COMP3231 Computer Architecture

Programming Assignment One

There are two parts in this assignment. To guide your coding, a partially completed program is provided to you for each part, which provides implementations for reading an input image (in .txt format), measuring the elapsed time of running your program code (CPU and GPU implementation), and saving the result to a file.

Please read the instruction below and code template carefully. You may refer to tutorial slides. **Do not search for existing GPU convolution code on the internet**.

Part 1: Sharpening a greyscale image (part1.cu)

In this part, you are going to write a program to sharpen a 2D greyscale image using a sharpening filter, which should be done by convoluting 2D 3x3 sharpen filter to the image. The 2D 3x3 sharpen filter used in this assignment is as follows (i.e., you can hard-code the filter in part 1):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Convolution is the process of adding each element of the image to its local neighbors, weighted by the filter. In the process of convolution, the center of the filter is moved from pixel to pixel in the image.

The following is an example showing the formula of convolution. (Note that the kernel should sum to 1; otherwise, you need to normalize the kernel)

1	2	3
4	5	6
7	8	9

Input

-1	-2	-1
0	0	0
1	2	1

Kernel

-13	-20	-17
-18	-24	-18
13	20	17

Output

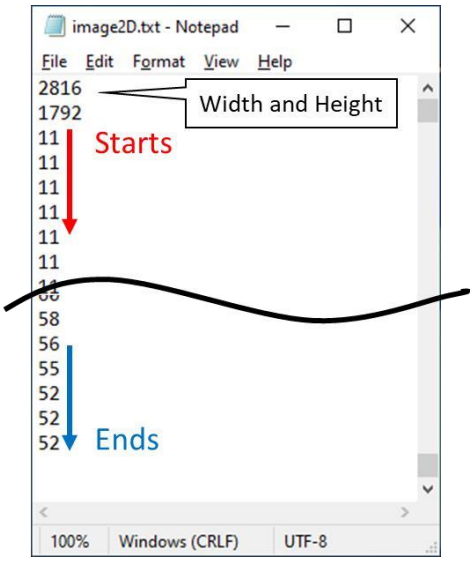
The ouput at (1, 1) for this example will be;

$$\begin{aligned} y[1,1] &= \sum_j \sum_i x[i,j] \cdot h[1-i, 1-j] \\ &= \quad x[0,0] \cdot h[1,1] \quad + x[1,0] \cdot h[0,1] \quad + x[2,0] \cdot h[-1,1] \\ &\quad + x[0,1] \cdot h[1,0] \quad + x[1,1] \cdot h[0,0] \quad + x[2,1] \cdot h[-1,0] \\ &\quad + x[0,2] \cdot h[1,-1] + x[1,2] \cdot h[0,-1] + x[2,2] \cdot h[-1,-1] \\ &= \quad 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 1 \\ &\quad + 4 \cdot 0 + 5 \cdot 0 + 6 \cdot 0 \\ &\quad + 7 \cdot (-1) + 8 \cdot (-2) + 9 \cdot (-1) \\ &= -24 \end{aligned}$$

We use zero padding for computing the values at borders (i.e., when computing the value of a pixel at the border of the image, the values for its ``neighbor'' pixels out of the image are treated as zero).

A detailed example can be found in the following link:
http://www.songho.ca/dsp/convolution/convolution2d_example.html

The input is a greyscale 2D image. The sample image is defined in image-grey.txt. The first 2 lines indicate the width and height of the image; the pixel values are flattened into a column and appended starting from the third lines.



Your output should have exactly the same format as the input (i.e., the first two columns and the number of lines should be identical as the input).

Part 2: 2D smooth (part2.cu)

In this part, you are going to write another program to smooth a 2D **colored** image using a Gaussian filter. Like part1, you should include serial code for CPU and parallel code for GPU in your program.

For this part, image smoothing should be done by convoluting a 2D gaussian filter to the image. A Gaussian filter has the most massive weight in the center and less weights far from the center. The size of the filter is defined in the code template by const int FILTER_WIDTH. The actual 2D filter is flattened into a 1D array and defined by int FILTER[]

Unlike part 1, your code should work for different filter sizes.

The input has similar format as part 1, but each line has three values, representing the R, G, B values (channel) of the pixel. The filter is applied to each channel independently, but your code should compute them in parallel.

Others:

- We provide python utility code for you to debug your program and visualize your output. Usage of the utility code is not necessary.
- For simplicity, use integer for intermediate results during computation and use integer division (3/2=1).
- Your GPU code's kernel time must be shorter than the CPU code's time. (Don't add dummy code to your CPU code. We will find out.)
- Both the correctness and the GPU code's performance are considered for marking.
    - We will run your program 10 times and only consider the best GPU time you achieved.

Coding and marking environment:

Please make sure that your program can compile, execute, and generate the required outputs on the standard environment, the CS GPU Farm. While you may develop your work on your own environment, you should always try your program (compile, execute and check results) on our standard environment before submission.

Submission:

Please compress completed part1.cu and part2.cu into a .zip file and submit it to Moodle.

You can include a README file describing the features you have implemented especially if you have turned in a partially finished implementation. You can also record down the best elapsed time you can achieve for our reference.

Late submission policy: See Slides #0

- based on your moodle submission time, not your file's last modified time on your computer. ``I have finished the assignment but forgot to submit'' it not accepted.

-END-