Lab3: Optimizing Histogram in CUDA

Xiaoyang Wang, Xueying Liao

a. What is the goal for the optimization, and briefly describe the changes you made in the source files for that optimization.

- The goal for the optimization is to reduce time consuming under the premise that the result is correct. Compared to time cost, we care less about space consuming.
- In the opt_2dhisto.cu file, each grid holds 1 block, and each block holds 256 threads.
- *HistCal()* calculates the histogram. *uint32to8()* transforms 32-bit bins to 8-bit bins to meet the requirement.
- We implement *atomicAdd()* in the *HistCal*, and use global memory to store the histogram.

b. Any difficulties with completing the optimization correctly.

- Diffficulties in General Idea:

  We choose the second optimization approach on the lecture slides and implement two additional optimization approaches, which is shared memory and multi-block.

  One approach for this lab is to implement 32 sub-histogram in shared memory for 32 threads/1 warp in a block. The shared memory is 48Kb in our GTX680 GPU. We utilize 32Kb shared memory to store 32 sub-histograms, and these 32 threads respectively use its own sub-histogram. The solution for the possible bank conflict is padding. But we soon realize the bank is organized by 4 bytes but our bins is in 1 byte so there'll always be potential bank conflicts. After completing calculation of the 32 sub-histograms, we then use sum reduction to merge them together and get the final histogram. The point of this optimization is to eliminate atomic operation, which will needs hundreds of cycles to assure read and write. However, after we finishing this optimization, the time consuming is much more than we expected, need about 25s for 1100 iteration(Time is wrong for 1000 iteration, don't know the reason). We guess the sum reduction need a lot of time. In this approach, it need 1024 iterations and 6 operations for each iteration. Implement multi-block processing will reduce the cost of the sum but it need atomic operation to generate the histogram in global memory. (Some bins in the output is incorrect but doesn't influence the overall performance). We abandon this approach finally.

  Another approach for this lab is assign tiles to blocks. The details is showed in part a of this report.

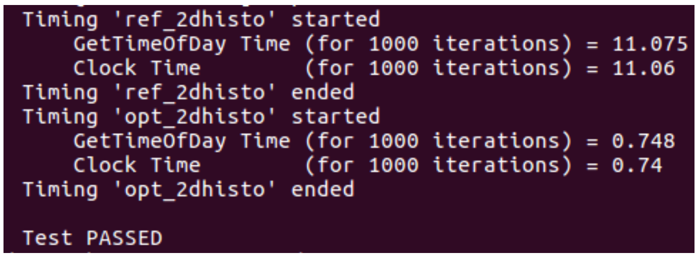
- Difficulties in Coding Detail:

  The most weird and confusing bug in our code is the type of input data. The data copied to GPU matches the original input data only for the first row. We print the value of the pointers of the end of first row and the start of the second row and find a gap between them. We guess the rows may stored in global memory separately, but soon we find it's due to the input padding. And the input padding also cause the mis-count for 0 value because the program also counts the padded part. The reason for padding may be coalescing access, 4 byte * 32 thread = 128 bytes. Keep the length of each row 128 will improve the performance.

c. The man-hours spent developing the optimization (even if it was abandoned or not working). This will be an indication of the optimization's difficulty.

We worked on this project for about 26 man-hours with two approaches.

d. If finished and working, the speedup of the code after the optimization was applied.



```
Timing 'ref_2dhisto' started
    GetTimeOfDay Time (for 1000 iterations) = 11.075
    Clock Time        (for 1000 iterations) = 11.06
Timing 'ref_2dhisto' ended
Timing 'opt_2dhisto' started
    GetTimeOfDay Time (for 1000 iterations) = 0.748
    Clock Time        (for 1000 iterations) = 0.74
Timing 'opt_2dhisto' ended

Test PASSED
```

The time using CUP for 1000 iterations is 11.075s, while the one on GPU with our optimization is 0.748s. The speed-up is approximately 14.8x.