CSCI-GA.3033-009
Multicore Processors: Architecture & Programming
Lab# 2

In this second lab you will write parallel code, on CIMS machines, to generate prime numbers from 2 to N and test scalability and performance.

General notes:

- Program will be written in OpenMP.
- The name of the source code file is: genprime.c
- You compile it as: *>gcc -g -Wall -fopenmp -o genprime genprimes.c*
- Write your program in such a way that to execute it I will type: *./genprime N t*
  Where
  N is a positive number bigger than 2 and less than or equal to N
  t is the number of threads and is a positive integer that does not 100.
- We will not test your program with N larger than 100,000.
- The output of your program is a text file *N*.txt (N is the number entered as argument to your program).
- Assume we will not do any tricks with the input (i.e. We will not deliberately test your program with wrong values of N or t).

The format of the output file *N*.txt

- one prime per line
- each line has the format: a, b, c
  - a: the rank of the number (1 means the first prime)
  - b: the number itself
  - c: the interval from the previous prime number (i.e. current prime – previous prime).
- Assume the first line of the file to be: 1, 2, 0
- The second line will then be: 2, 3, 1
- and the third: 3, 5, 2
- and so on.

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, given N:

1. Generate all numbers from 2 to N.
2. First number is 2, so remove all numbers that are multiple of 2 (i.e. 4, 6, 8, … N). Do not remove the 2 itself.
3. Following number is 3, so remove all multiple of 3 that have not been removed from the previous step. That will be: 9, 15, … till you reach N.
4. The next number that has not been crossed so far is 5. So, remove all multiple of 5 that have not been crossed before, till you reach N.
5. Continue like this till floor((N+1)/2).
6. The remaining numbers are the prime numbers.

Example:

Suppose N = 20

floor of (20+1)/2 = 10 ← where we stop.

Initially we have:
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20

Let's cross all multiple of 2 (but leave 2):
2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, 9, ~~10~~, 11, ~~12~~, 13, ~~14~~, 15, ~~16~~, 17, ~~18~~, 19, ~~20~~

Next number is 3, so we cross all multiple of 3 that have not been crossed:
2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, 17, ~~18~~, 19, ~~20~~

Next number that has not been crossed is 5, so we will cross multiple of 5 (i.e. 10, 15, and 20). As you see below, they are all already crossed.
2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, 17, ~~18~~, 19, ~~20~~

Next number that has not been crossed is 7, so we will cross multiple of 7 (i.e. 14). As you see below, they are all already crossed.
2, 3, ~~4~~, 5, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~, 11, ~~12~~, 13, ~~14~~, ~~15~~, ~~16~~, 17, ~~18~~, 19, ~~20~~

The next number that has not been crossed is 11. This is bigger than 10, so we stop here.

The numbers that have not been crossed are the prime numbers:
2, 3, 5, 7, 11, 13, 17, 19

The file that your program generates is 20.txt and looks like:

```
1, 2, 0
2, 3, 1
3, 5, 2
4, 7, 2
5, 11, 4
6, 13, 2
7, 17, 4
8, 19, 2
```

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 and exit.

We care only about 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();
Generate the prime numbers (as indicated by the algorithm above)
ttaken = omp_get_wtime() - t_start;
printf("Time take for the main part: %f\n", ttaken);
Write the output file and exit.
```

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 (N = 1000), y-axis is the speedup relative to the running time with 1 thread; and x-axis is the number of threads: 2, 5, 10, 25, 50, and 100.
  - graph 2 (N = 10000), y-axis is the speedup relative to the running time with 1 thread; and x-axis is the number of threads: 2, 5, 10, 25, 50, and 100.
- For each graph, explain the behavior that you see. Do NOT explain what the curve looks like!

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

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

Email the zip file to the grader, with subject line: lab2 submission.

Enjoy!