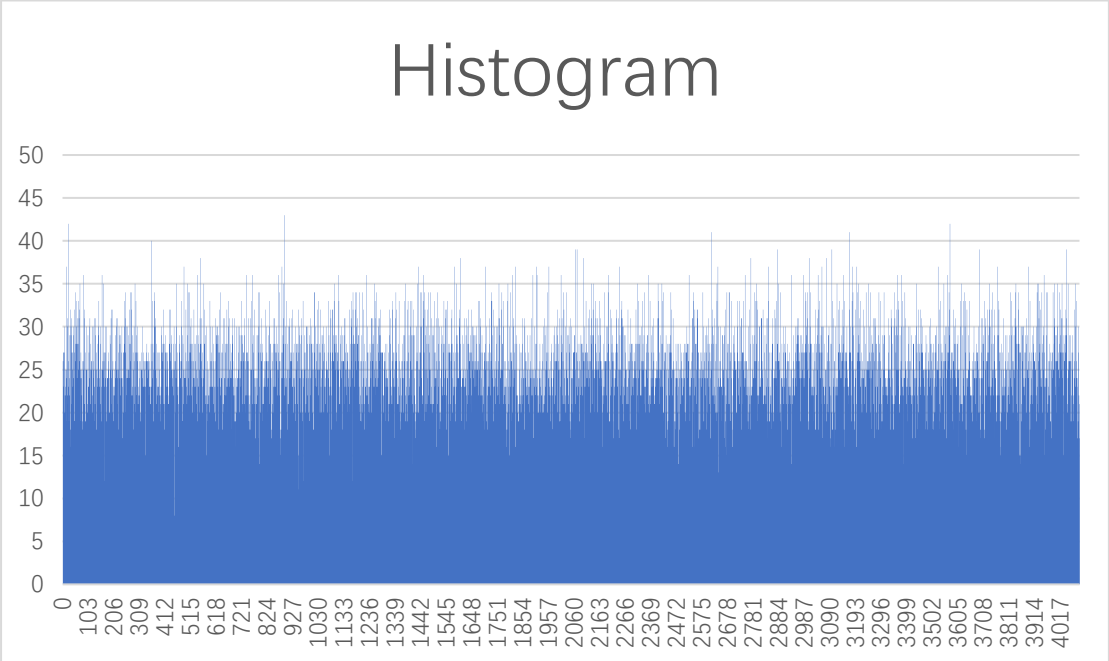
DD2360 Assignment 3
Group 35
Shiyuan Shan
Github repository: ShiyuanShan/DD2360HT23 (github.com)
Exercise 1:
1.1

1) Atomic operations are used to update histogram bins in shared memory to avoid race conditions when multiple threads write to the same bin simultaneously.
2) The loops that initialize shared memory bins in histogram_kernel and clean up bins in convert_kernel are unrolled. Loop unrolling can potentially improve instruction throughput by reducing loop overhead.
3) All threadDim.x can be replaced by the declared constant.

1.2 1) and 2) are the two major optimizations taken. 3) is not taken because it turns the code harder to read.
1.3 In total input_length global memory reads. Each element is visited only once.
1.4 In total input_length + num_bins atomic adds. Each element is categorized only once, and each shared bin is copied to the global bin once in the end.
1.5 In total num_bins* sizeof(unsigned int) shared memory. Bin size should be equal to the global bin size.
1.6 If every element in the input array has the same value, each thread within a block will attempt to update the same bin in the shared_bins array. This leads to contention, as multiple threads are competing to atomically increment the count in the same bin, resulting in low computation parallelism.
1.7 Input length is 100000, 256 threads per block and in total 391 blocks.



1.8 Achieved occupancy is 24.67%, and shared memory configuration size is 32.77 kb.

Possible problems are provided by ncu: This kernel's theoretical occupancy is

not impacted by any block limit. The difference between calculated theoretical (100.0%) and measured achieved occupancy (24.67%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel.

1.9

Exercise 2:

2.1

I use google colab for this exercise. Delete all “arch” related code in makefile.
Lines used for simulation are:

```
%cd sputniPIC/
!mkdir data
!make
!./bin/sputniPIC.out ./inputfiles/GEM_2D.inp
```

2.2 In mover_PC_gpu() is a parallelized version of mover_PC() by having part->nop threads, which means one kernel for one particle each time. The original for (int i = 0; i < part->nop; i++) loop gets unrolled.

2.3 Results are the same.

2.4

CPU:

```
****************************************
   Tot. Simulation Time (s) = 81.7802
   Mover Time / Cycle   (s) = 3.4422
   Interp. Time / Cycle (s) = 4.44722
****************************************
```

GPU:

```
****************************************
   Tot. Simulation Time (s) = 50.7898
   Mover Time / Cycle   (s) = 0.228208
   Interp. Time / Cycle (s) = 4.55658
****************************************
```