

Multi-Threaded Matrix Multiplication

Overall organization of the code:

The program reads the inputs file and stores the 2 matrices A & B and then begins to execute the row-wise multithreaded approach by creating a number of threads equals to the number of rows expected for the output matrix and then waits until all threads are successfully executed ,then the program creates a number of threads equals to the number of elements expected in the output matrix.

Main Functions :

Initiate program :

Responsible for initiating the program and preparing the environment to be able to execute by:

- Reading from the 1st text file the 1st matrix.
- Reading from the 2st text file the 2st matrix.
- Allocating memory for the output matrix.

Row Wise threading :

It creates a number of threads equal to the number of rows expected in the o/p matrix and waits for these threads to execute successfully.

Element Wise threading :

It creates a number of threads equal to the number of elements expected in the o/p matrix and waits for these threads to execute successfully.

Get Row :

It is the function done by each thread that is executed independently from each other it calculates the corresponding row of the output matrix to the passed parameter.

Get Element :

It is the function done by each thread that is executed independently from each other it calculates the corresponding elements of the output matrix to the passed parameters i & j.

Compilation & Running :

To compile the project: you need to run the following command

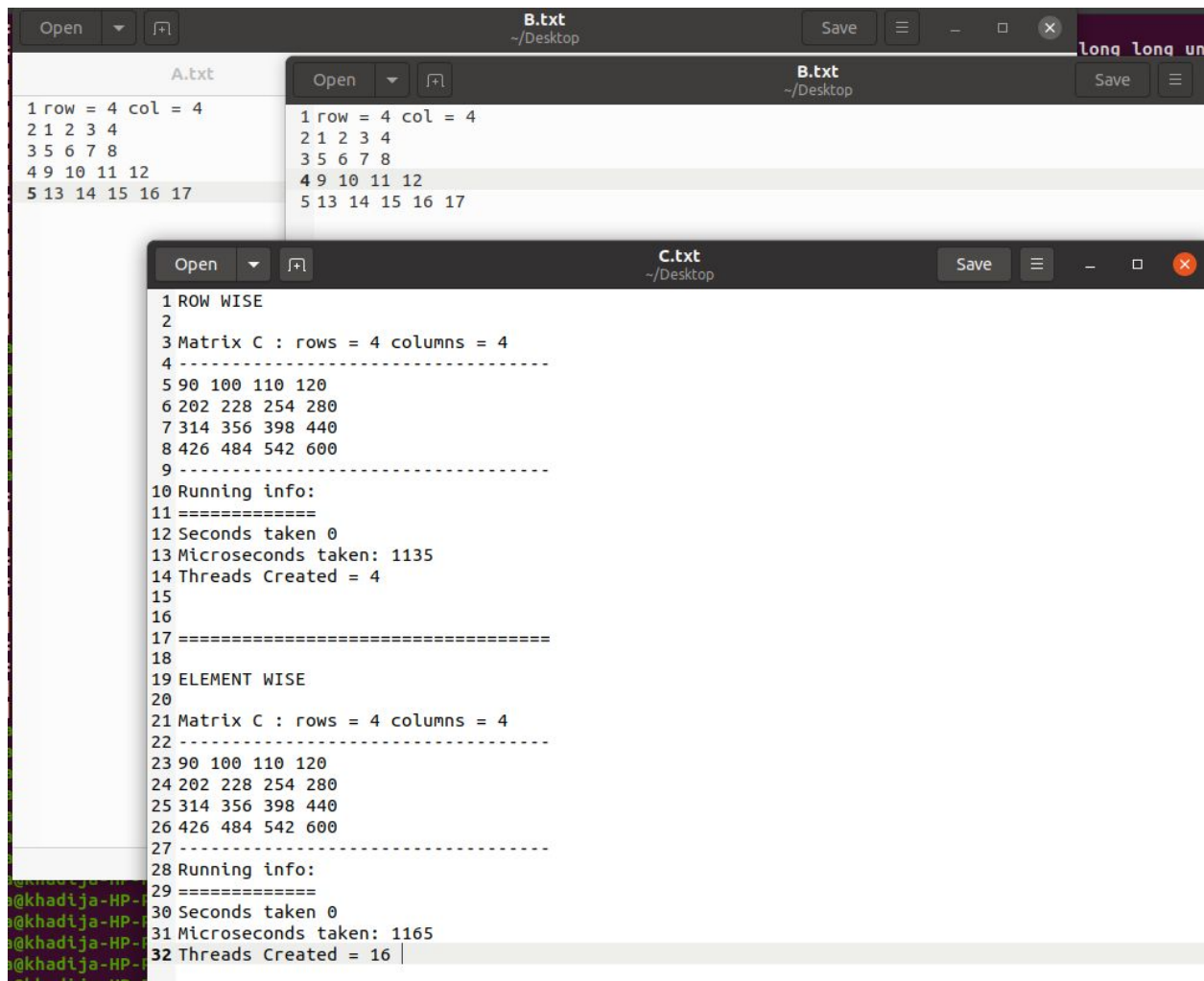
```
Make -f Makefile
```

To run the project : you need to run the following command

```
./matmult matrix1_file.txt matrix2_file.txt matrix3_file.txt
```

Sample Runs:

Multiplying Matrices :



```
Open A.txt
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 17

Open B.txt
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 17

Open C.txt
1 ROW WISE
2
3 Matrix C : rows = 4 columns = 4
4 -----
5 90 100 110 120
6 202 228 254 280
7 314 356 398 440
8 426 484 542 600
9 -----
10 Running info:
11 =====
12 Seconds taken 0
13 Microseconds taken: 1135
14 Threads Created = 4
15
16
17 =====
18
19 ELEMENT WISE
20
21 Matrix C : rows = 4 columns = 4
22 -----
23 90 100 110 120
24 202 228 254 280
25 314 356 398 440
26 426 484 542 600
27 -----
28 Running info:
29 =====
30 Seconds taken 0
31 Microseconds taken: 1165
32 Threads Created = 16
```

Open

A.txt

~/Desktop

Save

A.txt

1 row = 3 col = 3

2 1 2 3

3 5 6 7

4 9 10 11

Open

B.txt

~/Desktop

Save

B.txt

1 row = 3 col = 4

2 1 2 3 4

3 5 6 7 8

4 9 10 11 12

5

Open

C.txt

~/Desktop

Save

C.txt

1 ROW WISE

2

3 Matrix C : rows = 3 columns = 4

4 -----

5 38 44 50 56

6 98 116 134 152

7 158 188 218 248

8 -----

9 Running info:

10 =====

11 Seconds taken 0

12 Microseconds taken: 240

13 Threads Created = 3

14

15

16 =====

17

18 ELEMENT WISE

19

20 Matrix C : rows = 3 columns = 4

21 -----

22 38 44 50 56

23 98 116 134 152

24 158 188 218 248

25 -----

26 Running info:

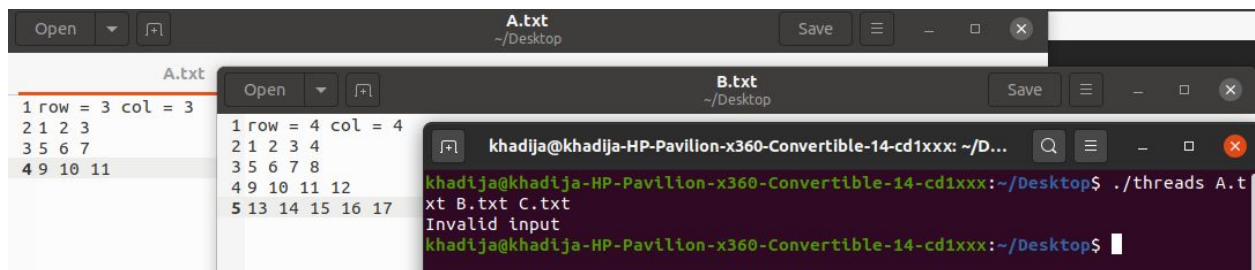
27 =====

28 Seconds taken 0

29 Microseconds taken: 270

30 Threads Created = 12

Invalid Input :



The screenshot shows a terminal window with two text files, A.txt and B.txt, open in the background. A.txt contains a 3x3 matrix and B.txt contains a 4x4 matrix. The terminal output shows the command `./threads A.t` being executed, followed by the error message `Invalid input`.

```
khadija@khadija-HP-Pavilion-x360-Convertible-14-cd1xxx: ~/Desktop$ ./threads A.t
Invalid input
khadija@khadija-HP-Pavilion-x360-Convertible-14-cd1xxx:~/Desktop$
```

In Conclusion :

Element-wise threading approach :

- Creates threads = output_Matrix_rows * output_Matrix_columns
- Takes time more than the row wise approach

Row-wise threading approach :

- Creates threads = output_Matrix_rows
- Faster than the element wise approach