CSCI-GA.3033-004
Graphics Processing Units (GPUs): Architecture and Programming
Programming Assignment 2

In this second lab you will write CUDA code to generate the prime numbers between 2 and N (inclusive) and test scalability and performance.

General notes:

- The name of the source code file is: genprimes.cu
- Write your program in such a way that to execute it I will type: *./genprimes N*
  Where N is a positive number bigger than 2 and less than or equal to 1000,000.
- The output of your program is a text file *N*.txt (N is the number entered as argument to your program).
  For example, if I type: ./genprimes 10
  The output must be a text file with the name 10.txt and that file contains: 2 3 5 7
  You can put a single space between each two numbers.
- You can assume that we will not do any tricks with the input (i.e. We will not deliberately test your program with wrong values of N, negative, float, non-numeric, or larger than 1000,000).
- As a way to help you, the following table contains the number of prime numbers below x.

x	number of primes
10	**4**
100	**25**
1,000	**168**
10,000	**1,229**
100,000	**9,592**
1,000,000	**78,498**
10,000,000	**664,579**

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 (only spaces between each two numbers, no commas):

2 3 5 7 11 13 17 19

How to measure the performance and scalability of your code?

To see how efficient your implementation is, you need to compare against a sequential version. Therefore, do the following:

1. Implement a C version called seqgenprimes.c that takes the same argument as the CUDA version. That is: ./seqgenprimes N will generate a text file N.txt that contains prime numbers between 2 and N.
2. Implement the CUDA version.
3. Run both program for the same input and measure the overall time using the *time* command in Linux. That is: *time ./genprimes N*
   Time will return 3 numbers: usr, system, and real. You will need to report the real.
4. Try with different values of N up to 1000,000 and see how the GPU version and CPU version compare. You may need to repeat the same experiment 5 times and take the median to get a more precise result.

The report

Write a report that contains the following:

- A list of all optimizations you have implemented in your GPU version
- A graph showing the speed of GPU version and CPU version (in milliseconds) for N= 100, 1000, 10000, 100000, and 1000000
- A bar with the same N values as x-axis and showing the speedup (CPU time / GPU time).
- Use nvprof to explain the results in the above two graphs. This means, do not say the CPU version is better than GPU version for small N because of the communication overhead. But show numbers on the communication overhead and how it increases with N. Are there any branch divergence? How is global memory access in gpu affecting performance? … etc. Correct analysis always means numbers not words!

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:

- seqgenprimes.c
- genprimes.cu
- pdf file containing the graphs and explanation.

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

Note: As a way to help you, we provide, on the course website, a list of the first 1 million prime numbers for you to check your implementation for correctness.

Enjoy!