GPU Project 2: A Better Version of SDH Computing Program

Brian Pinson
U91813366
CIS4930.004
06/28/2018

I. INTRODUCTION

This project seeks to measure the performance of the GPU memory system by measuring different implementations and experimenting to create a program that is as fast and efficient as possible. The program that will be used is designed to compute the spatial distance histogram of a set of points first using the CPU and a naive algorithm and then using a GPU and an algorithm that can take advantage of parallel processing.

The implementation used in this project is based on the algorithms found in the paper: "Efficient 2-Body Statistics Computation on GPUs: Parallelization & Beyond" written by Napath Pitaksirianan, Zhila Nouri, and Yi-Cheng Tu [1]. Constant memory was also used when applicable to help minimize memory bandwidth.

II. METHOD

The first step was to take Project 1 and change it to allow the user to input the size of the block. This would serve as the baseline to compare future implementations to. Project 1 arranges the threads into 2-dimensional arrays where each thread calculated the distance between two points where the x thread id represented one point and the y thread id represented the other. Project 1 could not be reused in Project 2 because all block sizes had to be the square of some integer and the number of threads needed for an input of size n was n^2 which meant any input slightly over 500,000 would cause the program to crash.

Project 2 had to be completely rewritten using a new algorithm. The implementation comes from the paper mentioned above and was completed in three stages. First the naive algorithm was implemented where each thread id represented one point and this point was compared with every other point. The greatest bottleneck with the naive algorithm was using atomicAdd to update the output. Each thread was competing with each other to update the output histogram and each update only incremented the histogram by one.

The second stage of the implementation was to take advantage of shared memory and lessen the times global memory had to be accessed. Before shared memory was used the algorithm had to be rewritten to be block-based. The "Block-based 2-BS computation" [1] algorithm finds the distance between a point represented by its thread id and every point in all other blocks greater than its own, then all other points in its own block if those points have a thread id greater than its own. The reason only threads greater than the thread id are used in the computation is because if the distance between points x and y was already found then finding the distance between points y and x would be superfluous and make the histogram erroneous.

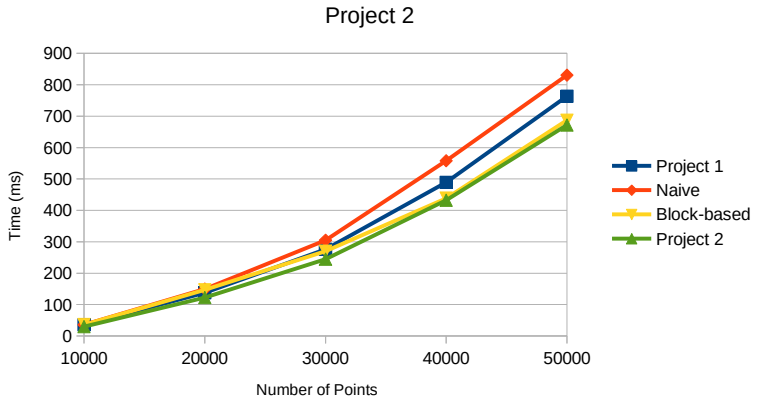
The final stage was to use shared memory, the registers, and constant memory to avoid accessing global memory and to take advantage of the faster speeds these memory types had. Shared memory had to be dynamically allocated since the block size would be decided by the user. Enough space had to be reserved to fit two blocks worth of atoms and an array that holds the histogram output. Each block had a copy of its atoms stored in its shared memory in an array called L. Before each block would compute the distances between its points and points in other blocks those atoms in the other blocks would be stored in shared memory in an array called R. Using __syncthreads() no thread would move on to another block before all other threads were finished so that all threads could use the shared memory. To speed things up each thread stored its own atom in the register so that it could access its own coordinates even faster. Then to make the algorithm even faster certain unchanging variables such as the number of data points were loaded into constant memory so that blocks could access information quickly in a warp read.

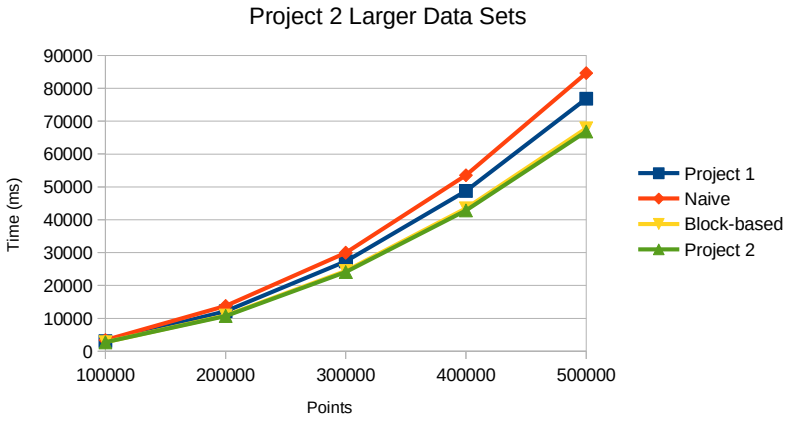
III. RESULTS

Experiments were run to test the accuracy of the GPU implementation as well as how quickly they were able to solve the problems. Since this project was done in three steps the improvements from project one and all three implementations were recorded. As the data shows each new implementation showed an increase in speed.

Points	Project 1 (ms)	Naive Implementation (ms)	Block-based Implementation (ms)	Project 2 (ms)
10000	35.49	35.39	36.08	29.89
20000	137.53	149.61	146.83	122.14
30000	275.93	305.14	269.73	244.69
40000	488.92	557.85	438.61	432.53
50000	763.33	830.53	687.11	671.41
100000	3042.26	3369.67	2735.48	2700.22
200000	12154.07	13766.55	10843.39	10733.81
300000	27331.32	29958.91	24425.02	24113.13
400000	48736.81	53549.46	43452.97	42843.49
500000	76804.67	84629.04	67847.91	66849.29

The results show that the Project 2 implementation is the fastest of all the implemented algorithms when using small and large data sets. Project 1 in its 2D approach was actually faster than a 1D naive approach to solving the problem. This might be because of how much more simpler the algorithm was. The Block-based implementation was very close to the shared memory implementation which was surprising but this is most likely due to many users competing to use the C4 lab machines. The results have varied depending on the hour the computers are being used but they are at least consistent in showing that each implementation is faster than the old one.





IV. Conclusion

This project clearly demonstrates how much slower global memory is and how it can bottleneck the entire program. Shared memory, constant memory, and registers are significantly faster however their limited size can also cause problems of their own when the user specifies a large block. It will be interesting to see how GPU's innovate and further the field of parallel processing over the next few years.

References

[1] Napath Pitaksirianan, Zhila Nouri, and Yi-Cheng Tu. " Efficient 2-Body Statistics Computation on GPUs: Parallelization & Beyond". Proceedings of 45th International Conference on ParallelProcessing, pp. 380-385., August 2016.