded output, enter the command './matmult.out'

- Otherwise , provide values instead of the default values by entering the commands ‘./matmult.out <first input file> <second input file> <output prefix file name>’ however all 3 values must be submitted in order for the program to run.

IV. Sample runs

1. *Sample run 1 :*

```
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ls
a.txt  b.txt  matmult.out
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ cat a.txt
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
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ cat b.txt
row=4 col=4
1       2       3       4
5       6       7       8
9       10      11      12
13      14      15      16
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ./matmult.out
Error in matrixes : first matrix columns must match second matrix rows to be able to multiply them...
Operation terminated
```

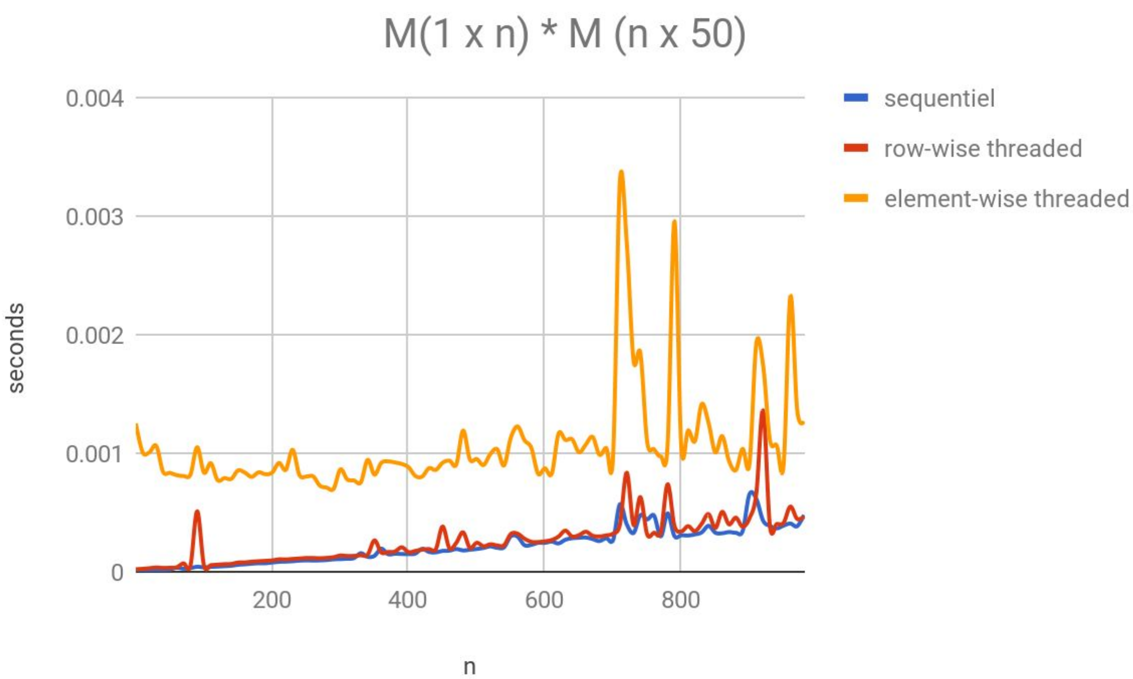
2. *Sample run 2 :*

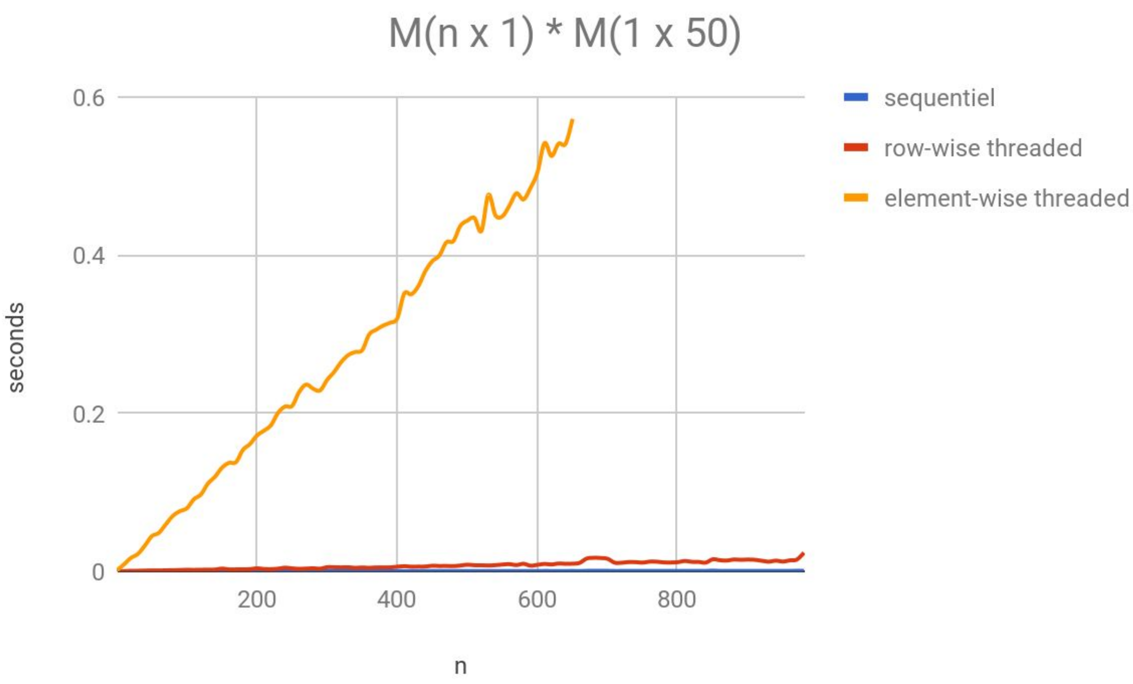
```
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ls
amr1_1.txt  amr1_2.txt  matmult.out
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ cat amr1_1.txt
row=2 col=3
1       2       3
4       5       6
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ cat amr1_2.txt
row=3 col=2
7       8
9       10
11      12
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ./matmult.out amr1_1.txt amr1_2.txt amr1_o.txt
Number of threads made for <each row computed by a thread>(method 1) : 2
Time taken to compute it in seconds =  0.000214

Number of threads made for <each element computed by a thread>(method 2) : 4
Time taken to compute it in seconds =  0.000711

amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ls
amr1_1.txt  amr1_2.txt  amr1_o_1.txt  amr1_o_2.txt  matmult.out
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ cat amr1_o_1.txt
58      64
139     154
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ cat amr1_o_2.txt
58      64
139     154
```

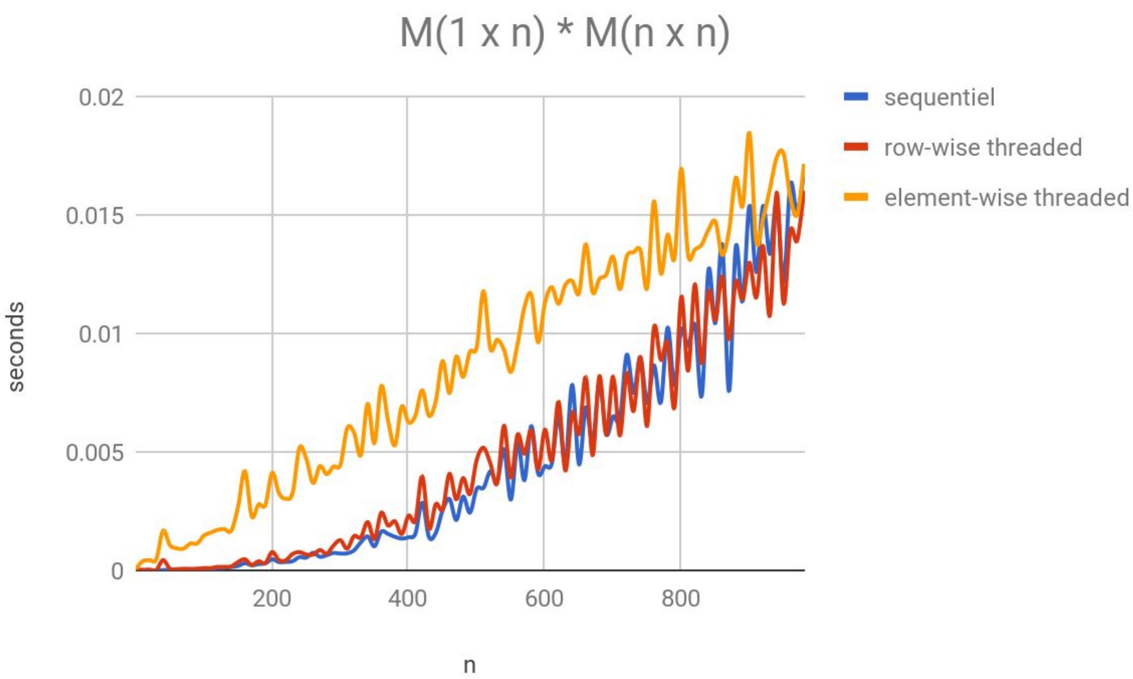
3. <u>*Sample run 3 :*</u>

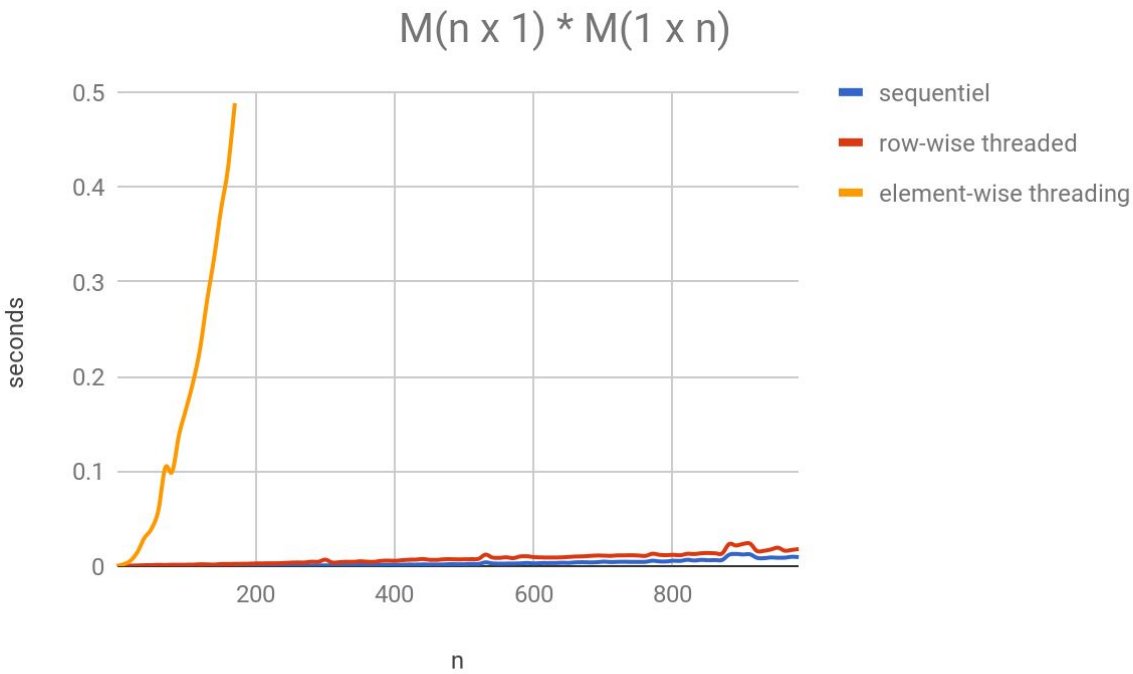
```
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ls
a.txt  b.txt  matmult.out
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ head a.txt -n 1
row=800 col=800
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ head b.txt -n 1
row=800 col=800
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ./matmult.out a.txt b.txt d
Number of threads made for <each row computed by a thread>(method 1) : 800
Time taken to compute it in seconds =  1.227084

ERROR during creation of thread in threaded element method with return code 11
Failed to create the threads needed sucessfully --> Operation failed !
Number of threads made for <each element computed by a thread>(method 2) : 32750
Time taken to compute it in seconds =  0.577110

amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ ls
a.txt  b.txt  d_1.txt  d_2.txt  matmult.out
amrnasr@amrnasr-Lenovo-Y50-70:~/MyFiles/projects/OS/lab2_45/bin$ head d_1.txt -n 1
170986800       341973600       512960400       683947200       854934000       1025920800      1196907600
3815200 2564802000      2735788800      2906775600      3077762400      3248749200      3419736000      35907
787630400       4958617200      5129604000      5300590800      5471577600      5642564400      5813551200
0458800 7181445600      7352432400      7523419200      7694406000      7865392800      8036379600      82073
404274000       9575260800      9746247600      9917234400      10088221200     10259208000     10430194800
27102400        11798089200     11969076000     12140062800     12311049600     12482036400     12653023200
49930800        14020917600     14191904400     14362891200     14533878000     14704864800     14875851600
72759200        16243746000     16414732800     16585719600     16756706400     16927693200     17098680000
95587600        18466574400     18637561200     18808548000     18979534800     19150521600     19321508400
```

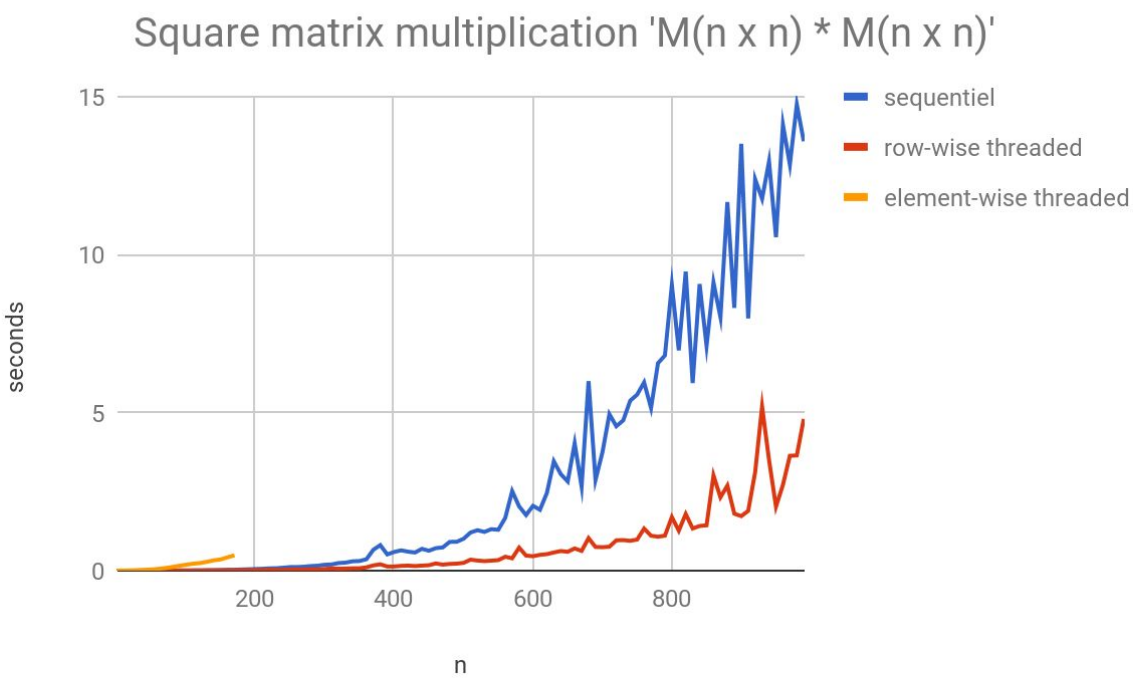
V. Comparison between methods




M(n x 1) * M(1 x 50)
seconds
0.6
0.4
0.2
0
200
400
600
800
n
sequentiel
row-wise threaded
element-wise threaded


M(1 x n) * M(n x n)
seconds
0.02
0.015
0.01
0.005
0
200
400
600
800
n
sequentiel
row-wise threaded
element-wise threaded





Conclusion :

- We can notice that sometime in the graphs, the element wise threaded performance data is cut before the other two methods and that's due to the OS being unable to give our application more threads due to the limitation and such it fails at providing us with useful data after that point and that's due to the OS limiting the number of threads our process is capable of creating.

- We can also conclude that as the size increase in all cases, the element wise threading is the worst way to calculate the multiplication of matrices and shouldn't be used as with the increasing size of the output matrices, a large number of threads get created and would actually waste a lot of time in switching between the massive number of threads to complete and because of that overhead, it is the least efficient method.

- In case of having our first matrix in the multiplication having only one dimension, both the traditional sequential approach and row-wise threading are similar since in the row-wise threading we'd only have created one thread to calculate our output.

- However, as the matrices become more evenly distributed and becomes close to a square matrix, we notice how the row-wise threading is much faster than the sequential as the matrices size increases and that's due to multi core processors separating the work of row calculating threads on multiple cores, hence enhancing the speed of calculations and due to the relatively normal number of threads, the overhead is worth it as in the end the time decreases.

- Finally, when we start moving from the square matrix to having our first matrix having only 1 column, the performance of the sequential approach is slightly better than row-wise threading due to the fact that the overhead of creating these threads and switching between them is much larger due to the nature of having multiple rows with little elements in them, making row-wise threading really inefficient.

- Possible solution : Would be to check matrices sizes if it is closer to having a single row with multiple columns, use row-wise threading but if it is closer to having a single column with multiple row, the use of column-wise threading could enhance the results theoretically.