

# DD2360

## Assignment 3 - Report

Heng Kang  
hengkang@kth.se

Jian Wen  
jianwe@kth.se

Git repo: <https://github.com/SinGuLaRiT2001/DD2360HT23>

### Ex1. Histogram and Atomics

Running the program:

```
[81] !nvcc -arch=sm_75 ./HistAtom.cu -o HistAtom

!./HistAtom 400000

The input length is 400000
Reference result:
110 101 95 106 73 105 90 86 96 89 110 84 107 105 87 103 87 104 73 109 107 71 90 104 73 95
Kernel runtime: 0.000227 ms
GPU result:
110 101 95 106 73 105 90 86 96 89 110 84 107 105 87 103 87 104 73 109 107 71 90 104 73 95
Calculation match!
```

1. Describe all optimizations you tried regardless of whether you committed to them or abandoned them and whether they improved or hurt performance.

**ANSWER:** The codes for my histogram kernel are shown below:

```
__global__ void histogram_kernel(unsigned int *input, unsigned int
*bins,
                                unsigned int num_elements,
                                unsigned int num_bins) {
//@@ Insert code below to compute histogram of input using shared
memory and atomics
    __shared__ unsigned int hist_temp[NUM_BINS];
    int index = threadIdx.x + blockIdx.x * blockDim.x;
    for (unsigned int i = threadIdx.x; i < NUM_BINS; i+=blockDim.x) {
        hist_temp[i] = 0;
    }
    __syncthreads();
    if (index < num_elements) {
        atomicAdd(&(hist_temp[input[index]]), 1);
    }
    __syncthreads();
    for (unsigned int i = threadIdx.x; i < NUM_BINS; i+=blockDim.x) {
        atomicAdd(&(bins[i]), hist_temp[i]);
    }
}
```

At first, I implement the histogram kernel by going through all the first NUM\_BINS elements in hist\_temp, and use several if/else pairs to determine whether and how a value should be given to hist\_temp[i].

However, this kind of iteration settings is not necessary, because I soon realized that not all elements are going to be used: I just need to focus on the  $\text{threadIdx.x} + i \times \text{blockDim.x}$  elements. Then I can reduce the iterations by increasing my steps.

Another optimization that I tried is to set a condition for executing `atomicAdd`. This operation actually took longer time than I expected so it is also meaningful to reduce its execution as much as possible: therefore, I used “if ( $\text{index} < \text{num\_elements}$ )” to filter out useless actions.

I also tried to use `memset()` to replace the first iteration, this greatly simplify the codes but the execution time is not changing that much (or not changing).

The optimizations that I performed reduced approximately 0.00038ms for a inputlength of 400000.

2. Which optimizations you chose in the end and why?

**ANSWER:** I chose the first and second optimization, because by comparing the execution time period, they produced significant improvement in efficiency. The last optimization is not adopted because the optimization, compared to the new std library (`string.h`) that is involved, is not a good trade.

The final version of my optimized code shows 0.000227ms of execution time with a input length of 400000.

3. How many global memory reads are being performed by your kernel? Explain

**ANSWER:** In the histogram kernel, each thread performs one global memory read from the input array. So there are *input\_length* global memory reads in total.

4. How many atomic operations are being performed by your kernel? Explain

**ANSWER:** The number of atomic operations is  $2 \times \text{input\_length}$ . Because each thread performs two atomic operation (`atomicAdd`) to update the shared histogram using 1 and `hist_temp[i]`.

5. How much shared memory is used in your code? Explain

**ANSWER:** The `hist_temp` array is an unsigned int[] containing `NUM_BINS` elements, which takes  $4 \times \text{NUM\_BINS}$  bytes. Each block contains one such array, and I assigned 128 threads for each block. So the total memory usage will be  $(\text{input\_length}/128) \times 4 \times \text{NUM\_BINS}$  bytes.

6. How would the value distribution of the input array affect the contention among threads? For instance, what contentions would you expect if every element in the array has the same value?

**ANSWER:** High contention will occur if all elements in the input array have the same value. This is because every thread will be updating the same

histogram bin, leading to serialized atomic operations and reduced parallel efficiency. A diverse input array can let updates occur at different bins, thus to reduce contention and improve performance.

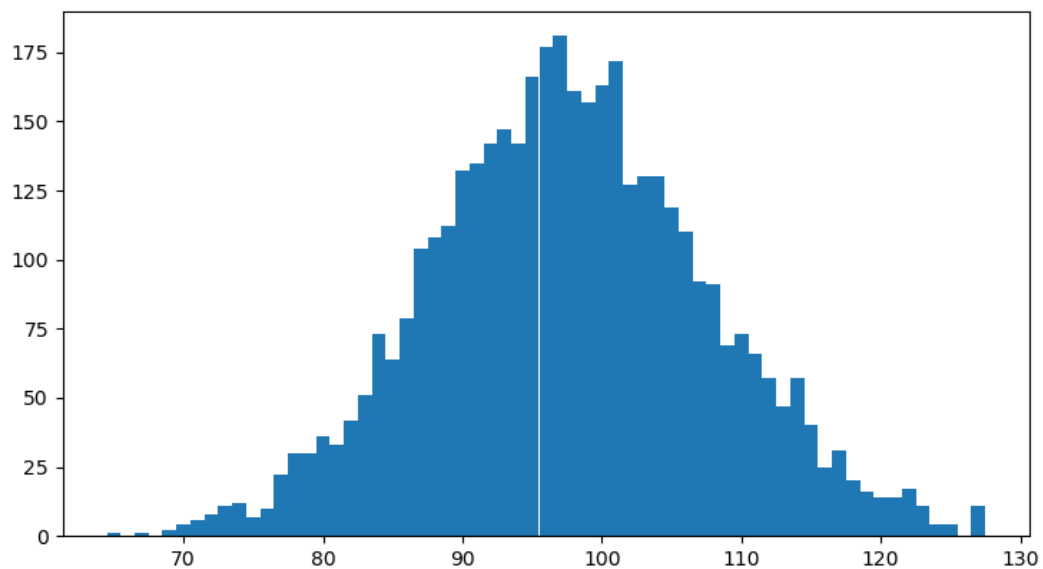
7. Plot a histogram generated by your code and specify your input length, thread block and grid.

**ANSWER:** The histogram are generated using Python matplotlib. The parameters are set as:

*input\_length=400000*

[histogram kernel] *block\_dim=128, grid\_dim=3125*

[convert kernel] *block\_dim=128, grid\_dim=32*



8. For a input array of 1024 elements, profile with Nvidia Nsight and report Shared Memory Configuration Size and Achieved Occupancy. Did Nvsight report any potential performance issues?

**ANSWER:** I have the cuda program tested with 128 threads per block the result shows:

Shared Memory Configuration Size	Kbyte	65.54
Achieved Occupancy	%	12.48

There are two warnings that “This kernel’s theoretical occupancy (50.0%) is limited by the required amount of shared memory.” and “The grid for this launch is configured to execute only 8 blocks, which is less than the GPU’s 40 multiprocessors. This can underutilize some multiprocessors.”

I increased threads per block to 1024, and tested again:

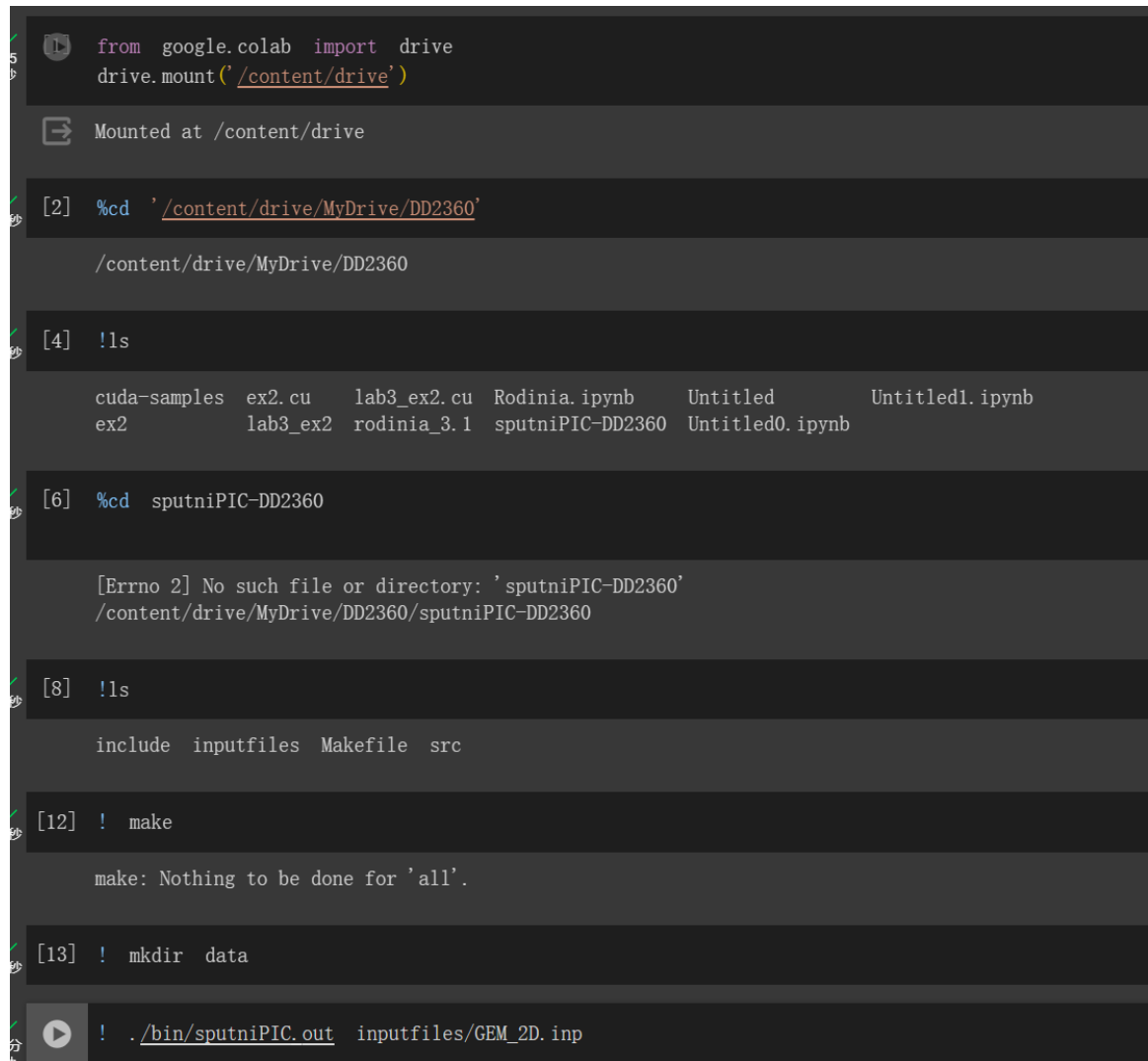
Shared Memory Configuration Size	Kbyte	32.77
Achieved Occupancy	%	96.40

There are two warnings that “This kernel grid is too small to fill the available resources on this device, resulting in only 0.0 full waves across all SMs. Look at Launch Statistics for more details.” and “The grid for this launch is configured to execute only 1 blocks, which is less than the GPU's 40 multiprocessors.”

## Ex2. A Particle Simulation Application

1. Describe the environment you used, what changes you made to the Makefile, and how you ran the simulation.

**ANSWER:** Google is used to run the code in this exercise. According to the changes of the Makefile, the ARCH is changed to sm\_75. To run the simulation, I followed the instructions of the introduction of spunitGPU.pdf, and ran the following commands:



```
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[2] %cd '/content/drive/MyDrive/DD2360'

/content/drive/MyDrive/DD2360

[4] !ls

cuda-samples  ex2. cu      lab3_ex2. cu  Rodinia. ipynb  Untitled      Untitled1. ipynb
ex2           lab3_ex2    rodinia_3.1   sputniPIC-DD2360  Untitled0. ipynb

[6] %cd sputniPIC-DD2360

[Errno 2] No such file or directory: 'sputniPIC-DD2360'
/content/drive/MyDrive/DD2360/sputniPIC-DD2360

[8] !ls

include  inputfiles  Makefile  src

[12] ! make

make: Nothing to be done for 'all'.

[13] ! mkdir data

! ./bin/sputniPIC.out inputfiles/GEM_2D.inp
```

2. Describe your design of the GPU implementation of mover\_PC() briefly.

**ANSWER:** I allocated GPU memory for particles in the Particles.cu file. Then I added the method which is used for launching GPUs to do simulation into the corresponding head file. I also modified the main function in sputniPIC.cpp to let it invoke the GPU method.

3. Compare the output of both CPU and GPU implementation to guarantee that your GPU implementations produce correct answers.

**ANSWER:** I opened the .vtk files generated by the simulations and compared the output.

For the file rhoe\_10.vtk: they are almost the same.

LOOKUP_TABLE default	LOOKUP_TABLE default
-1.72137e-18	-1.93935e-18
-1.82578e-18	-1.93936e-18
-1.9288e-18	-1.93936e-18
-1.64438e-18	-1.93936e-18
-1.82318e-18	-1.93936e-18
-1.80315e-18	-1.93936e-18
-1.9008e-18	-1.93936e-18
-1.9595e-18	-1.93936e-18
-1.96062e-18	-1.93936e-18
-1.90236e-18	-1.93936e-18
-1.93711e-18	-1.93936e-18
-1.92202e-18	-1.93936e-18
-1.76678e-18	-1.93936e-18
-1.69521e-18	-1.93936e-18
-2.01186e-18	-1.93936e-18
-1.9444e-18	-1.93936e-18
-1.92596e-18	-1.93936e-18
-1.83411e-18	-1.93936e-18

For the file rho\_net\_10.vtk: The output of the GPU are all zero, but the CPU doesn't, but the difference between each output is tiny.

LOOKUP_TABLE default	LOOKUP_TABLE default
-8.89965e-05	0
-0.00016443	0
6.73247e-05	0
0.000180923	0
3.86278e-06	0
0.000191473	0
3.31744e-05	0
-3.29298e-05	0
0.000211425	0
0.00020405	0
2.30237e-05	0
0.000279595	0
0.000127751	0
-0.000183836	0
-0.000216214	0
0.000243326	0
0.000142679	0
-9.86999e-05	0

For the file rhoi\_10.vtk: they are almost the same.

LOOKUP_TABLE default	LOOKUP_TABLE default
1.93935e-18	1.94788e-18
1.93936e-18	1.91674e-18
1.93936e-18	1.91703e-18
1.93936e-18	1.89704e-18
1.93936e-18	2.09185e-18
1.93936e-18	1.99951e-18
1.93936e-18	1.88832e-18
1.93936e-18	1.85028e-18
1.93936e-18	1.87391e-18
1.93936e-18	1.8475e-18
1.93936e-18	2.0066e-18
1.93936e-18	1.98433e-18
1.93936e-18	1.87581e-18
1.93936e-18	1.90086e-18
1.93936e-18	1.91322e-18
1.93936e-18	2.05826e-18
1.93936e-18	1.91482e-18
1.93936e-18	1.95792e-18

In conclusion, the result is correct.

4. Compare the execution time of your GPU implementation with its CPU version.

**ANSWER:** CPU(without the time saving data in data folder):

```
*****
Tot. Simulation Time (s) = 64.105
Mover Time / Cycle (s) = 3.52769
Interp. Time / Cycle (s) = 2.59901
*****
```

CPU(including saving data):

```
*****
Tot. Simulation Time (s) = 93.0991
Mover Time / Cycle (s) = 3.88624
Interp. Time / Cycle (s) = 5.08455
*****
```

Gpu(without the time saving data in data folder):

```
*****
Tot. Simulation Time (s) = 29.3349
Mover Time / Cycle (s) = 0.0270004
Interp. Time / Cycle (s) = 2.61765
*****
```

GPU(including saving data):

```
*****
Tot. Simulation Time (s) = 49.4001
Mover Time / Cycle (s) = 0.0550164
Interp. Time / Cycle (s) = 4.55708
*****
```

## Contributions

### Jian Wen

Implemented the exercise 1 (Histogram and Atomics) in the assignment.

### Heng Kang

Implemented the exercise 2 (A Particle Simulation Application) in the assignment.