MP6

(2 pts) QUESTION 1. For all the kernels, describe at least 2 optimizations you tried, regardless if they do or do not improve the overall performance. (Please do not answer optimizations that are asked in the later questions)

Solution:
Optimization 1 (1pt): Using (3 * Image_width * Image_height) number of threads to process the casting the Image into unsigned char instead of just (Image_width * Image_height) since the workload is more evenly distributed. (More parallelism)

Optimization 2 (1pt): Combine the step of applying histogram equalization and casting the unsigned char back to float to reduce the unnecessary function call time. (Kernel Fusion)

(This is an open-ended question, so any reasonable optimizations should be considered as correct.)

(1 pt) QUESTION 2. Which of your optimizations improve the performance most? Please explain it with Time reduced. (Timer information could be found at your attempt details page. Feel free to show the performance improvement if you have successfully implemented optimizations asked in the later questions).

Solution: optimization and corresponding performance result (1pt)
Students will need to include one optimization and comparison of Time for corresponding kernels copied from their Timer Output.

(2pts) QUESTION 3. For the private histogram kernel, how many global memory reads and writes are being performed by your kernel in terms of N (number of pixels), NUM_BINS, NUM_BLOCKS, BLOCK_SIZE. Do not include atomic operations for global read/writes. Explain your answer.

Solution:
Global reads: N (1pt, answer and explanation)
Global writes: 0 (1pt, answer and explanation)

All pixels are read only once. Also, there are no global writes because all write operations are atomic.

(1pt) QUESTION 4. For the histogram kernel, how many atomic operations are being performed by your kernel in terms of N (number of pixels), NUM_BINS, NUM_BLOCKS, and BLOCK_SIZE. (Considering atomic operations to both shared memory and global memory)

Solution:
NUM_BINS * NUM_BLOCKS + N (no explanation required)
There is an atomic_Add operation for every pixel, and each block writes NUM_BINS values; therefore, there are total NUM_BINS * NUM_BLOCKS + N atomic operations.

(2pts) QUESTION 5. Recall from the lecture that we use "stride=blockDim.x * gridDim.x" in our private histogram kernel. We let each thread collect multiple input pixel data that are "stride" away. Alternatively, we can also simply launch the kernel with ceil(1.0* numPixels/BLOCK_SIZE) thread blocks and let each thread collect exactly one input pixel data. For a large input picture, is the "stride" version better than the simple version? Please explain your answer for full credits.

Solution:
(1pt) Yes, answer any one of the following reasons
(1pt) 1. This is called thread coarsening. Large number of thread blocks will be serially scheduled and the overhead for indexing computation will be serialized as well. By coarsen multiple block executions, we can effectively reduce this overhead and block launch overhead.
2. For each thread block, we need to commit the private histogram back to the global histogram. Reducing the number of thread blocks can also redQuce the total number of global atomic operations.

(2pts) QUESTION 6. Given NUM_BINS=256, BLOCK_SIZE=1024, and HIST_NUM=4, we want to further improve our private histogram kernel with the following design. We declare multiple private histograms as:

```
__shared__ unsigned int privateHistogram[NUM_BINS][HIST_NUM];
```

When we are collecting pixel data, we perform the following indexing scheme:

```
atomicAdd(&privateHistogram[input[idx]][threadIdx.x%HIST_NUM], 1);
```

In the end of the kernel, we will collapse the HIST_NUM dimension (either by iteration or reduction) and then commit the private histogram to global histogram. Please list 2 benefits of this design over a single privateHistogram. Explain your answers to earn full credits.

Solution:
(1pt)1. Coalesced memory accesses with HIST_NUM in the second dimension and threadIdx.x%HIST_NUM access patterns. (1pt)

(1pt)2. Relaxing contention within a warp. This is a warp queue (or warp histogram in this context) implementation. It tries to avoid part of warp-wise atomic operations by using separate private histograms.