Assignment #3

Programming Massively Parallel Processors

Due Date:	As specified on the Moodle
Topics covered:	CUDA programming, Atomic Operation, Shared Memory, Histogram, Scan, Reduction
Deliverables:	The submitted project files are the source code of your CUDA program (kernel.cu). The 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 program. Learn how to use shared memory, atomic operation, histogram and scan pattern to write more efficient CUDA program.

1 Objectives

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

```
1 /*
2 * Copyright 2026.  All rights reserved.
3 *
4 * Author: William Zheng
5 *
6 * Please refer to the end user license associated
7 * with this source code for terms and conditions that govern
     your use of
8 * this software. Any use, reproduction, disclosure, or
     distribution of
9 * this software and related documentation outside the terms
10 * is strictly prohibited.
11 *
12 */
13
14 #include <assert.h>
15 #include "histogram_common.h"
16
17 //assume inRGB is RGB model
18 // dataYUV is for YUV model
19 // inRGB, dataYUV, outRGB : non-interleaving layout
20 // histo, histoCdf : for Y component only
21
22 extern "C" void histogram256CPU(
23     uint* histo,
24     float* histoCdf,
```

```
25      uchar* inRGB,
26      float* dataYUV,
27      uchar* outRGB,
28      uint imgWidth,
29      uint imgHeight,
30      uint imgChannels
31  )
32  {
33      // Validate inputs
34      assert(imgChannels >= 3);
35
36      //initialize histogram bin and cdf values
37      for (int i = 0; i < HISTOGRAM256_BIN_COUNT; i++) {
38          histo[i] = 0;
39          histoCdf[i] = 0.0f;
40      }
41
42      int imgSz = imgWidth * imgHeight;
43      float totalPixels = (float)(imgWidth * imgHeight);
44
45      //RGB to YUV conversion
46      for (unsigned int row = 0; row < imgHeight; row++) {
47          for (unsigned int col = 0; col < imgWidth; col++) {
48              int i = row * imgWidth + col;
49
50              // Convert RGB to YUV (BT.601 standard)
51              float r = (float)inRGB[i];
52              float g = (float)inRGB[i + imgSz];
53              float b = (float)inRGB[i + 2 * imgSz];
54
55              float y = 0.299f * r + 0.587f * g + 0.114f * b;
56              float u = -0.169f * r - 0.331f * g + 0.499f * b +
                    128.0f;
57              float v = 0.499f * r - 0.418f * g - 0.0813f * b +
                    128.0f;
58
59              dataYUV[i] = CLAMP(y, 0.0f, 255.0f);
60              dataYUV[i + imgSz] = CLAMP(u, 0.0f, 255.0f);
61              dataYUV[i + 2 * imgSz] = CLAMP(v, 0.0f, 255.0f);
62          }
63      }
64
65      //histogramming on Y component with proper binning
66      for (unsigned int row = 0; row < imgHeight; row++) {
67          for (unsigned int col = 0; col < imgWidth; col++) {
68              int i = row * imgWidth + col;
69
```

```
70              // Use proper rounding to nearest integer for bin
                    index
71              float y_val = dataYUV[i];
72              int bin = (int)(y_val + 0.5f); // Round to nearest
                    integer
73              bin = CLAMP(bin, 0, HISTOGRAM256_BIN_COUNT - 1);
74              histo[bin]++;
75          }
76      }
77
78      //cdf calculation with proper probability
79      histoCdf[0] = (float)histo[0] / totalPixels;
80      for (int i = 1; i < HISTOGRAM256_BIN_COUNT; i++) {
81          float pdf = (float)histo[i] / totalPixels;
82          histoCdf[i] = pdf + histoCdf[i - 1];
83      }
84
85      float cdfMin = histoCdf[0];
86
87      // Apply histogram equalization
88      for (unsigned int row = 0; row < imgHeight; row++) {
89          for (unsigned int col = 0; col < imgWidth; col++) {
90              int i = row * imgWidth + col;
91
92              // Get bin index with proper rounding
93              float y_val = dataYUV[i];
94              int bin = (int)(y_val + 0.5f);
95              bin = CLAMP(bin, 0, HISTOGRAM256_BIN_COUNT - 1);
96
97              // Apply histogram equalization formula
98              float equalized_y;
99
100             equalized_y = 255.0f * (histoCdf[bin] - cdfMin) /
                    (1.0f - cdfMin);
101
102             equalized_y = CLAMP(equalized_y, 0.0f, 255.0f);
103
104             // Get UV components
105             float u = dataYUV[i + imgSz] - 128.0f;
106             float v = dataYUV[i + 2 * imgSz] - 128.0f;
107
108             // Convert YUV back to RGB
109             float r = equalized_y + 1.402f * v;
110             float g = equalized_y - 0.344f * u - 0.714f * v;
111             float b = equalized_y + 1.772f * u;
112
113             // Store results
114             outRGB[i] = (uchar)CLAMP(r, 0.0f, 255.0f);
```

```
115                 outRGB[i + imgSz] = (uchar)CLAMP(g, 0.0f, 255.0f);
116                 outRGB[i + 2 * imgSz] = (uchar)CLAMP(b, 0.0f,
                        255.0f);
117             }
118         }
119 }
```

histogram_cpu.cpp

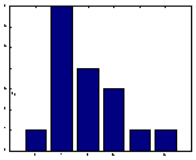
2 Implementation Requirement

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.

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

input image

(a) Image Pixel Value



(b) Histogram

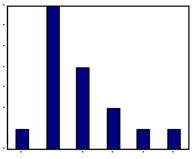
Figure 1: A sample image with a skewed histogram (poor intensity distribution)

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

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:

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

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 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.

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 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.

3 YUV Model

Histogram equalization is a non-linear process. Channel splitting and equalizing each channel separately is incorrect. Equalization involves intensity values of the image, not the color components. So for a simple RGB color image, histogram equalization cannot be applied

directly on the channels. It needs to be applied in such a way that the intensity values are equalized without disturbing the color balance of the image. So, the first step is to convert the color space of the image from RGB into one of the color spaces that separates intensity values from color components. Some of the possible options are HSV/HLS, YUV, YCbCr, etc. In this assignment, we assume that the conversion between YUV and RGB models are as follows:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.499 \\ 0.499 & -0.418 & -0.0813 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{2}$$

and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.344 & -0.714 \\ 1 & 1.772 & 0 \end{bmatrix} \times \begin{bmatrix} Y' \\ U - 128 \\ V - 128 \end{bmatrix} \tag{3}$$

In the CUDA program, you need to convert the input of RGB model into YUV model based on Eq 2 before you start histogramming on the component of Y in YUV model. In the program, you use the computed values of Y to index the generated CDF values to obtain the values of Y' and convert Y'UV to RGB model at the end using Eq. 3.

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
output_cpu_histogram_file output_gpu_histogram_file output_cpu_cdf_file
```

where $input\_file$ is the input image file (BMP format), $output\_cpu\_file$ and $output\_gpu\_file$ are the output files (BMP format) for CPU version and GPU version respectively. The input file format is BMP. The C functions (read/write) for BMP 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.
3. Ensuring exact floating-point consistency between CPU and GPU is challenging due to inherent architectural and compiler differences. The best approach is to configure both environments to use the same default rounding mode and manage potential compiler optimizations that cause discrepancies.
    - Use the Same Rounding Mode (Round-to-Nearest-Even): Both CUDA and standard C/C++ typically default to the round-to-nearest-even mode (IEEE 754 standard).
    - Control Fused Multiply-Add (FMA) Behavior FMA instructions perform a*b+c with a single rounding step, which is faster and often more accurate but produces results that differ from separate a*b then +c operations.

  - To match CPU behavior (less common): Disable FMA on the GPU by passing the compiler flag -fmad=false to nvcc. This will likely reduce performance and accuracy on the GPU.[1]
  - To match GPU behavior (more common): Ensure your CPU compiler settings enable FMA optimization and link against an FMA-aware math library, if available, though matching GPU FMA behavior on the CPU side is more difficult to force consistently.[2]
- Manage Compiler Flags and Optimization Levels Different optimization levels can affect the order of operations, leading to different results. Use similar optimization levels for both CPU and GPU code where possible. Be aware of other CUDA compiler flags like -ftz (flush denormals to zero), -prec-div, and -prec-sqrt, and ensure equivalent settings are used on the host.
- Employ Numerical Tolerances for Comparisons Due to the inherent non-associative nature of floating-point arithmetic and subtle hardware differences, it is almost impossible to guarantee bit-for-bit identical results.

4 Rubrics

This assignment will be graded over 100 points. Here is the breakdown:

- If your code fails to compile with the given template, zero is awarded immediately.
- If you did not use shared memory, atomic operation, histogram (privatization) and scan (reduction and post-reduction two phases) to implement the histogram equalization, zero is awarded immediately.
- Comments/Code Readability. Comments are important for the code to enhance readability. Up to 10 points will be deducted for insufficient comments.
- Correctness. You must ensure that the code works for images of all sizes. At least pass the test for 512×512 picture. Pictures with different sizes could be used in the evaluation.
- Latency. The speedup is defined as time of CPU version / time of CUDA version). Close to at least 150% speedup for 512×512.

The lecturer reserves the right to impose reasonable penalties for code that violates general practices or does not match the specification in an obvious way that has not been mentioned above. In exceptional cases, the lecturer reserves a discretionary right to allow resubmission or submission after the deadline.

[1] In the Visual Studio IDE, go to Project Properties > CUDA C/C++ > Command Line. Add compiler flag -fmad=false as Additional Options.

[2] In the Visual Studio IDE, go to Project Properties > Configuration Properties > C/C++ > Code Generation > Floating Point Model. Setting this to /fp:fast enables optimizations. Ensure the target architecture supports FMA: Go to Project Properties > Configuration Properties > C/C++ > Code Generation > Enable Enhanced Instruction Set and select an option like Advanced Vector Extensions 512 (X86/X64) (/arch:AVX512) or other AVX2 options, as AVX2 introduced FMA support.