Parallel Computing
Lab Assignment 3

In this lab you will write CUDA code to count the numbers divisible by x in the range [2, y].

Description:

- The name of the source code file is divisible.cu
- Write your program in such a way that to execute it I type: *./divisible x y type*
  Where:
  y is a positive number bigger than or equal to 2 and less than or equal to 1,000,000 (one million).
  x is a positive integer less than or equal to 100,000
  type can be 0 or 1. If 0, execute a CPU only version. If 1, execute GPU version. This means you need to implement also a sequential version inside your code.
- Your program must find how many numbers are divisible by x in the range [2, y].
- The output of your program is a sentence printed on the screen as follows (substitute the correct numbers of M, x, and y as well as B and C):
- If the execution was for the GPU version:
  There are *M* numbers divisible by x in the range [2, y].
  Number of blocks used is B
  Number of threads per block is C
  If the execution was for the CPU version, then print only:
  There are *M* numbers divisible by x in the range [2, y].
- You decide on B and C. You may need to do some trial and error to find the combination that gives the best performance.
- The total number of threads must not exceed 5000,
  That is: (number of threads per block x number of blocks) <= 5000.
- 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).
- You compile with: **nvcc -o divisible divisible.cu**

How to implement that?

- If GPU version:
  - Allocate an array of char of y+1 elements in the host and initialize this array to 0.
  - Allocate a similar array in the device.
  - Copy the array of the host to the one on the device.
  - Decide how many blocks and threads per block are in the grid. Assume 1D grid and 1D blocks. Keep in mind that the number of threads per block and number of blocks depend on y.
  - Each thread in every block will be assigned one or more entries of the array to update.
  - The index of the array is the number the thread needs to explore whether it is divisible by x. For example, if the array is called A[], then A[15] means we

are exploring whether 15 is divisible by x. If 15 is divisible by x, then A[15] = 1. Otherwise, A[15] remains 0.
  - Once done, the array is copied back to the host.
  - The host counts how many 1s in the array and prints the result on the screen as shown earlier.
  - Free the allocated arrays both in host and device.

- If CPU version: [There are better ways to implement this. But we will stick with the following version to have the same algorithm as the GPU version.]
  - Call a function that loops through the elements of the array.
  - At each loop iteration, it checks whether the loop index is divisible by x.
  - If it is divisible by x, sets A[loop index] to 1.
  - Once done with the first loop, implement a second loop that goes through each element of array A and count the number of 1s.

Report:

We need to measure the time of the GPU version and CPU version for different problem sizes. For that, we will use the *real* part of the Linux *time* command as we say before.

Draw table 1 that has the following columns:
- x
- y
- number of threads per block
- number of blocks
- speedup = time of CPU/time of GPU

The rows are follows (You fill out the threads, blocks and speedups):
- row 1: x = 17, y = 1024
- row 2: x = 17, y = 4096
- row 3: x = 2, y = 1024
- row 4: x = 2, y = 4096

Then, answer the following questions:
1. What is the pattern you see as y increases?
2. Explain why you think this pattern arises.
3. Do you think x has an effect in performance for either CPU or GPU? Why?

What do you have to submit: divisible.cu file.

Important: You must do the test for divisibility in CUDA and execute it on the device. If the test for divisibility is done on the host, no points will be awarded.

Enjoy!