CSCD 445/545 GPU Computing Lab1
Finding Primes on GPU

Rules: If you have skipped one or more classes without acceptable excuses, the instructor may refuse to answer any questions about the lab or homework assignment. I check attendance regularly.

Rules: Your code must use C and CUDA Language. If your program shows a compilation error, you get a zero for this lab assignment. Please verify/test that your code will be compiled and run on the GPU server.

Submission: Wrap up all your **source files**, **a make file** and **a lab report** that is detailed 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 lab1.

Please also submit your single zip file on EWU Canvas by following CSCD445 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 0 to N inclusively. 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 **unsigned long long int** type in CUDA.

On the top of your main.cu, you may write, *typedef unsigned long long bignum;* Then you can use ***bignum*** as a type in your code.

2, The program produces an array of results, **result[i]** is either 0 (meaning number **i** is not a prime) or 1 (meaning number **i** is a prime). Tricky part: before you use the d_result array on the device to save the result, **you will have to initialize all elements in d_result array to zero**. Think why?

```
cudaMemset(d_results, 0, arrSize);
```

3, You must 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 0 to N. Your CUDA threads have to skip the even numbers in the range of 0 to N, because we know that those are definitely

not primes (except for 2). In particular, thread 0 processes number 1 (you may assign thread 0 to process number 2, using *"if (id == 0) d_results[2] = 1"* in your kernel), 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 (timing.h and timing.c provided in the class Files->Demo->d01_prime0.zip), and your CUDA parallel execution cost. The parallel time cost has to include the GPU memory setup cost (allocation and copy etc.), which has not been there in the sequential demo code.

6. You may explore the usage of combination of __device__ and __host__ for the function int isPrime(bignum x), because both host and GPU use that function. I have provided the function in the space below. Please type in the code in your source file, (not copy and paste),

```
__host__ __device__ int isPrime(bignum x)
{
  if(x == 1)  return 0;
  if(x % 2 == 0 && x > 2) return 0;
  bignum i = 2, lim = (bignum) sqrt((float) x) + 1;
  for(; i< lim;  i++){
    if ( x % i == 0)
      return 0;
  }
  return 1;
}
```

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 0 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)					
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						

Three sample runs shown below,

```
ytian@csee-gpu01:~/gpu/labs/Lab1Solu2$ ./lab1 10000000 1024
Find all prime numbers in the range of 0 to 10000000...

Parallel code executiontime in seconds is 0.116456
Total number of primes found on the GPU in that range is: 664579.

Serial code executiontime in seconds is 3.543579

Total number of primes by CPU in that range is: 664579.

%%% The speedup(SerialTimeCost / ParallelTimeCost) when using GPU is 30.428472
%%% The efficiency(Speedup / NumProcessorCores) when using GPU is 7.607118
ytian@csee-gpu01:~/gpu/labs/Lab1Solu2$ ./lab1 1000 1024
Find all prime numbers in the range of 0 to 1000...

Parallel code executiontime in seconds is 0.065345
Total number of primes found on the GPU in that range is: 168.

Serial code executiontime in seconds is 0.000015

Total number of primes by CPU in that range is: 168.

%%% The speedup(SerialTimeCost / ParallelTimeCost) when using GPU is 0.000230
%%% The efficiency(Speedup / NumProcessorCores) when using GPU is 0.000057
ytian@csee-gpu01:~/gpu/labs/Lab1Solu2$ ./lab1 1000000 1024
Find all prime numbers in the range of 0 to 1000000...

Parallel code executiontime in seconds is 0.057924
Total number of primes found on the GPU in that range is: 78498.
```

Serial code executiontime in seconds is 0.140005

Total number of primes by CPU in that range is: 78498.

%%% The speedup(SerialTimeCost / ParallelTimeCost) when using GPU is 2.417057
%%% The efficiency(Speedup / NumProcessorCores) when using GPU is 0.604264