ASSIGNMENT #1

CS300/CSD2170, Fall 2022

Due Date:	As specified on the moodle
Topics covered:	CUDA programming model, Execution model
Deliverables:	The submitted project file is the source code of CUDA file (kernel.cu) and cpu.cpp. The files should be put in a folder and subsequently zipped according to the stipulations set out in the course syllabus.
Objectives:	Learn how to use CUDA runtime API to write kernel functions.

Programming Statement

This is a CUDA C programming assignment. Students are expected to finish the programming (CUDA C/C++) for a given computation problem.

Problem Statement

In this assignment, we will write CUDA programs to determine a distribution in a 2D space using synchronous iteration on a GPU. Assume that there is a 2-dimensional square space with $n \times n$ points, where the boundary edge points have fixed values. The objective is to find the value distribution within the 2D space. The value of the interior points depends upon the values of the points around it. We can find the value distribution of the interior points by dividing the area into a fine mesh of points, $v_{i,j}$. The value at an inside point is calculated as the average of the values of the four neighboring points.

The edge points are the points (i,j) when $i = 0, i = n - 1, j = 0$, or $j = n - 1$, and have fixed values corresponding to the fixed values of the edges. The value at each interior point is calculated as:

$$v_{i,j}^{(K+1)} = \frac{v_{i-1,j}^{(K)} + v_{i+1,j}^{(K)} + v_{i,j-1}^{(K)} + v_{i,j+1}^{(K)}}{4} \tag{1}$$

where $0 < i < n - 1$; $0 < j < n - 1$, and K is the iteration number.

Stopping condition: You can have a fixed number of iterations or when the difference between two consecutive iterations (calculated as average among all points) is less than or equal to some number. For this assignment, we will use the fixed number of iterations.

Assume we have $n \times n$ points (including edge points). The edge points have the value of 26.67, except points ((0, 10) to (0, 30) inclusive) have the value of 65.56. Assume that all internal points are initialized to zero.

During the calculation of the distribution, you will first initialize the values of the inputs for all the $n \times n$ points (including edge points) by the following function

```
void initPoints(float *pointIn, float *pointOut, uint nRowPoints);
```

The CPU version for the calculation of the distribution is as follows:

```
heatDistrCPU(float *pointIn, float *pointOut, uint nRowPoints, uint nIter);
```

In GPU version, you will use the following function to calculate the distribution with the given inputs pointed by *d_DataIn*:

```
void heatDistrGPU(float *d_DataIn, float *d_DataOut, uint nRowPoints, uint nIter);
```

, where *d_DataOut* points to the outputs.

In the above function, you will firstly launch the calculation kernel

```
__global__ void heatDistrCalc(float *in, float *out, uint nRowPoints)
```

Once it is done, you need to call `cudaDeviceSynchronize()` to wait for all the threads in the grid to finish. Then, it is expected to use the outputs of K iteration calculation as the inputs in the $K + 1$ iteration calculation so you will call the following kernel to update

```
__global__ void heatDistrUpdate(float *in, float *out, uint nRowPoints)
```

You are required to turn in your homework that is compilable under the Windows environment using Visual Studio.

What to do

What you have to do:

1. Complete the sequential C version of the problem.
2. Write CUDA version of the above code.
3. Execute the programs in (1) and (2) with n = 100; 500; 1,000; 10,000. Assume 50 iterations.
4. Obtain the overall time taken (i.e. neither user nor system times). You may need to repeat the experiments 5 times and take the average to amortize any skew that may be caused by system load.
5. You may draw a bar-graph showing n (x-axis) and the time (y-axis) for the 2 versions of the program (2 bars per n on the same graph) to see the trend.
6. What are your conclusions regarding:
   - When is GPU usage more beneficial (at which n and why)?
   - When is the speedup (i.e. time of CPU version / time of CUDA version) at its lowest? And why?
   - When is the speedup at its highest? And why?
7. Repeat (4), (5), and 6 with 500 iterations.
8. What is the effect of increasing the number of iterations?

Grading Guideline

Please ensure that:

1. Functional Correctness: Produces correct result (CUDA C/C++ version is correct).
2. Coding:
   (a) Correct usage of CUDA library calls and C extensions.
   (b) Correct usage of thread id's in computation.

If your code fail to pass the test cases of $n = 50, 500, 1000, 10000$ for both 50 and 500 iterations, zero mark will be given. Your submission will be graded according to the test results. The latency is tested for the test cases of $n = 100$, 500, $1,000$ and $10,000$.