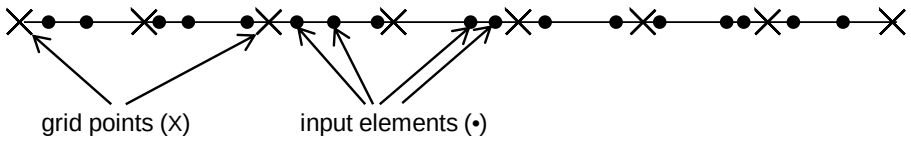
Lab 4: Input Binning

EE5355 - Parallel Algorithmic Techniques

Due: See Moodle

1 Objective

The purpose of this lab is to understand input binning and its impact on performance in the context of a simple computation example. The example is composed of a 1-dimensional grid of discrete points and an array of elements each having a non-discrete value and a non-discrete position in the grid, as shown in the following figure.



In this example, we would like to compute the total impact of all the input elements on each of the output elements. The total impact on a single output grid point is computed as follows:

$$out[j] = \sum_{i=0}^{N-1} \frac{val[i]^2}{(pos[i-j])^2}$$

2 Procedure

Step 1: Transfer the tarfile lab4.tgz into your project directory and extract it to create the lab4 directory.

Step 2: In the file `kernel.cu`, you will find a function named `cpu_normal`. This function implements a simple CPU version of the computation.

Compile and test the code:

```
make
./binning <m>            # Mode: m, Grid: 20,000, Input: 60,000
./binning <m> <M>        # Mode: m, Grid:      M, Input:    3*M
./binning <m> <M> <N>    # Mode: m, Grid:      M, Input:      N
```

For the mode option, select mode 1. There are 5 modes available for this lab:

- Mode 1 executes the CPU version
- Mode 2 executes the GPU version without any cutoff or input binning
- Mode 3 executes the GPU version with cutoff but no input binning
- Mode 4 executes the GPU version with cutoff and input binning where the input binning is performed on the CPU
- Mode 5 executes the GPU version with cutoff and input binning where the input binning is performed on the GPU

Mode 1 is provided to you, and you will be implementing modes 2-5 throughout the lab.

Step 3: Edit the kernel `gpu_normal_kernel` in the file `kernel.cu` to implement the same computation as `cpu_normal` on the GPU. Note however that `cpu_normal` performs the computation using a scatter pattern. For the kernel `gpu_normal_kernel`, you are required to use a gather pattern which is more efficient. Compile and test the code using mode 2.

Step 4: Edit the kernel `gpu_cutoff_kernel` in the file `kernel.cu` to implement the computation on the GPU, this time only considering input values that fall within a cutoff range of the output grid point. Compile and test the code using mode 3.

Step 5: Edit the kernel `gpu_cutoff_binned_kernel` in the file `kernel.cu` to implement the computation on the GPU. In this version, the input has been sorted into bins for you. You must loop over the bins and for each bin check if either of its bounds is within the cutoff range. If yes, you must loop over the input elements in the bin, check if each element is within the cutoff range, and if so, include it in your computation. Compile and test the code using mode 4.

Step 6: In the previous step, the input binning was performed for you in the CPU. In this step, you must perform the input binning on the GPU. Edit the kernels `histogram`, `scan`, and `sort` to perform the appropriate operations for input binning. To simplify your life, we have fixed the number of bins to 1,024 so that the scan can be performed in a single thread block.

Step 7: Edit the file `report.txt` to answer the following questions:

1. Try running all four implementations using varying input combinations. Use at least 5 different combinations. Try scaling the number of input elements and the grid size differently having both low, both high, one low and one high, etc. Comment on the performance impact of each mode for the various input sizes. Explain why you think certain implementations perform better than others for various input combinations. In modes 4 and 5, make sure to also include the preprocessing time in your comparison, not just the kernel launch time.

Step 8: Submit your assignment. You should only submit the following files:

- `kernel.cu`
- `report.txt`

3 Grading

Your submission will be graded based on the following criteria.

- Functionality/knowledge: 75%
  - Correct code and output results for mode 2
  - Correct code and output results for mode 3
  - Correct code and output results for mode 4
  - Correct code and output results for mode 5
  - Performance
- Answers to question: 25%
  - Thoughtfulness of input combinations used
  - Thoroughness of performance comparison and explanations demonstrating a clear understanding of the factors influencing performance
  - Neatness and clarity