Project 1: Spatial Distance Histogram Computation

Instructor: Dr. Yi-Cheng Tu
Course: CIS6930/4930 Computing on Massively Parallel Systems
Submission Deadline: Wednesday, September 25, 2019 at 11:30 PM

1 Problem Statement

This project is about computing spatial distance histogram (SDH) of a collection of 3D points. The SDH problem can be formally described as follows: given the coordinates of N particles (e.g., atoms, stars, moving objects in different applications) and a user-defined distance w, we need to compute the number of particle-to-particle distances falling into a series of ranges (named buckets) of width w: $[0, w), [w, 2w), \ldots, [(l-1)w, lw]$. Essentially, the SDH provides an ordered list of non-negative integers $H = (h_0, h_1, \ldots, h_{l-1})$, where each $h_i (0 \leq i < l)$ is the number of distances falling into the bucket $[iw, (i+1)w)$. Clearly, the bucket width w is a key parameter of an SDH to be computed.

2 Getting Started

For you to get started, we have written a C program to compute SDH. The attached file `SDH.cu` shows a sample program for CPUs and serves as the starting point of your coding in this project. Interesting thing is that you can still compile and run it under the CUDA environment. Specifically, you can type the following command to compile it.

```
nvcc SDH.cu -o SDH
```

To run the code, you add the following line to your testing script file:

```
./SDH 10000 500.0
```

Note that the executable takes two arguments, the first one is the total number of data points and the second one is the bucket width (w) in the histogram to be computed. We strongly suggest you compile and run this piece of code before you start coding.

3 Tasks to Perform

You have to write a CUDA program to implement the same functionality as shown in the CPU program. Both CUDA kernel function and CPU function results should be displayed as output.

Write a CUDA kernel that computes the distance counts in all buckets of the SDH, by making each thread work on one input data point. After all the points are processed, the resulting SDH should be copied back to a host data structure.

1. Transfer the input data array (i.e., atom_list as shown in the sample code) onto GPU device using CUDA functions.

2. Write a kernel function to compute the distance between one point to all other points and update the histogram accordingly. Note that between any two points there should be only one distance counted.

3. Copy the final histogram from the GPU back to the host side and output them in the same format as we did for the CPU results.

4. Compare this histogram with the one you computed using CPU bucket by bucket. Output any differences between them - this should be done by printing out a histogram with the same number of buckets as the original ones except each bucket contains the difference in counts of the corresponding buckets.

Note that the output of your program should only contain the following: (1) the two histograms, one computed by CPU only and the other computed by your GPU kernel. (2) any difference you found between these two histograms.

Hint: To correctly compute the histogram, you may need **atomic functions** to atomically modify a variable, such as adding the per-thread counts to the global histogram. These atomic functions can be called inside a CUDA kernel function. For example:

```
int a = atomicAdd(ptr, b);
```

In the code above, `atomicAdd` is a function that atomically adds value `b` to the variable represented by pointer `ptr` and returns the old value of `*ptr` to `a`. This function works with pointers in global memory and shared memory. For more details please refer to `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions`

4 Environment

All projects will be tested on CUDA 7.5 on one machine of the C4 lab. If you prefer to work on your own computer, make sure your project can be executed on the GPU card of a C4 lab computer.

5 Instructions to Submit Project

You should submit one .cu file with your implementation. For this project, we suggest you just add code to the file named SDH.cu we provided. Rename the file as `proj1-xxx.cu`, where `xxx` is your USF NetID. Submit the file only via assignment link in Canvas. E-mail or any other form of submission will not be graded. Once you submit your file to Canvas, try to download the submitted file and open it in your machine to make sure no data transmission problems occurred. For that, we suggest you finish the project and submit the file at lease one hour before the deadline.

6 Rules for Grading

Following are some of the rules that you should keep in mind while submitting the project. The project will be graded by a set of test cases we run against your code.

- All programs that fail to compile will get zero points. We expect your submission be compiled by running the simple line of command shown above.
- If, after the submission deadline, any changes are to be made to make the main code work, they should be done within 3 lines of code. This will incur a 30% penalty.
- Program should run for different numbers of CUDA blocks and threads. Use of any tricks to make it run on one thread or only on CPU will result in zero points. However, performance of your GPU code is not considered in grading.