Parallel Computing
OpenMP Programming Assignment

In this second lab you will write multithreaded code, with OpenMP, to generate prime numbers from M to N (inclusive) and test scalability and performance.

General notes:

- The program will be written in OpenMP.
- The name of the source code file is: genprimes.c
- You compile it as: gcc -Wall -O3 -std=c99 -fopenmp -o genprimes genprimes.c
- Write your program in such a way that to execute it I will type: *./genprimes M N t*
  Where:
  M is a positive number bigger than 2
  N is a positive number bigger than 2 and N > M (largest N is one billion)
  t is the number of threads and is a positive integer that does not exceed 64.
- The output of your program consists of three things:
  - A text file *N*.txt (N is the upper bound of the range that you will search for primes [M, N]).
  - The timing information (continue reading for more information about this one.).
  - You print the following statement on the screen:
    "The number of prime numbers found between *M* and *N* is *k* seconds.
    Time taken for the main part: x seconds"
    (insert correct values for M, N, k, and x)
- Assume we will not do any tricks with the input (i.e. We will not deliberately test your program with wrong values of M, N or t).

The format of the output file *N*.txt

- One prime per line
- For example, if M = 3 and N = 10 then the file 10.txt will look like:
  3
  5
  7

The algorithm for generating prime numbers:

There are many algorithms for generating prime numbers and for primality testing. Some are more efficient than others. For this lab, we will implement the following algorithm, even though there may be more efficient ones, but this one is easier to parallelize. Given M and N:
For each integer X in [M, N]:

- Check the divisibility of X by each number from 2 to floor(square root of x).
- If it is not divisible by any then the number is prime.
- You can optimize the check for the number by stop checking for primality of a number x if x is divisible by a factor. For example, to check 12, we must check whether it is divisible by numbers 2, 3, 4, 5, and 6. But as soon as we find it is divisible by 2, you can stop the check for 12 and conclude it is not prime. There are several other optimizations but implement only this one for this lab.

How to measure the execution time?
Your code will be doing three major parts:

- Read the input from the command line
- Generate the prime numbers (as indicated by the algorithm above)
- Write the output file, print the output message on the screen, and exit.

We care only about the timing of the middle part. Therefore, you do the following:

```
double tstart = 0.0, tend=0.0, ttaken;
Read the input from the command line
tstart = omp_get_wtime();
Create the threads.
Generate the prime numbers (as indicated by the algorithm above).
#pragma omp barrier // Explicit barrier
Threads terminated (except master thread)
ttaken = omp_get_wtime() - t_start;
Print the results on the screen as indicated earlier in this lab description.
Write the output file, print the result statement, and exit.
```

To help you check the correctness of your output, we are proving the file 1stmillion.txt that contains the first one million prime numbers. That file is not in the format required for you to follow though.

The report
Speedup will be calculated using the time measurement indicated above.

- After you implement the OpenMP version of the above algorithm, generate the following graphs:
  - **graph 1** (M= 10, N = 5000), y-axis is the speedup relative to the time of one thread; and x-axis is the number of threads: 2, 5, 10, 15, 20, and 25.
  - **graph 2** (M= 10, N = 5000,000), y-axis is the speedup relative to the time with one thread; and x-axis is the number of threads: 2, 5, 10, 15, 20, and 25.
- For each graph, explain the behavior that you see. This involves two things:
  - What is the pattern you see?
  - Why do you think we have this pattern?

What to submit:
A single zip file. The file name is your netID.zip
Inside that zip file you need to have:

- genprimes.c
- pdf file containing the graph and explanation.

The zip file is submitted through Brightspace as usual

Enjoy!