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

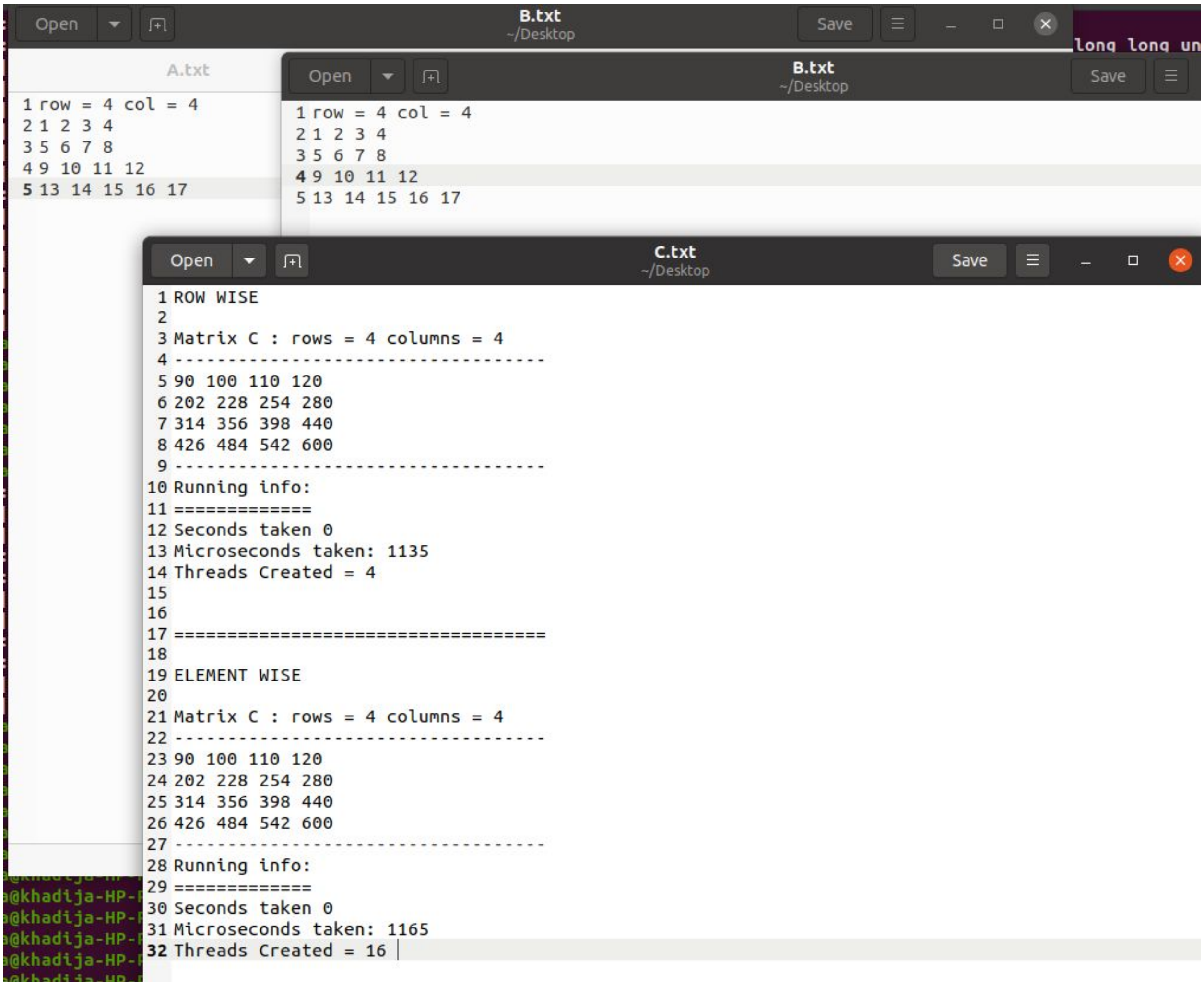
```
Make -f Makefile
```

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 :



A.txt

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

B.txt

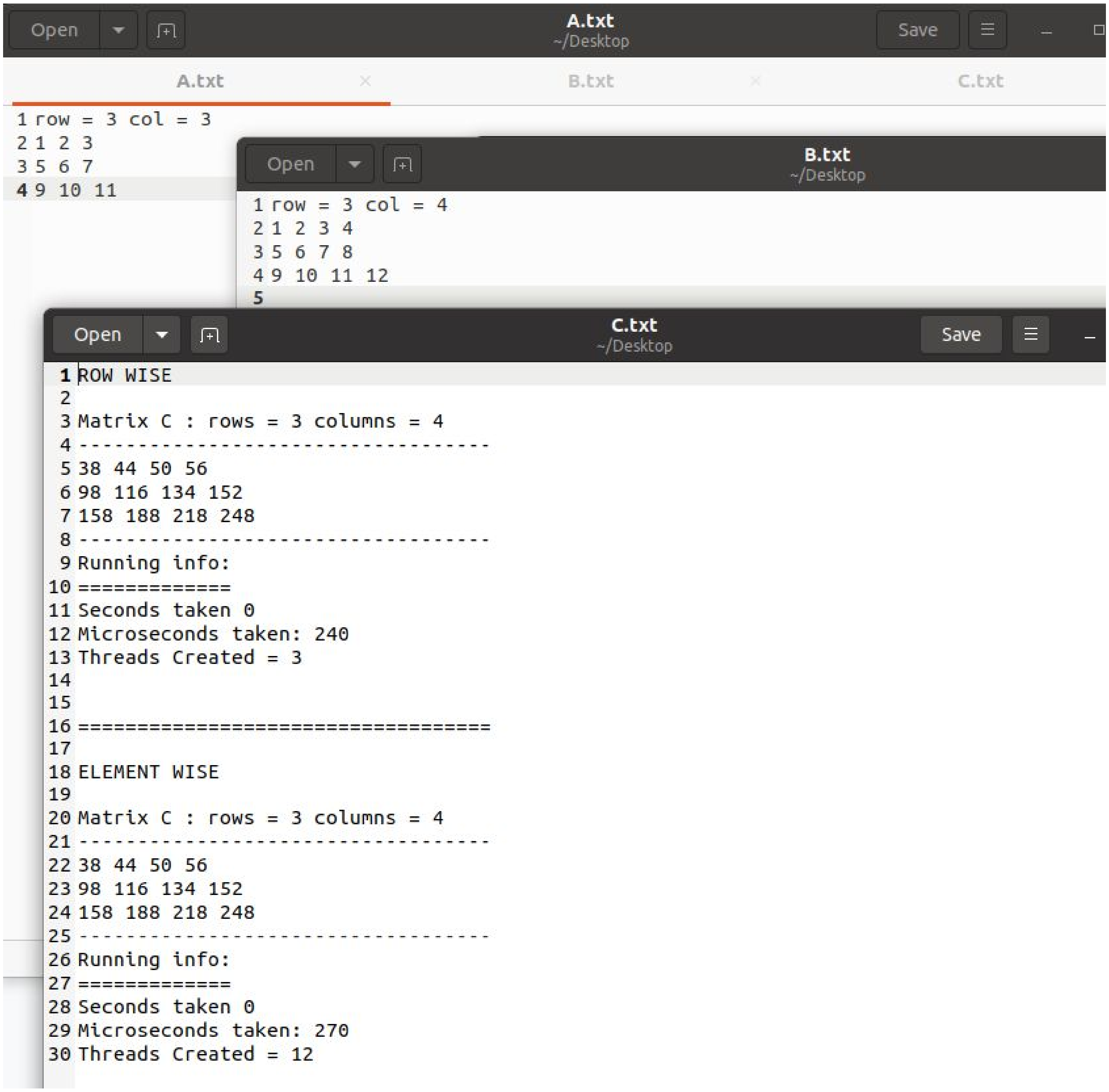
```
row = 4 col = 4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16 17
```

C.txt

```
ROW WISE

Matrix C : rows = 4 columns = 4
-----------------------------------
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
-----------------------------------
Running info:
=============
Seconds taken 0
Microseconds taken: 1135
Threads Created = 4


===================================

ELEMENT WISE

Matrix C : rows = 4 columns = 4
-----------------------------------
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
-----------------------------------
Running info:
=============
Seconds taken 0
Microseconds taken: 1165
Threads Created = 16
```

```
long long un
```

```
a@khadija-HP-
a@khadija-HP-
a@khadija-HP-
a@khadija-HP-
```

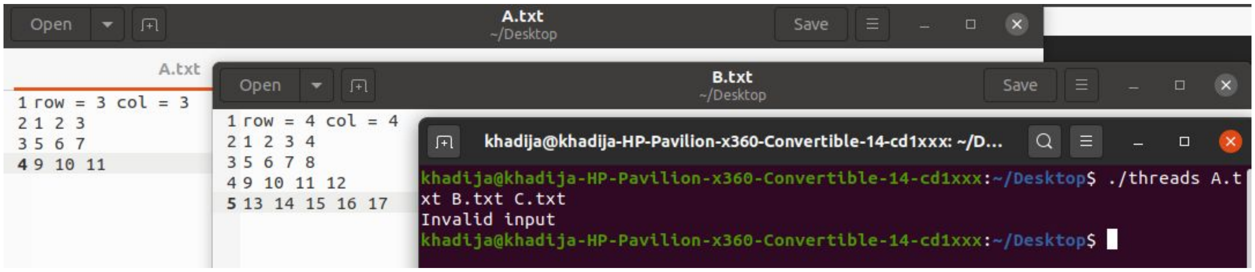


```
row = 3 col = 3
1 2 3
5 6 7
9 10 11
```

```
row = 3 col = 4
1 2 3 4
5 6 7 8
9 10 11 12

```

```
ROW WISE

Matrix C : rows = 3 columns = 4
-----------------------------------
38 44 50 56
98 116 134 152
158 188 218 248
-----------------------------------
Running info:
=============
Seconds taken 0
Microseconds taken: 240
Threads Created = 3


===================================

ELEMENT WISE

Matrix C : rows = 3 columns = 4
-----------------------------------
38 44 50 56
98 116 134 152
158 188 218 248
-----------------------------------
Running info:
=============
Seconds taken 0
Microseconds taken: 270
Threads Created = 12
```

Invalid Input :


Open
A.txt
~/Desktop
Save
B.txt
khadija@khadija-HP-Pavilion-x360-Convertible-14-cd1xxx: ~/D...

```
row = 3 col = 3
1 2 3
5 6 7
9 10 11
```

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

```
khadija@khadija-HP-Pavilion-x360-Convertible-14-cd1xxx:~/Desktop$ ./threads A.t
xt B.txt C.txt
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