CSCD 439/539 GPU Computing Lab5
Global Communication and Atomic Operations

No Late Submissions are accepted. **Rules:** Your code must use C and CUDA C Language. If your program shows a compilation error, you get a zero for this lab assignment.

Submission: Wrap up all your **source files and other data files** into a single zip file. Name your zip file as *FirstInitialYourLastName*Lab5.zip. For example, if your legal name is Will Smith, you should name your zip file as wsmithlab5.zip. A simple makefile has been provided to compile your code into a target **lab5**.

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→Lab5→ Submit Assignment to upload your single zip file.

Problem Description:

Given an array of integer number **B**, normalizing **B** produces an array **Bn** with each element in float type, such that 0<=Bn[i]<=1. **Bn** is calculated by using equation **Bn[i] = B[i] / maxB**, where **maxB** is the maximal number in original array **B**.

Based on the lecture notes about global communication and atomic operations, you are required to implement the following features and answer the questions.

1, Write the first kernel that computes the global maximal number in the input dataset, you are required to divide the input array into several regions (simply using each block for each region works fine) and use an array **regional_max**. Threads corresponding to one region update only one corresponding element in the **regional_max** array. Please check out lecture notes for more details. The signature of this kernel is provided. Please follow the existing signature of this function.

2, Write the second kernel that actually normalizes the input array, by using the global maximum value returned in the kernel of step one. The normalized array is stored into an output array. The signature of this kernel is provided. Please follow the existing signature of this function.

3, In this program, how is the global synchronization (or communication) achieved ?

4, Can we combine kernel **global_max** and kernel **normalize** into one kernel? What are the difficulties if we try to do that?

5, How needed data is passed from first kernel into second kernel? Did we reallocate memory for these data or did we make any copy of these data?

6, What conclusion(s) you can draw with regards to passing data around during multiple kernel launches? After first kernel is done, we launch second kernel, is data in global memory automatically wiped out or presistent before second kernel launch?