MP6 Journal

Course: EE5351 Applied Parallel Programming
Submitted by: Shaman Narayana (naray146)

Optimisation methods tried

1)2D execution:

- Similar to the matrixmul operation, in the kernel, inputs were called in the 2 dimensional format. But, the speed up observed was only 6x.
- This was abandoned after realising that image will have spatial similarity between its pixels, so all the threads will try to update the same bin and because of contention, performance will be less. To exploit spatial similarity, 1D is beneficial.
- Around two hours was spent on this implementation (without taking into consideration the time spent on basic setup).

2)1D execution:

- The elements in memory will be stored in row-major format, so taking this as input directly (by removing the zero_padding), this was used in the kernel for computation.
- The speed up observed was 10x. This is with the histo_bins in shared memory.
- Challenge was in clearing the output memory. This was the first time resetting memory was necessary. This was done by making each thread clear up one memory space. This was the performance hit.
- Around three hours was spent on this.

3)atomicCAS:

- Initially, the understanding was that any input image element is processed, then the histogram_bin needs update.
- And another challenge was, declaring the histogram_bin as uint8_t data type (since 255 is the saturation limit). So, both of the above was taken care by calling a atomicCAS function (taken from NVIDIA website. No extra header file was needed).
- This made lot of the execution in series since all threads had to pass through atomicCAS function. The performance dropped below 1x.
- Around five hours was spent on this.

4)Going back to global memory

- Observation was that lot of time was taken up in clearing the shared memory. So, tried not to use the shared memory and a performance gain was observed!
- The cuda specific function to clear the global memory was used (cudaMemset from NVIDIA website).
- The performance now was 10x.
- Time spent was approx two hours.

5)conditional execution of atomicAdd

- Initially all the threads were updating the histo_bins irrespective of the value. Since the problem statement defines the saturation value, once the histo_bin reaches this value, no updates are necessary. So, a condition was placed.
- A performance gain of extra 10x was observed!
- This was due to avoiding many serial operations (atomicAdd).
- The performance gain for large inputs is 20x.
- Time spent was approx three hours.

6)second kernel to saturate the bins

- A second kernel was written to make sure all the histo_bins are saturated properly. Since this is executed in parallel, a further performance gain was observed.
- Time spent - half an hour
- Performance gain was 25x for large inputs

7)Calling the kernel in loop

- The optimum block_size was found to be 512 after trial and error and this put up a restriction on the number of grids that can be launched in a kernel at once.
- So, to cater to large input data elements, kernel is called in a loop. Even though no performance gain can be attributed to this implementation, large data sets can be handled now.

Statistics:

Input Image Size	CPU execution time (seconds)	GPU execution time (seconds)	Speed up in GPU
120*60	0.001	0.003	0.33
996*1024	0.11	0.01	11
31245*9999	32.99	1.05	31.42

Simulation results:

1) Size: 996*1024
```
Timing 'ref_2dhisto' started
GetTimeOfDay Time (for 50 iterations) = 0.109
Clock Time      (for 50 iterations) = 0.11
Timing 'ref_2dhisto' ended
Timing 'opt_2dhisto' started
GetTimeOfDay Time (for 50 iterations) = 0.008
Clock Time      (for 50 iterations) = 0.01
Timing 'opt_2dhisto' ended

Test PASSED
```

2) 120*60
```
Timing 'ref_2dhisto' started
GetTimeOfDay Time (for 50 iterations) = 0.001
Clock Time      (for 50 iterations) = 0
Timing 'ref_2dhisto' ended
Timing 'opt_2dhisto' started
GetTimeOfDay Time (for 50 iterations) = 0.002
Clock Time      (for 50 iterations) = 0
Timing 'opt_2dhisto' ended

Test PASSED
```

3) 31245*9999
```
Timing 'ref_2dhisto' started
GetTimeOfDay Time (for 50 iterations) = 32.979
Clock Time      (for 50 iterations) = 32.99
Timing 'ref_2dhisto' ended
Timing 'opt_2dhisto' started
GetTimeOfDay Time (for 50 iterations) = 1.045
Clock Time      (for 50 iterations) = 1.05
Timing 'opt_2dhisto' ended

Test PASSED
```