Code Organization and Main Functions

In lab 2, we employed the concept of multi-threading using the problem of matrix multiplication. Different approaches were taken in creating the threads. First, we create a thread per matrix. Second, a thread per each row is created. And finally, a thread per element.

My C code is organized in the following manner:

1- void setup_env(): on running the code, the directory is changed to the current working directory.

2- int read_files(char *, char *, char *): is used to create the output files and initialize the input files.

3- void rows_cols_no(FILE *, int): is used to extract the number of rows and columns from the file.

4- void create_matrices(FILE *, int): is used to populate the matrices a, b and c.

5- void thread_per_mat(char *): initialize the threads and calculate the output then save it to the corresponding file.

6- void thread_per_row(char *): initialize the threads (per row) and save the output to the corresponding file.

7- void thread_per_element(char *): initialize the threads (per element) and save the output to the corresponding file.

8- void *compute_per_row(void *): compute the output of the corresponding row and populate the output array.

9- void *compute_per_elmnt(void *): compute the output of the corresponding element and populate the output array.

How to Compile and Run the Code

we first open the terminal in the working directory and compile the .c with this command:

```
gcc main.c -lpthread
```

we then run the code through the following command:

```
./a.out
```

in case we want to use the default names of the files (a, b, and c). And in case we want to specify the files:

```
./a.out x y z
```

where x and y are the input files and z is the output file.

Sample Runs

The contents of the first file are:

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

The second file has the same contents as the first. After running the code, 3 files are created as shown in figure 1.

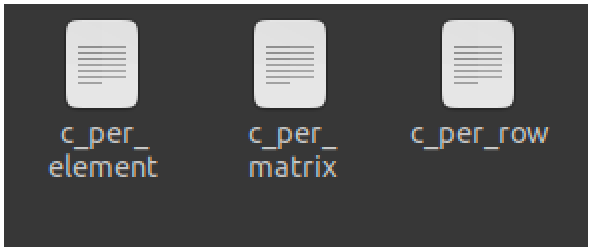
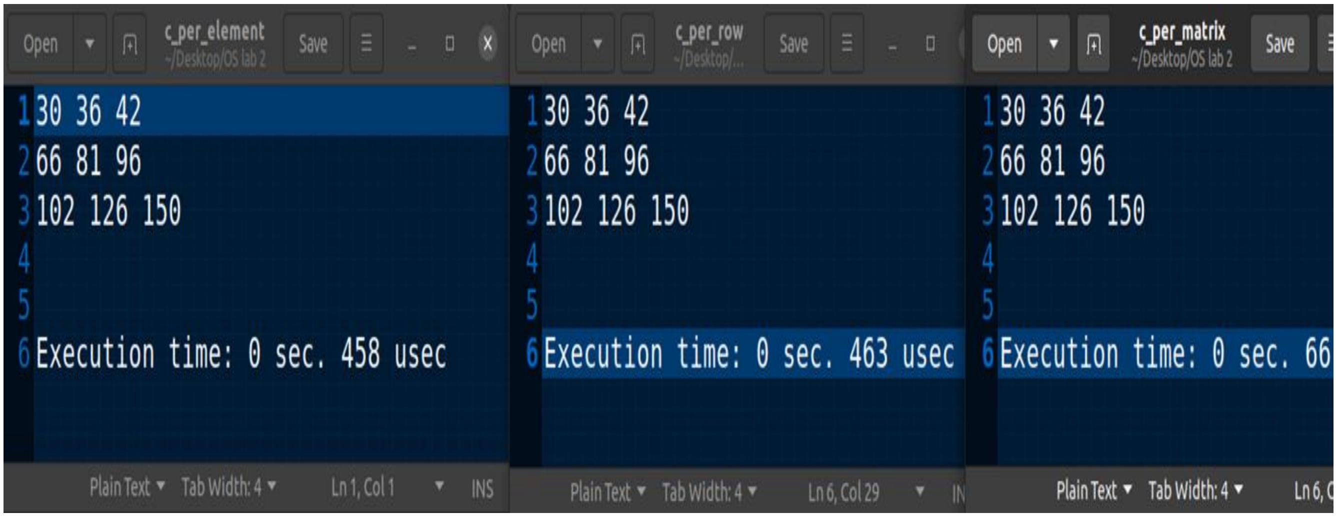


Figure 1

The result of the multiplication in each file is shown in figure 2. We notice that creating a thread per matrix is the most effective method.



```
30 36 42
66 81 96
102 126 150


Execution time: 0 sec. 458 usec
```

```
30 36 42
66 81 96
102 126 150


Execution time: 0 sec. 463 usec
```

```
30 36 42
66 81 96
102 126 150


Execution time: 0 sec. 66
```

Figure 2