Assignment III: CUDA Basics II

Wang, Chenyi
chenyiw@kth.se

Ahmed, Ouday
ouday@kth.se

December 10, 2023

0 Git repository and Contributions

Git repository: https://github.com/OudayAhmed/DD2360HT23
Both group members completed the assignment task, and we arranged a meeting to discuss our solutions and write the report together.

1 Exercise 1 - Histogram and Atomics

1. (a) Using shared memory in the histogram kernel: Improved kernel launch time and increased memory throughput.

2. We used the shared memory for optimization because this optimization allows faster memory reads compared with using only global memory.

3. There are `num_elements` global memory reads in the histogram kernel and `NUM_BINS` global memory reads in the convert kernel. For the histogram kernel, there is a total of `num_elements memory` read from the input array. For the convert kernel, there is a total of `NUM_BINS` memory read from the bins array.

4. There are `num_elements + num_blocks * NUM_BINS` atomic operations in the histogram kernel. The total number of atomic operations in the `histogram_kernel` is the sum of

   (a) atomic operations for updating the shared memory bins for each input element (num_elements)

   (b) the atomic operations for updating the global memory bins for each bin (num_blocks * NUM_BINS)

5. The shared memory size is `num_Blocks * NUM_BINS` $\cdot$ 4 (byte)s. The shared memory is declared as an array of `unsigned int` within each block with a size of `NUM_BINS`.

6. If every element in the input array has the same value, this leads to a very high contention scenario. Since all threads in a block are updating the same bin, they will all be contending for the same memory location. This contention is resolved by atomic operations (atomic).

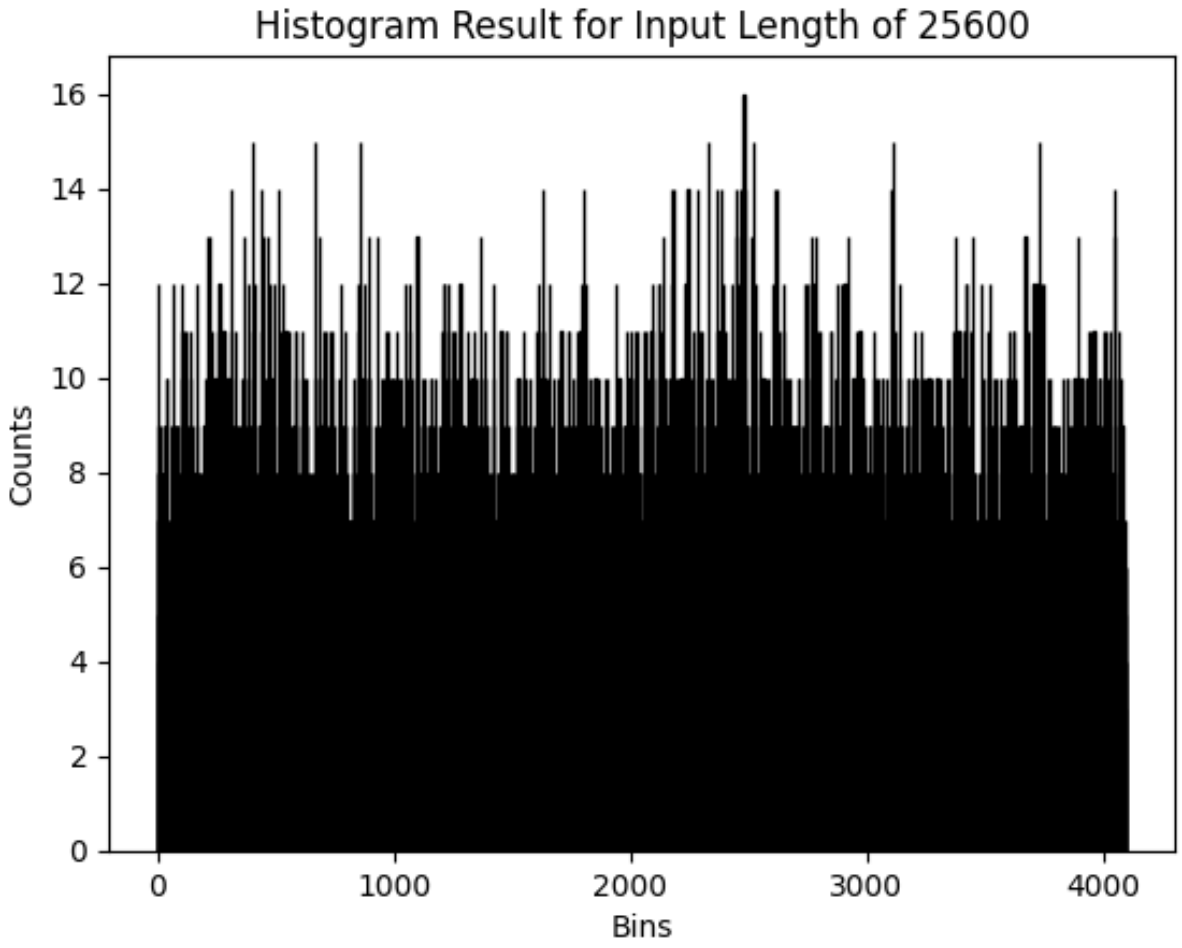

Figure 1: histogram results, input length is 25600, thread per block is (32, 1, 1), grid dimension is (800, 1, 1).

7. See Figure1
8. See profile result in Figure 2 and Figure3
   (a) The **Shared Memory Configuration Size** is 32.77 (KByte)
   (b) The **Achieved Occupancy** for the histogram kernel is 17.99 %
   (c) The **Achieved Occupancy** for the convert kernel is 92.50 %
   (d) Nvsight suggests that there might be warp scheduling overheads or workload imbalances during the kernel execution. In addition, it also informs that the required amount of shared memory limits this kernel's theoretical occupancy.

Block Size		1,024
Function Cache Configuration		cudaFuncCachePreferNone
Grid Size		1
Registers Per Thread	register/thread	36
Shared Memory Configuration Size	Kbyte	32.77
Driver Shared Memory Per Block	byte/block	0
Dynamic Shared Memory Per Block	byte/block	0
Static Shared Memory Per Block	Kbyte/block	16.38
Threads	thread	1,024
Waves Per SM		0.03

Figure 2: Shared Memory Configuration Size for histogram kernel

Section: Occupancy

Block Limit SM	block	16
Block Limit Registers	block	1
Block Limit Shared Mem	block	2
Block Limit Warps	block	1
Theoretical Active Warps per SM	warp	32
Theoretical Occupancy	%	100
Achieved Occupancy	%	17.99
Achieved Active Warps Per SM	warp	5.76

Figure 3: Achieved Occupancy for histogram kernel

```
!awk '{ getline f2 < "./data/test.vtk"; printf "%.10f\n", $1 - f2 }' ./data/rho_net_10.vtk
```

Figure 4: The command used to compare the output of both CPU and GPU implementation

2 Exercise 2 - A Particle Simulation Application

1. Google Collab. The sm-30 nvcc flag in the MakeFile was changed to sm-75 to be able to compile. To run the code, use the command "make" and "./bin/sputniPIC.out input files/GEM_2D.inp"

2. We have found that the original algorithm runs for certain `sub_cycles`, and within each cycle, all particles are updated via a for loop, which takes quite a lot of time on the CPU. We then extracted this particle loop part into a CUDA kernel function. In order to achieve this, we have also to allocate GPU memory for the parameters that we will need inside the kernel. To get the correct value in the flattened 1D array with a 3D index, we used the `get_idx` function defined in Alloc.h.

3. We use this command (See in Figure4) to compare the output of both CPU and GPU implementation. We found the results are the same.

4. See Table1

Time(s) of	CPU Mover	GPU Mover
Total Simulation	82.901	51.2227
Mover Time / Cycle	3.49125	0.265203
Interp. Time / Cycle	4.50369	4.56622

Table 1: execution time comparison