Name: Rohan Jhaveri

Assignment 4 : GPU (CUDA) Histogram and Atomics

[Item 1] – .cu codes submitted and results for Histogram without atomics is shown below

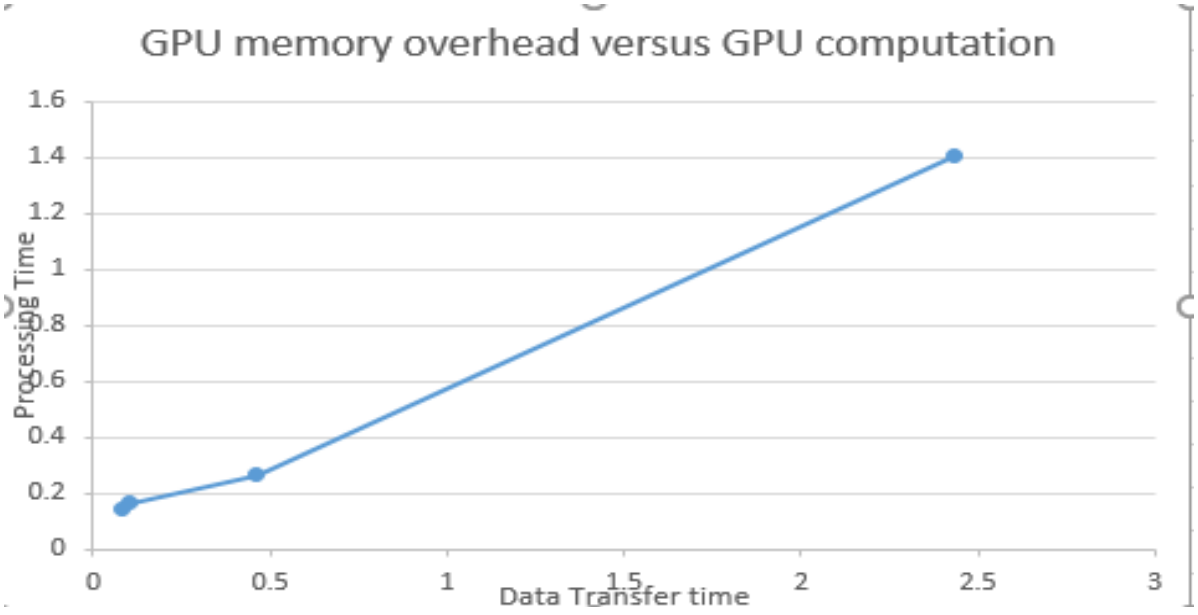
Fill in the execution times using the event timers (T=32, H=64)

Array Size (N)	CPU Execution	GPU Execution + CPU clean up: **histogram_kernel**	GPU Speedup
1000	0.001000(ms)	0.246000 (ms)	0.04x
10000	0.0110	0.340	0.02x
100000	0.111000	0.727000	.001x
1000000	1.192000	3.836000	0.3x

Implementation details – This implementation makes use of one kernel to generate histogram and a function to merge the partial outputs of the individual blocks in GPU. The number of blocks and threads are decided by the user. Based on that, the data partition per block is decided.
The issue with this implementation is that even though the processing time is not too much as the data size increases, the memory transfer time increases and causes the benefit to drop. In other words, no speed up is achieved.
The figures above do not hold a constant block size, only constant thread size for different array sizes. The point of this implementation is, that processing time can be reduced by increasing the number of blocks, but memory transfer time will always be high. The same can be seen in the figure and table below.

Graph: GPU memory overhead versus GPU computation

Array Size (N)	Memory Time(ms)	Processing Time(ms)
1000	0.082000	0.143000
10000	0.104	0.163
100000	0.463000	0.264000
1000000	2.431000	1.405000



Name: Rohan Jhaveri

Assignment 4 : GPU (CUDA) Histogram and Atomics

Item[2]: Performance Evaluation

(a)Block Execution time.

4 blocks and 32 threads. All times in (ms)

Array Size (N)	Block 1	Block 2	Block 3	Block 4
1000	0.094818	0.095108	0.095216	0.094894
10000	0.1386	0.138056	0.138038	0.138640
100000	0.56672	0.56859	0.572182	0.57218
1000000	6.642112	6.651236	6.688734	6.698324

The longest running time for the block is for block 2 in the total of 4 blocks for 1000 N size. The longest took 0.095216 msec and shortest took 0.094818 msec.

Calculating by the block execution times seen above, there will be a delay or stall of 0.0001 ms for the last block to finish while all others blocks are done. When the data size increases, the stall becomes 0.01 ms (in last row).

[Item 3] – histogram_atomic_kernel: You are to examine an optimization through the use of an atomic instruction to enforce one thread at a time accessing to individual locations in the histogram array.

Array Size (N)	GPU Execution + CPU clean up: **histogram_kernel**	GPU Execution + CPU clean up: **histogram_atomic_kernel**	Speedup
1000	0.246000 (ms)	0.12	2.05x
10000	0.340	0.1667	2.03x
100000	0.814	0.725	1.125x
1000000	6.148	5.892	1.03x

[Item 4] – Graph: Overhead of atomic clock

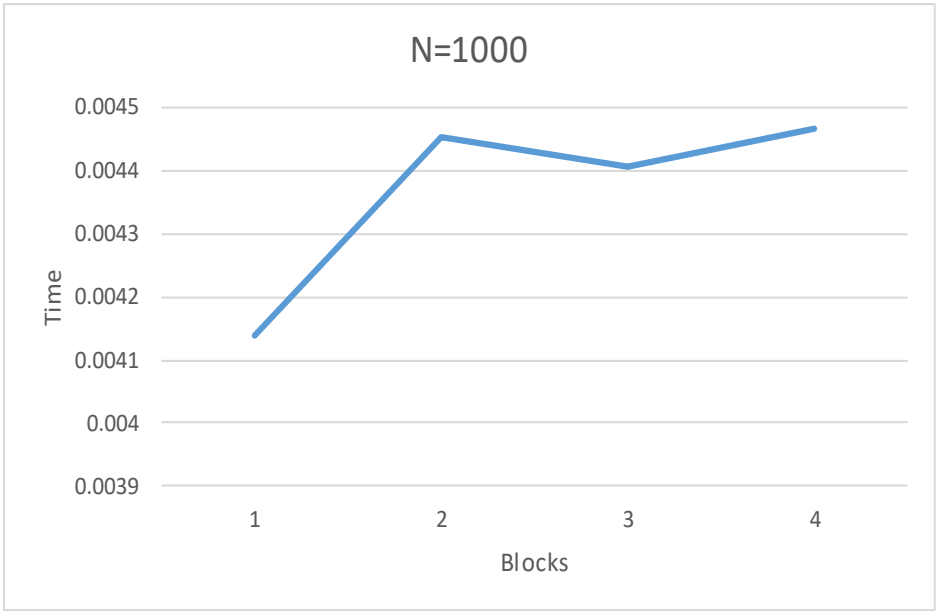
The 2nd image below the images show a table of GPU memory transfer time overhead vs the gpu processing time. It is even more steep showing how the memory transfer hurts the processing advantage even further.

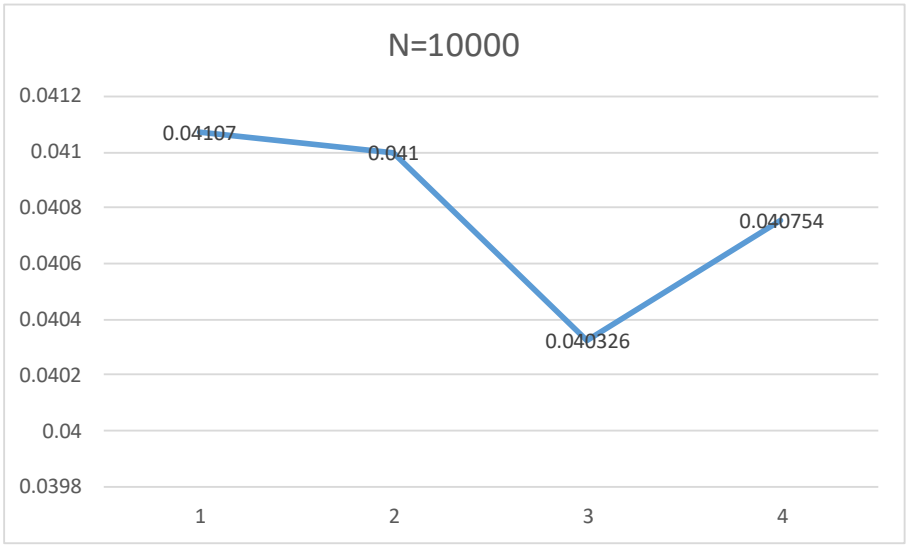
The table and the graphs below represent how much time each block implementing the atomic instruction takes. It can be seen that all of them take fairly the same amount of time.
Overhead of atomics in each block, for 4 blocks and 32 threads each in (ms).

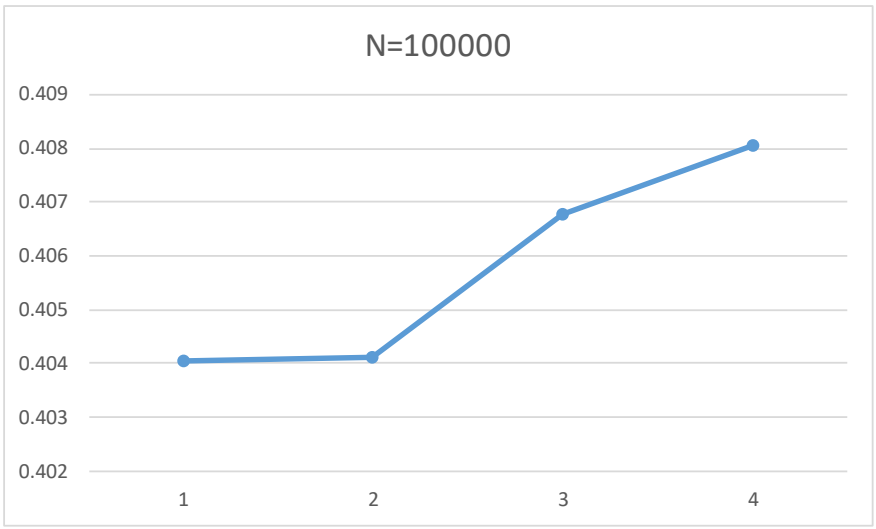
Array Size (N)	Block 1	Block 2	Block 3	Block 4
1000	0.00414	0.004452	0.004406	0.004468
10000	0.04107	0.04100	0.040326	0.040754
100000	0.404028	0.404124	0.406776	0.408070
1000000	5.80264	5.812834	5.852628	5.854978

Name: Rohan Jhaveri

Assignment 4 : GPU (CUDA) Histogram and Atomics







Name: Rohan Jhaveri

Assignment 4 : GPU (CUDA) Histogram and Atomics