# (/mp/59811?show=code) Histogram Attempt

## Attempt Summary

Submit Attempt for Grading

Remember to answer the questions before clicking.

| | |
|---|---|
| Dataset Id: | 5 |
| Created: | less than a minute ago (2022-03-21 03:29:56 +0000 UTC) |
| Status: | Correct solution for this dataset. |

## Timer Output

| Kind | Location | Time (ms) | Message |
|---|---|---|---|
| **Generic** | main.cu::167 | 9.182929 | Importing data and creating memory on host |

## Program Run Standard Output

```
1024, 683, 3
```

## Program Code

```
1  // Histogram Equalization
```

```
2
3  #include <wb.h>
4
5  #define wbCheck(stmt)
6    do {
7      cudaError_t err = stmt;
8      if (err != cudaSuccess) {
9        wbLog(ERROR, "Failed to run stmt ", #stmt);
10       wbLog(ERROR, "Got CUDA error ...  ", cudaGetErrorString(err));
11       return -1;
12     }
13   } while (0)
14
15
16 #define HISTOGRAM_LENGTH 256
17 #define chan_count 3
18 //implementation-independent code piece for unsigned char
19 typedef unsigned char uint8_t;
20 typedef unsigned int  uint_t;
21 #define TILE_WIDTH 32
22 #define RGB_MAX 255.0
23
24
25 /*
26 int next = (numAColumns + BLOCK_SIZE - 1) / BLOCK_SIZE;
27 float hold = 0;
28 */
29 //For the CUDA Kernels
30 //Here I cast the image to an unsigned char:
31 //  PARTS SIMILAR TO SECTIONS FROM MP3
32
33 __global__ void float_to_uint8_t(float *input, uint8_t *output, int width
34   int x = (blockIdx.x * blockDim.x) + threadIdx.x;
35   int y = (blockIdx.y * blockDim.y) + threadIdx.y;
36
37   if (x < width && y < height){
38     int idx = blockIdx.z * (width * height) + y * (width) + x;
39     output[idx] = (uint8_t) ((HISTOGRAM_LENGTH - 1) * input[idx]);
40   }
41 }
42
43 //convert an input image from RGB color scale to grayscale
44 __global__ void color_to_dark(uint8_t *input, uint8_t *output, int width,
45   int x = (blockIdx.x * blockDim.x) + threadIdx.x;
46   int y = (blockIdx.y * blockDim.y) + threadIdx.y;
```

```
47
48    if (x < width && y < height){
49      int idx = y * (width) + x;
50      uint8_t R = input[3 * idx + 0];
51      uint8_t G = input[3 * idx + 1];
52      uint8_t B = input[3 * idx + 2];
53      output[idx] = (uint8_t) (0.07*B + 0.71*G + 0.21*R);
54    }
55  }
56
57  //Get a histogram of the image
58  __global__ void dark_to_graph(uint8_t *input, uint_t *output, int width,
59    __shared__ uint_t histogram[HISTOGRAM_LENGTH];
60
61    int index_threads = threadIdx.x + threadIdx.y * blockDim.x;
62    if (index_threads < HISTOGRAM_LENGTH) {
63      histogram[index_threads] = 0;
64    }
65
66    __syncthreads();
67    int x = blockIdx.x * blockDim.x + threadIdx.x;
68    int y = blockIdx.y * blockDim.y + threadIdx.y;
69    if (x < width && y < height) {
70      int idx = y * (width) + x;
71      uint8_t val = input[idx];
72      //utilize atomic add function
73      //B.14. Atomic Functions
74      atomicAdd(&(histogram[val]), 1);
75    }
76
77    __syncthreads();
78    if (index_threads < HISTOGRAM_LENGTH) {
79      atomicAdd(&(output[index_threads]), histogram[index_threads]);
80    }
81  }
82
83  //Compute the scan and prefix sum of the histogram to arrive at the histo
84  //We get a scan of histogram -> histogram equalization function
85  // Cumulative Distribution Function @ https://www.cs.umd.edu/class/fall20
86  // >> Brent-Kung derivived parallel inclusive scan algorithm
87  // >> http://www.sci.utah.edu/~acoste/uou/Image/project1/Arthur_COSTE_Pro
88  __global__ void scan_to_stat(uint_t *input, float *output, int width, int
89    __shared__ uint_t cmlt_dist_func[HISTOGRAM_LENGTH];
90    int x = threadIdx.x;
91    cmlt_dist_func[x] = input[x];
```

```
92
93    //Scan pt-1
94    for (unsigned int scanner = 1; scanner <= HISTOGRAM_LENGTH / 2; scanner
95      __syncthreads();
96      int idx = (x + 1) * 2 * scanner - 1;
97      if (idx < HISTOGRAM_LENGTH) {
98        cmlt_dist_func[idx] += cmlt_dist_func[idx - scanner];
99      }
100   }
101   //Scan pt-2
102   for (int scanner = HISTOGRAM_LENGTH / 4; scanner > 0; scanner /= 2) {
103     __syncthreads();
104     int idx = (x + 1) * 2 * scanner - 1;
105     if (idx + scanner < HISTOGRAM_LENGTH) {
106       cmlt_dist_func[idx + scanner] += cmlt_dist_func[idx];
107     }
108   }
109   __syncthreads();
110   output[x] = cmlt_dist_func[x] / ((float) (width * height));
111 }
112
113
114 //Apply the histogram equalization function
115 //get color corrected image from input image
116 __global__ void equal_func(uint8_t *shift, float *cmlt_dist_func, int wid
117   int x = blockIdx.x * blockDim.x + threadIdx.x;
118   int y = blockIdx.y * blockDim.y + threadIdx.y;
119
120   if (x < width && y < height) {
121     int idx = blockIdx.z * (width * height) + y * (width) + x;
122     uint8_t val = shift[idx];
123
124     float equalized = 255 * (cmlt_dist_func[val] - cmlt_dist_func[0]) / (
125     float clamped   = min(max(equalized, 0.0), 255.0);
126
127     shift[idx] = (uint8_t) (clamped);
128   }
129 }
130
131 //Cast back to float
132 __global__ void uint8_t_float(uint8_t *input, float *output, int width, i
133
134   int x = blockIdx.x * blockDim.x + threadIdx.x;
135   int y = blockIdx.y * blockDim.y + threadIdx.y;
136
```

```
  if (x < width && y < height) {
    int idx = blockIdx.z * (width * height) + y * (width) + x;
    output[idx] = (float) (input[idx] / 255.0);
  }
}

//@@ insert code here

int main(int argc, char **argv) {
  wbArg_t args;
  int imageWidth;
  int imageHeight;
  int imageChannels;
  wbImage_t inputImage;
  wbImage_t outputImage;
  float *hostInputImageData;
  float *hostOutputImageData;
  const char *inputImageFile;

  //@@ Insert more code here
  float   *deviceImageFloat;
  float   *deviceImagecmlt_dist_func;
  uint_t  *deviceImageHistogram;
  uint8_t *deviceImageUChar;
  uint8_t *deviceImageUCharGrayScale;

  args = wbArg_read(argc, argv); /* parse the input arguments */

  inputImageFile = wbArg_getInputFile(args, 0);

  wbTime_start(Generic, "Importing data and creating memory on host");
  inputImage = wbImport(inputImageFile);
  imageWidth = wbImage_getWidth(inputImage);
  imageHeight = wbImage_getHeight(inputImage);
  imageChannels = wbImage_getChannels(inputImage);

  hostInputImageData  = wbImage_getData(inputImage);

  outputImage = wbImage_new(imageWidth, imageHeight, imageChannels);

  hostOutputImageData = wbImage_getData(outputImage);

  wbTime_stop(Generic, "Importing data and creating memory on host");

  //print width, height, and channel of image
```

```
182   printf("%d, %d, %d\n", imageWidth, imageHeight, imageChannels);
183
184   /*
185   Cuda Toolkilt Documentation - Programming Guide @ B.29
186   Assertion stops the kernel execution if expression is equal to zero.
187   Triggers a breakpoint withing a debugger
188   and the debugger can also be stopped to inspect the device's current st
189   */
190   assert(imageChannels == chan_count);
191
192   //@@ @@//
193   //@@ Here I allocate GPU memory @@//
194   int imageArea = imageWidth * imageHeight;
195   int imageVol = imageWidth * imageHeight * imageChannels;
196   cudaMalloc((void**) &deviceImageFloat, imageVol * sizeof(float));
197     //image grayscale
198   cudaMalloc((void**) &deviceImageUChar, imageVol * sizeof(uint8_t));
199   cudaMalloc((void**) &deviceImageUCharGrayScale, imageArea * sizeof(uint
200     //the actual histogram
201   cudaMalloc((void**) &deviceImageHistogram, HISTOGRAM_LENGTH * sizeof(ui
202   cudaMemset((void**) &deviceImageHistogram, 0, HISTOGRAM_LENGTH * sizeof
203     //the Cumulative Distribution Function
204   cudaMalloc((void**) &deviceImagecmlt_dist_func, HISTOGRAM_LENGTH * size
205
206   //@@ Here I copy memory to the GPU @@//
207   //it is the memory input into the GPU
208   cudaMemcpy(deviceImageFloat, hostInputImageData, imageVol * sizeof(floa
209   //@@ Initialize the grid and block dimensions here:
210   dim3 dimensionBlock;
211   dim3 dimensionGrid;
212
213   //for uint8_t
214   dimensionBlock = dim3(TILE_WIDTH, TILE_WIDTH, 1);
215   dimensionGrid = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), ima
216   //perform float to uint8_t:
217   float_to_uint8_t<<<dimensionGrid, dimensionBlock>>>(deviceImageFloat, d
218   cudaDeviceSynchronize();
219
220   //convert to grayscale
221   dimensionBlock = dim3(TILE_WIDTH, TILE_WIDTH, 1);
222   dimensionGrid = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), 1);
223
224   color_to_dark<<<dimensionGrid, dimensionBlock>>>(deviceImageUChar, devi
225   cudaDeviceSynchronize();
226
```

```
227   //convert to histogram
228   dimensionBlock = dim3(32, 32, 1);
229   dimensionGrid  = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), 1)
230
231   dark_to_graph<<<dimensionGrid, dimensionBlock>>>(deviceImageUCharGraySc
232   cudaDeviceSynchronize();
233
234   //convert to cdf
235   dimensionBlock = dim3(HISTOGRAM_LENGTH, 1, 1);
236   dimensionGrid  = dim3(1, 1, 1);
237
238   scan_to_stat<<<dimensionGrid, dimensionBlock>>>(deviceImageHistogram, d
239   cudaDeviceSynchronize();
240
241   //equalization function
242   dimensionBlock = dim3(32, 32, 1);
243   dimensionGrid  = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), im
244
245   equal_func<<<dimensionGrid, dimensionBlock>>>(deviceImageUChar, deviceI
246   cudaDeviceSynchronize();
247
248   //convert to uint8
249   dimensionBlock = dim3(32, 32, 1);
250   dimensionGrid  = dim3(ceil(imageWidth/32.0), ceil(imageHeight/32.0), im
251
252   uint8_t_float<<<dimensionGrid, dimensionBlock>>>(deviceImageUChar, devi
253   cudaDeviceSynchronize();
254
255   //@@ insert code here
256   //CPU Operations follow
257
258   //@@ Here I copy the output memory to the CPU
259   cudaMemcpy(hostOutputImageData, deviceImageFloat, imageWidth * imageHei
260
261   //@@ Here I check the output image solution and free GPU memory
262   wbSolution(args, outputImage);
263   cudaFree(deviceImageFloat);
264   cudaFree(deviceImageUChar);
265   cudaFree(deviceImageUCharGrayScale);
266   cudaFree(deviceImageHistogram);
267   cudaFree(deviceImagecmlt_dist_func);
268   // Free CPU Memory
269   free(hostInputImageData);
270   free(hostOutputImageData);
271
```

```
272
273   wbTime_stop(GPU, "Freeing GPU Memory");
274
275
276   //@@ insert code here
277   //DONE
278
279   return 0;
280 }
281
```