CSCE 4930: GPU Computing
Homework 3

This assignment should be completed by teams of 2 students. Only one submission per team is required.

You are required to implement and run all the tasks below on a computer having a CUDA-Enabled device. You can use your own computer or computers from the machine learning lab (Room# 2086). It is required to use NVIDIA's Nsight Eclipse Edition IDE (or NVIDIA's Nsight Visual Studio Edition IDE if you are using your own machine running Windows).

For all tasks you are required to do the following:

1. Write a base (sequential) function that implements the required functionality.
2. Write a CUDA-kernel and a wrapper function that initializes the necessary device data and calls that kernel to implements the same functionality. **Make sure to use error checking for all CUDA API calls**.
3. Write a full program that calls both functions on the same input data and verifies that they both produce the same output.
4. Time both versions and compute the speedup (or slowdown) obtained from parallelization. When timing the parallel version, you should do it in two different ways and compute the speedup in each case:
    a. Timing the kernel only
    b. Timing the entire wrapper function (including the memory allocation and data transfer overheads).
5. Your program should also print the performance of the sequential and parallel versions in GFLOPS
6. Manually compute the GPU device utilization

Task 1:

Write a program to perform 2D convolution on **greyscale** images of **arbitrary size**. Your program should support the following eight operations: blur, emboss, outline, sharpen, and sobel (left, right, top, and bottom). The image file name and the selected operation should be provided by the user. The program should then open the file converting it to a 2D array, apply convolution to it based on the selected mask, and save the result as a new image file. The parallel version of your program should use **tiling (using 8x8 tiles)** and **constant memory** and should rely on **general caching** for halo-cells. **Boundary conditions should be handled by replicating the edge values**.

Here are the masks to be used for each of the requested operations:

Blur: $\begin{bmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{bmatrix}$, Emboss: $\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$, Outline: $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$,

Sharpen: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$, Left Sobel: $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$, Right Sobel: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$,

Top Sobel: $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$, Bottom Sobel: $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

For a visual (interactive) explanation of the different convolution operations and their effect on image, you can check this URL: http://setosa.io/ev/image-kernels/

You are free to use any C library to open and save image files. One such library, libwb, can be found here: https://github.com/abduld/libwb
This library supports the Netpbm Portable Pixel Map (PPM) file format only and uses the **`wbImage_t struct`** described in slide 15 of lecture 17. To use the library, you have to compile it using the following sequence of Linux commands:

```
cd [path_to_libwb]
mkdir build
cd build
cmake ..
make
```

You also need to include the library's main header file in your program using the directive:
`#include "wb.h"`
Additionally, you need to modify the LD_LIBRARY_PATH in your Linux environment to include the path of your libwb.so. That should be the path to the build folder created above.
Finally, you will need to set the project properties in Nsight, to be able to locate, the library headers, the library path, and the library itself.

Here are some functions you might find useful in the wb library:

- **`wbImage_t wbImage_new(int height, int width, int channels);`**
- **`void wbImage_delete(wbImage_t img);`**

- **`wbImage_t wbImport(char * filename);`**
- **`void wbExport(char * filename, wbImage_t img);`**

- **`int wbImage_getWidth(wbImage_t img);`**
- **`int wbImage_getHeight(wbImage_t img);`**
- **`int wbImage_getChannels(wbImage_t img);`**
- **`int wbImage_getPitch(wbImage_t img);`**
- **`float *wbImage_getData(wbImage_t img);`**

Task 2:

Repeat Task 1 using tile sizes of 16x16 and 32x32.

Task 3:

Write a program to compute the **total intensity of one or more rectangular subareas of a greyscale image of arbitrary size**. The program should use the **summed-area table** (aka **integral image**) to do this computation in constant time. The image file name and the coordinates of the pixels delimiting the rectangular subareas should be provided by the user. The program should then open the file converting it to a 2D array, compute the corresponding summed-area table, and finally use the obtained table to quickly compute the total intensity of the areas specified by the user.

The summed-area table is a 2D array **S** of the same size as the original image 2D array **I** such that the value of each **S** element at a position (x,y) is the sum of the intensities of all pixels in **I** with lower coordinates (i.e., all the pixels above and to the left of the pixel at (x,y)).

$$S(x,y) = \sum_{\substack{x' \le x \\ y' \le y}} I(x',y')$$

So, the summed-area table computation can be seen as a 2D extension of the prefix-sum (scan) computation. As such you should use a 2D extension of the hierarchal approach studied in the lecture.

So, if the original image I = $\begin{bmatrix} 10 & 50 & 100 & 0 \\ 50 & 255 & 35 & 0 \\ 0 & 16 & 10 & 80 \\ 10 & 20 & 200 & 150 \end{bmatrix}$, the summed-are table S will be $\begin{bmatrix} 10 & 60 & 160 & 160 \\ 60 & 365 & 500 & 500 \\ 60 & 381 & 526 & 606 \\ 70 & 411 & 756 & 986 \end{bmatrix}$

To find the total intensities in a rectangular area delimited by the points pixels A (x_1,y_1), B (x_2,y_1) , C(x_1,y_2) and D (x_2,y_2), we can do this in constant time using the formula:

$$\sum_{\substack{x_1 < x \le x_2 \\ y_1 < y \le y_2}} I(x,y) = S(D) + S(A) - S(B) - S(C)$$

For more information about summed-area tables, check this URL:
https://computersciencesource.wordpress.com/2010/09/03/computer-vision-the-integral-image/

Task 4:

Write a program that generates a random square matrix A of size NxN with mostly zeros, and two vectors X and Y of size N. The sparsity of the matrix (i.e., the ratio of zero to non-zero elements) should be specified by the user. Your program should convert the matrix A to the ELL-sectioned JDS format and then compute the SpMV operation on it. The conversion itself should be parallelized. Here is a suggested strategy for the conversion:

1. Using the GPU device, count the number of non-zero elements per row.
2. Using the CPU host, sort the rows based on the results returned by the GPU in step 1. Make sure to generate the row re-order indices.
3. Using the GPU device, convert each section of rows into the ELL format.

Once the conversion is done, use the GPU device to make the SpMV computation.

- **This assignment's weight is equivalent to 2 assignments** (40 marks instead of 20 marks). Tasks 1 and 2 are considered one assignment, while tasks 3 and 4 are considered a second assignment.
- Submission Deadline: Sunday, December 8, 2019 11:59 PM.
- Submission method: Electronically through BB.
- Your submission should be **a single compressed folder (zip)** containing a PDF report including output screenshots, timing, and utilization results. The compressed folder should also contain a source code subfolder for each task.
- Check the readability of your submission. Hard to read submissions will be ignored.
- Plagiarism is NOT tolerated. Late submissions are NOT accepted.