Multicore Processors: Architecture & Programming
Lab 1

In this first lab you will implement a histogram in OpenMP.

What is the problem definition?

Your program reads a text file that consists of a random combination of the four letters: *a, b, c, and d*. The file can contain any number of these characters and not in a specific order. Your program must determine the which of the four letters occurred the most and print the result on the screen.
For example: *b occurred the most 400 times of a total of 1000 characters.*
The output means that the letter b occurred the most frequently among the four letters and it occurred 400 times out of the 1000 characters that were in the file.

What is the input?

The input to your program consists of three things:

- N: number of threads, can be 0 (purely sequential), 1 (OpenMP version with one thread), or 4 (OpenMP with four threads).
- num: number of characters in the file
- Filename: This is the name of the file that contains the characters. The format of the file is simple. It consists of a character followed by another character, etc. For example:
  acabdaddd

We are providing you with a file random-char.c to generate the characters file. Feel free to use that file, compile and run it to generate the files, or build your own file generator.

Assume your program is called *maxnum*, the command line is expected to be:

```
./maxnum N num filename
```

You must generate the filename first, either by using random-char.c or by hand if you want, before executing the above command.

What is output?

You program must print on the screen:

x occurred the most y times of a total of z characters.

Where x is the character that occurred the most (a, b, c, or d) and y is the number of times x has occurred in the filed and z is the total number of characters in the file.

How will you solve this?

Here is an algorithm for a sequential version:

```
Assume we have an array of four integers, one for each of the four characters
while(not done reading the characters)
{
   Read a character
   Determine the entry of the array to be incremented
   Increment the entry
}
Scan the four entries of the array and determine the highest one
Print the output on the screen
```

For the OpenMP version, the one with four threads, you need to do the following (Note: Some details are left on purpose for you to figure out):

- Load the whole file in an array of characters (i.e. string). Assume its name is: list[]
- Each thread will be responsible for N/4 characters. You need to take care of the corner case when N is not divisible by 4.
- Assume N/4 = step
- Thread 0 will take care of characters from 0 to step-1. Thread 1 will take care of characters from step to 2*step – 1, and so one.
- Let each thread have its own array of four int where it puts the count of the four characters.
- At the end, combine the four arrays into one.
- Finally, pick the largest one and print the output.

Refresh Your Knowledge about C

OpenMP is built on top of C in our case. To write the program, you need to refresh your memory about the following C items. This list is not exhaustive but contains the most important items and the ones that people tend to forget if they don't use the language a lot.

- How to read inputs from the command line. This has to do with the arguments of the main() function: argc and argv.
- How to open/close a file.
- How to used fread()
- How to use printf
- Dynamic allocation with malloc() and free()

The report

Beside your source code, you need to submit a report based that contains the results of several experiments as will be discussed. You will use the **time** command in Linux. You need to use the same CIMS machine (crunchy1, crunchy3, …) for all your runs. The **time** command, as we saw in class, produces three numbers: real, usr, sys.

You need to fill out two tables.
Table 1 contains the *real* part of the **time** command.

N →	1,000	10,000	100,000	1,000,000	10,000,000	100,000,000
Seq						
1-thread						
4-threads						

The sequential is a purely sequential code. The 1-thread contains the OpenMP commands but generates 1 thread. The 4-threads is OpenMP with 4 threads. The first row shows the number of characters in different files. So, you need to generate those different files.

Table 2 contains the *cpu+sys* part of the **time** command.

N →	1,000	10,000	100,000	1,000,000	10,000,000	100,000,000
Seq						
1-thread						
4-threads						

Draw two bar-graphs. Both of them have N as the x-axis.

- In graph 1: The y-axis is the speedup (time seq / time 4 threads) where time is the *real* of **time** command. That is, you will get the numbers from the first table.
- In graph 2: The y-axis is the efficiency = speedup (that you calculated in first graph)/4

Finally answer the following questions at the end of the report:

1. Which machine have you used to do the experiment (crunchy1, crunchy3, …)? Doing them on your laptop will not be accepted.
2. Given the Table 1 and graph 1 what is the relationship between N and the speedup?
3. How can you justify that relationship you showed in the above question?
4. Given Table 1 and Table 2: Was the *real* time mostly bigger, smaller, equal to *cpu+sys* time?
5. What is your justification for the trend in question 4 above?
6. What is the trend you see in graph 2?
7. How do you explain the trend you found in question 6 above?

Regarding compilation &naming convention:

- Name your source file: netID-code.c (where netID is your netID number).
- Name your report netID-report.pdf (where netID is your netID number).
- Do the compilation and execution on crunchyx (x = 1, 3, 5, or 6) machines.
- Use the latest version of gcc on crunchy by typing: module load gcc-9.2 (or higher)
- Compile with: gcc -fopenmp -Wall -std=c99 -o maxnum netID-code.c

What to submit?

Add netID-code.c and netID-report.pdf to a single zip file netID.zip

How to submit? Through the assignment sections of Brightspace.

Have Fun!