Assignment #4

CS 398 GPGPU Programming, Fall 2020

Due Date:	As specified on the moodle
Topics covered:	Atomic Operation, Shared Memory, Histogram, Scan, Reduction
Deliverables:	The project files (release version only) including source code and the report `A4Report.pdf`. These files should be put in a folder and subsequently zipped according to the stipulations set out in the course syllabus.
Objectives:	Implementation of histogram equalization in CUDA C. Learn how to use shared memory, atomic operation, histogram and scan pattern to write more efficient kernel functions.

Objectives

This assignment is the implementation of image processing routines in CUDA C. For your reference, a C implementation are provided.

Implementation Requirement

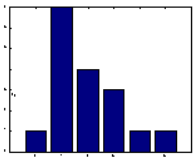
Image Adjustment. Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

An image histogram is a graphic representation of the frequency counts of all allowable pixel intensities. Figure 1 shows a sample of an image with a skewed histogram.

4	1	3	2
3	1	1	1
0	1	5	2
1	1	2	2

input image

(a) Image Pixel Value



(b) Histogram

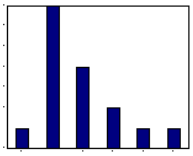
Figure 1: A sample image with a skewed histogram (poor intensity distribution)

Images with skewed distributions can be helped with histogram equalization as shown in Figure 2.

5	3	4	4
4	3	3	3
1	3	5	4
3	3	4	4

output image

(a) Image Pixel Value



(b) Histogram

Figure 2: A histogram-equalized sample with histogram and normalized sum

intensity	sum	normalized sum	cdf
0	1	1/16*5=0.31255	1/16
1	8	8/16*5=2.5	8/16
2	12	12/16*5=3.75	12/16
3	14	14/16*5=4.375	14/16
4	15	15/16*5=4.6875	15/16
5	16	16/16*5=5.0	16/16

Table 1: Normalized Sum

Histogram equalization is a point process that redistributes the image's intensity distributions in order to obtain a uniform histogram for the image. Histogram equalization can be done in a few steps:

1. Create the histogram for the image.
2. Calculate the cumulative distribution function histogram (normalized sum).
3. Calculate the new values through the general histogram equalization formula.
4. Assign new values for each gray value in the image.

Table 1 shows the normalized sum of the image in Figure 1, where *sum* is the cumulative value for each intensity and normalized sum is obtained after dividing *sum* by total number of pixels, i.e. 16, and multiplying it with the maximum intensity value, i.e. 5, in this example. The histogram-equalized image, and its histogram are shown in Figure 2. Note that the resulting histogram is not truly uniform, but it is better distributed than before. Truly uniformed histograms for discrete images are difficult to obtained because of quantization.

The pseudo-code of the histogram equalization code is as follows:

```
for grey_level = 0; grey_level <= 255; grey_level++
  images_histogram[grey_level] = 0;
  for i = 0; i < row_size; i++
    for j = 0; j < col_size; j++
      if dataIn(i,j) == grey_level
        images_histogram[grey_level] = images_histogram[grey_level] + 1;
```

```
      end
    end
  end
end
```

where `dataIn` is the array of pixel values and `images_histogram` is the array of histogram.

In order to implement this on the GPU, an image histogram function is needed. Note that you have to use the 256 level method. In addition, the calculation of CDF (and therefore the minimum and maximum value of the pixel intensities) can be carried out on GPU.

To obtain the sum in Table 1, you need to perform a prefix-sum (i.e. scan pattern as taught in the class) for the pixel intensity (i.e. histogram). The actual assignment of new values for each gray value (pixel intensity) in the image, instead of using normalized sum as shown in Table 1, follows the following equation:

$$CLAMP(255 * ((cdfVal) - (cdfMin))/(1 - (cdfMin)), 0.0, 255.0) \quad (1)$$

where $cdfVal$ is the prefix-sum of histogram bin value divided by the total number of pixels. $cdfVal$ is the cumulative value (i.e. cdf in Table 1) for the original pixel value, and $cdfMin$ is the minimum cumulative value (e.g. the cdf value at the first row in Table 1) and $CLAMP(x, start, end)$ is defined as $min(max((x), (start)), (end)))$. To obtain the new value for each pixel, the minimum value of pixel intensity $cdfMin$ is also required, as shown in Eq. 1. Please refer to the reference C code for the details.

The following figures show the image before equalization and after equalization.

Before equalization

After equalization

Modes of Operations

Given an input image name, read it from file, perform image adjustment, and finally save the output to a file. The following command line syntax should be used:

```
Your_program input_file output_cpu_file output_gpu_file
```

where $input_file$ is the input image file (pgm format), $output\_cpu\_file$ and $output\_gpu\_file$ are the output files (ppm format) for CPU version and GPU version respectively. The input file format is PGM. The C functions (read/write) for PGM/PPM file have been included for your reference.

1. You can assume that the input image is no bigger than 16Megapixels (e.g., 4096×4096);
2. Note that while there is glHistogram extention in OpenGL, it is usually NOT hardware accelerated.

Report Requirement

In the report you need to compare your CUDA and CPU implementation. More specifically, you need to provide timing information in the following table

	CPU		CUDA	
Image Size	A	B	A	B
512×512				
1024×1024				
2048×2048				

Where A represents the image-histogram-equalization time and B represents the image adjustment time. Future more, for GPU-based implementation, calculate the time to download and read-back the image data. Report the total time for all test cases. Discuss any tuning/optimization you have done to improve the GPU performance. Discuss the advantage or disadvantage of using CUDA for image processing tasks.

Grading Guideline

Your submission will be graded on the following parameters.

1. Functionality: 50%
   (a) Correct CUDA implementation (40%)
   (b) CUDA Implementation tuning (10%)
2. Performance and report: 30%. The speedup is defined as time of CPU version / time of CUDA version). Close to at least 200% speedup for 512×512, 400% speedup for $1,024 \times 1,024$, and 500% speedup for $2,048 \times 2,048$ on the computers with GeForce GTX1060 6GB in Tesla lab is required for full credit here.
3. Answer the following questions regarding to histogram (10%), (for 512×512 image)
   (a) Describe all optimizations you tried regardless of whether you committed to them or abandoned them and whether they improved or hurt performance.
   (b) Were there any difficulties you had with completing the optimization correctly.
   (c) Which optimizations gave the most benifit.
   (d) For the histogram kernel, how many global memory reads are being performed by your kernel? explain.
   (e) For the histogram kernel, how many global memory writes are being performed by your kernel? explain.
   (f) For the histogram kernel, how many atomic operations are being performed by your kernel? explain.

(g) For the histogram kernel, what contentions would you expect if every element in the array has the same value?

(h) For the histogram kernel, what contentions would you expect if every element in the input array has a random value?

4. Answer the following questions regarding to scan (10%), (for 256 bin histogram)

(a) How many floating operations are being performed in your reduction kernel? EXPLAIN.

(b) How many global memory reads are being performed by your kernel? EXPLAIN.

(c) How many global memory writes are being performed by your kernel? EXPLAIN.

(d) What is the minimum, maximum, and average number of real operations that a thread will perform? Real operations are those that directly contribute to the final reduction value.

(e) How many times does a single thread block synchronize to reduce its portion of the array to a single value?

(f) Suppose a you want to scan using a binary operator that's not commutative, can you use a parallel scan for that?

(g) Is it possible to get different results from running the serial version and parallel version of scan? EXPLAIN.