CSCD 439/539 GPU Computing Lab1
Finding Primes on GPU

No Late Submissions are accepted.

Rules: Your code must use C and CUDA Language. If your program shows a compilation error, you get a zero for this lab assignment.

Submission: Wrap up all your **source files**, **a make file** and **a lab report** described at the end of this write-ups into a single zip file. Name your zip file as *FirstInitialYourLastName*Lab1.zip. For example, if your legal name is Will Smith, you should name your zip file as wsmithlab1.zip.

You are required to submit a make file along with all your source code. In the make file, please generate a target executable cscd439lab1.

To compile your source code **lab1.cu** (for an example) in current folder:

```
nvcc -arch=sm_35 -o lab1 lab1.cu
```

To run your executable: **./lab1**

Before you leave the laboratory, please show the TA or the instructor how your program works, they will give you a score for this Lab assignment.

For archive purpose, please also submit your single zip file on EWU Canvas by following CSCD439-01 Course →Assignments→Lab1→ Submit Assignment to upload your single zip file.

Problem Description:

Based upon the lecture regarding a prime finder program, you are required to implement a parallel version of the prime finder on GPU using CUDA C. You are required to implement these features.

1, The executable takes two command line arguments: **N** and **blockSize**, which means we like to find all prime numbers in the range of 3 to N inclusively. We ignore the edge cases of checking number of 0, 1 and 2 to simplify your program. Also, we like to launch a grid of threads on GPU that consists of ceil(**(N + 1) / 2.0** / **blockSize**) blocks. You have to explore and use the **long long int** type in CUDA.

2, The program returns an array of results, **result[i]** is either 0 (meaning number **i** is not a prime) or 1 (meaning number **i** is a prime).

3, You have to use the memory copy and allocation functions for GPU and Host CPU. After kernel is done, copy results from GPU global memory to CPU memory.

4, Each CUDA thread has to process only one number in the range 3 to N. Your CUDA threads have to skip the even numbers in the range of 3 to N, because we know that those are definitely not primes. In particular, thread 0 processes number 1 (use if to deactivate thread 0 here), thread

1 processes number 3, thread 2 processes number 5, thread 3 processes number 7, and thread 4 processes number 9,……

5, Time your sequential CPU code (provided in the class DemoCode 1 and 2 folder already), and your CUDA parallel execution cost. The parallel time cost has to include the memory setup cost(allocation and copy etc.), which has not been there in the sequential demo code.

6. You have to explore the usage of combination of __device__ and __host__ for the function int isPrime(long long int cur), because both host and GPU use that function.

7, **On CPU**, you have to sum up the **results** array returned by the GPU. Please output a message that shows the total number of primes found in the range of 3 to N, to verify the correctness of your program. **When you time your program, please exclude the time cost for this summation operation on CPU.**

8, Run your program on the lab machine, fill in the table below to compare with the sequential code. Some configuration parameters in the table below might be wrong. Please indicate which ones are wrong. (Hint, the maximal number of threads in one block is 1024. And maximal number of blocks for one-D grid is 65536, which may be different on the GPU server because it is of a newer generation.) What performance did you observe when configuration parameters were wrong?

a) Please use N=10,000,000, and blockSize as a variable

blockSize(in number of threads)	**64**	**128**	**256**	**512**	**1024**
GPU Time Cost (sec)	0.477	0.480	0.449	0.431	0423
Speedups (compared with sequential time cost)					
Sequential Time Cost on CPU (sec)					

b) Please use blockSize = 1024, and vary input **N** to fill in the table below.

N	**200,000**	**2,000,000**	**4,000,000**	**8,000,000**	**16,000,000**	**32,000,000**
GPU Time Cost (sec)						
CPU Time Cost (sec)						
Speedups						