Histogram Generation Report

Name: Alan Manuel Loreto Cornídez
Professor: Dr. Ali Akoglu
Class: High Performance Computing | ECE 569
Assignment: Homework 4
Due Date: April 4th, 2023

Table Of Contents

Table Of Contents **1**
Results **2**
Analysis-1: **3**
V0 vs V1 on Experiment1, how much and privatization 3
V0 vs V1 on Experiment 2, how much with privatization 3
Experiment1 vs Experiment2 on V0, impact of input data on V0 3
Experiment1 vs Experiment2 on V1, impact of input data on V1 3
Analysis-2 **4**
Short Answer **5**
How many global memory reads are being performed by each kernel? Explain. 5
How many global memory writes are being performed by each kernel? Explain. 5
How many atomic operations are being performed by each kernel? Explain. 5

Results

Experiment 1			
Random Data Generation			
Test	**Method 0**	**Method 1**	**Method 2**
1	0.105	0.0822	82.5864
2	0.1007	0.0966	73.0431
3	0.1006	0.093	72.3964
4	0.097	0.0923	73.1855
5	0.0999	0.0904	73.3573
6	0.0998	0.0915	80.0096
7	0.1005	0.0897	72.0405
8	0.1439	0.0935	72.4136
9	0.1029	0.0911	72.6778
10	0.1032	0.0907	78.1255
Average (ms)	**0.1054**	**0.0911**	**74.9836**

Experiment 2			
All in a single bin			
Test	**Method 0**	**Method 1**	**Method 2**
1	0.3969	0.6294	86.333
2	0.3962	0.1016	77.9207
3	0.4036	0.0986	77.8515
4	0.4003	0.0963	77.2503
5	0.3976	0.096	76.6543
6	0.3966	0.0928	74.3155
7	0.4027	0.0938	74.8778
8	0.4005	0.1015	74.2637
9	0.4063	0.097	74.6303
10	0.398	0.0982	74.6946
Average (ms)	**0.3999**	**0.1505**	**76.8792**

Analysis-1:

V0 vs V1 on Experiment1, how much and privatization

Version 0 and version 1 both achieved a similar execution time on experiment 1 in comparison to each other. The execution times where not more than 100 μS apart from each other. This means that shared memory did not provide much of an execution time reduction whenever the data was random.

V0 vs V1 on Experiment 2, how much with privatization

In contrast, experiment 2 highlights how shared memory can have an effect on reducing global memory accesses and thus, increasing the arithmetic operations that are conducted. In experiment we can see that the execution time between version 0 and version 1 is less than 50% (0.399 ms vs 0.15 ms). Since the same data point is being accessed, shared memory allows faster execution of atomic add operations whenever atomic collisions are experienced.

Experiment1 vs Experiment2 on V0, impact of input data on V0

Experiment 2 shows how the nature of the input data affects execution time of the global memory implementation. When the data is random, there are less atomic collisions occurring, and thus, faster execution when compared to data where one bin must be incremented. More atomic collisions occur in experiment 2 and global memory accesses show the memory bandwidth at its limit. There was an increase in execution time of about 0.30 ms when the nature of input data was changed.

Experiment1 vs Experiment2 on V1, impact of input data on V1

The nature of the data also affects the shared memory implementation. However, as we can see, there is not as much of an impact to execution time when compared to the impact of execution time on the global memory implementation. There was about a 0.04 ms increase in execution time. (0.15 ms vs 0.09 ms). This shows how shared memory implementations can benefit from applications where data is not normally distributed.

Analysis-2

Explain your implementation approach for Version-2. Discuss the superiority of your approach (Version 2) compared to versions 0 and 1. What is the parallelization opportunity that you exploited or what drawbacks of Versions 0 and 1 did you resolve? Does performance vary based on the nature of the distribution of the data? For which test cases does Version-2 perform better than Versions 0 and 1? Discuss any relevant execution time trends. (1.5 page limit)

My attempt at optimizing the implementation was unsuccessful. My attempt involved creating a local histogram and assigning one thread to a certain value in the local histogram. I would then accumulate each thread's histogram in the shared memory and then copy the shared memory histogram to the global memory. As you can see in my results, I did not actually optimize the implementation. In fact, it was much slower than the other two implementations.

Some of the issues that arose were that I was only able to get the correct amount up to the amount of threads launched. That is, if there were 1024 threads launched in the kernel, then I would only have the first 1024 bins of the histogram correct.

In theory, this method would be superior because each thread would be responsible for a bin range in the histogram and thus, atomic add operations collisions would be less common, in turn increasing the parallelization of the histogram.

Execution time for my implementation had worse performance than both of the other kernel implementations. I had code in there that was not optimized and caused a big increase in execution time.

Short Answer

How many global memory reads are being performed by each kernel? Explain.

Global Kernel

The number of global memory reads is equal to the number of elements in the input. This is because each thread reads each element from global memory.

Shared Memory

The number of global memory reads is also equal to the number of elements in the input. The main difference between the global and shared memory kernel is that they each write to a different histogram.

Optimized Kernel

TODO

How many global memory writes are being performed by each kernel? Explain.

Global Kernel

The number of global memory reads is equal to the number of elements in the input. This is because each atomic operation occurs in the global memory.

Shared Memory

The number of global memory reads is also equal to the number of bins in the histogram. This is because the shared memory is where most memory writes occur when accumulating the histogram. The global memory is written to after the shared memory histograms have been created. NOTE: If there are multiple shared memory blocks, then the number of global memory writes is equal to the number of bins in the histogram times the number of shared memory blocks.

Optimized Kernel

The number of global memory reads is also equal to the number of elements in the input.

How many atomic operations are being performed by each kernel? Explain.

Global Kernel

The number of atomic operations is equal to the number of elements in the input. This is because each atomic operation occurs in the global memory for every element that is read from the input

Shared Memory

The number of atomic operations in the shared memory implementation is equal to the number of elements plus the number of bins in the histogram. This is because an atomic operation occurs every time the shared memory is updated and when the global histogram is updated.

Optimized Kernel

The number of global memory reads is also equal to the number of elements in the input.