DD2360
Assignment III: CUDA Basics

Rui Shi

December 20, 2022

Link to github: `https://github.com/RuiShi324/DD2360`

1 Exercise 1 - Your first CUDA program and GPU performance metrics

1. Explain how the program is compiled and run.

   Answer: As the GPU I am using is Tesla T4 of Google colab, nvcc -arch should be specified as SM_75.

   To compile .cu file: !nvcc -arch=sm_75 ex1.cu -o ex1.out

   To run .out file: !./ex1.out $< InputLength >$

   To profile program: !/usr/local/cuda-11/nv-nsight-cu-cli ./ex1.out $< InputLength >$

2. For a vector length of N:

   (a) How many floating operations are being performed in your vector add kernel? N floating operations

   (b) How many global memory reads are being performed by your kernel? 2N, as there are two vectors, each of them is performed N global memory reads by the kernel.

3. For a vector length of 1024:

   (a) Explain how many CUDA threads and thread blocks you used.

   Answer: For a vector length of 1024, I used 128 threads per block, and 1024 / 128 = 8 blocks in total.

(b) Profile your program with Nvidia Nsight. What Achieved Occupancy did you get?

Answer: The achieved occupancy could get to 12.13%.

4. Now increase the vector length to 131070:

   (a) Did your program still work? If not, what changes did you make?

   Yes, it still works.

   (b) Explain how many CUDA threads and thread blocks you used.

   Answer: I used still 1024 threads per block, and 131070 / 1024 = 128 blocks in total.

   (c) Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?

   Answer: The Achieved Occupancy could get 77.72%.

5. Further increase the vector length (try 6-10 different vector length), plot a stacked bar chart showing the breakdown of time including (1) data copy from host to device (2) the CUDA kernel (3) data copy from device to host. For this, you will need to add simple CPU timers to your code regions.

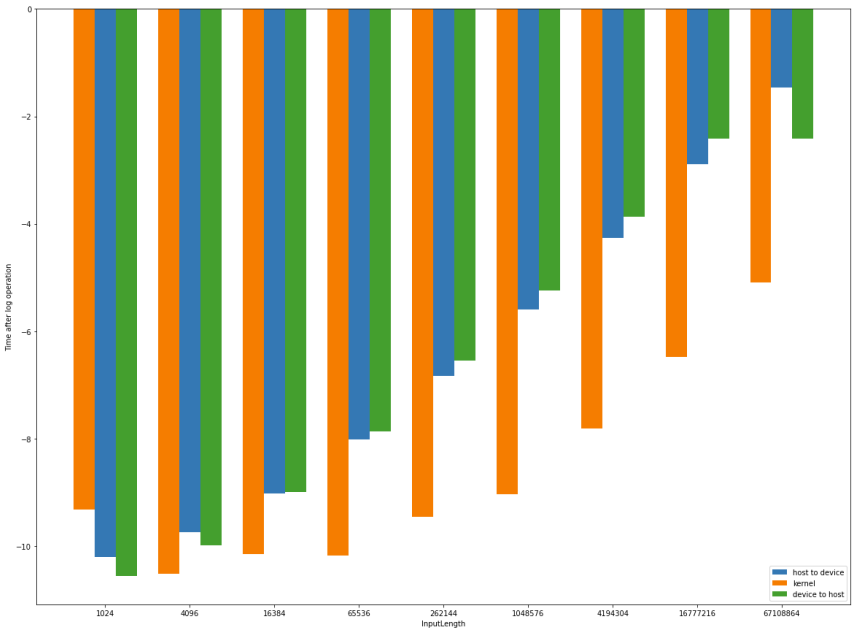

Figure 1: Breakdown of time after log operation

To have a better review of time, I did log operation on them. As shown in figure 1, time increases exponentially.

2 Exercise 2 - 2D Dense Matrix Multiplication

1. Name three application domains of matrix multiplication.

   Answer: Deep learning (calculate loss, etc.), solution of linear systems of equations, signal processing, transformation of co-ordinate systems, etc.

2. How many floating operations are being performed in your matrix multiply kernel?

   Answer: $2 \times numARows \times numAColumns \times numBColumns$

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

   Answers: $2 \times numARows \times numAColumns \times numBColumns$ (same as question 2)

4. For a matrix A of (128x128) and B of (128x128):

   (a) Explain how many CUDA threads and thread blocks you used.

   Answer: In this case, I used $16 \times 16 = 256$ threads per block, and 64 blocks in total.

   (b) Profile your program with Nvidia Nsight. What Achieved Occupancy did you get?

   Answer: The Achieved Occupancy could get 38.58%.

5. For a matrix A of (511x1023) and B of (1023x4094):

   (a) Did your program still work? If not, what changes did you make?

   Yes, it still worked.

   (b) Explain how many CUDA threads and thread blocks you used.

   Answer: In this case, I used $32 \times 32 = 1024$ threads per block, and [511/32] $\times$[4094/32]= 16 $\times$128 = 2048 blocks

   (c) Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?

   Answer: The Achieved Occupancy could get 98.28%.

6. Further increase the size of matrix A and B, plot a stacked bar chart showing the breakdown of time including (1) data copy from host to device (2) the CUDA kernel

(3) data copy from device to host. For this, you will need to add simple CPU timers to your code regions. Explain what you observe.

7. Now, change DataType from double to float, re-plot the a stacked bar chart showing the time breakdown. Explain what you observe.

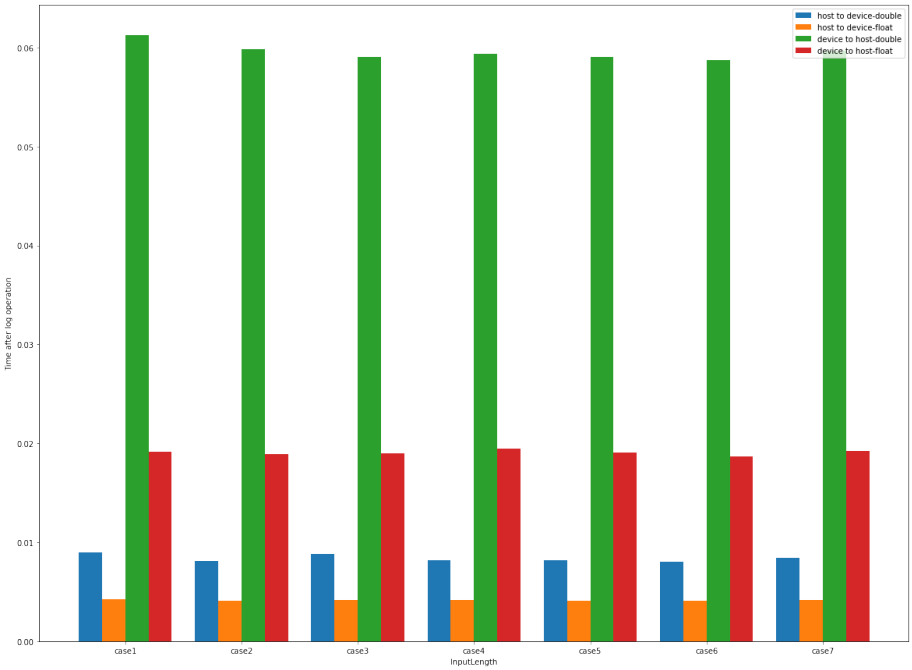

Figure 2: Comparison of copy time between double and float

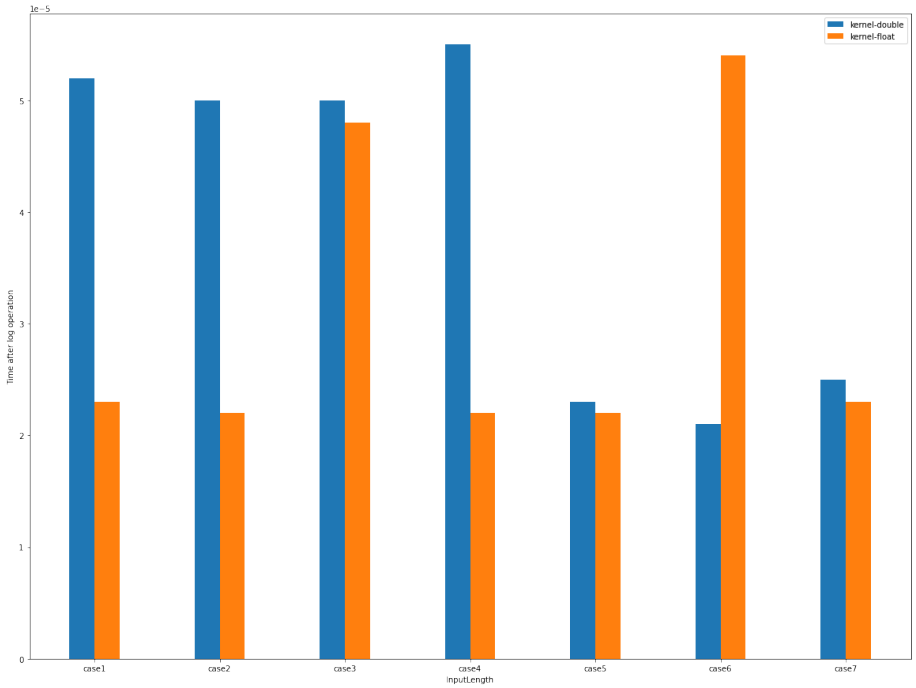

Figure 3: Comparison of kernel time between double and float

As shown in 2 and 3, when changing DataType from double to float, time decreases significantly. While increasing size of matrices A and B, time trends are stable.

3 Exercise 3 - Histogram and Atomics

1. Describe all optimizations you tried regardless of whether you committed to them or abandoned them and whether they improved or hurt performance.
   (a) Use shared memory to improve performance.
   (b) Use atomics to access to global memory directly.
   (c) Use one block, for it could reduce the times of accessing global memory.
2. Which optimizations you chose in the end and why?

   Answer: I chose one thread per block because I tried this method and found it efficient. In theory, it is all easy to understand and implement.
3. How many global memory reads are being performed by your kernel? Explain

Answer: For a vector length of N, as I selected the method of using 1 thread per block, N global memory reads are being performed by the kernel.

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

   Answer: For a vector length of N, one atomic operation is being performed by the kernel.

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

   Answer: num_blocks $\times$ num_bins $\times$ 4 bytes = 16384 $\times$ num_blocks 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: The more values are the same, the larger execution time will be, or the smaller contention among threads.

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

   Answer: Here I set input length as 8192, so 8 thread blocks and gird size = 8.

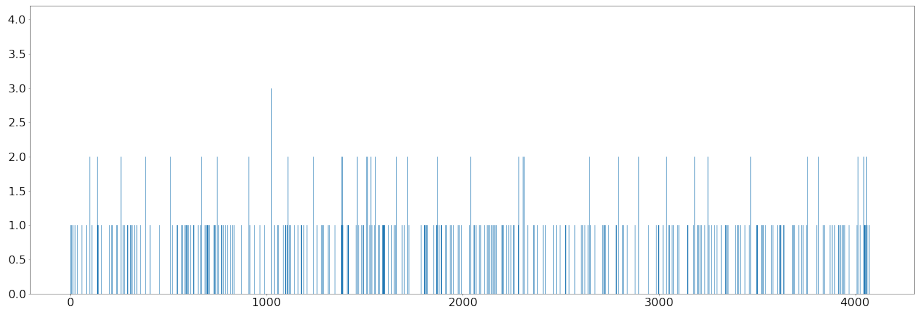

Figure 4: Histogram

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: From the Nvidia Nsight, The Shared Memory Configuration Size is 32.77 Kbytes, and the Achieved Occupancy is 91.39.

   Yes, it reports as shown in the figure 5.

```
    Waves Per SM                                                                          0.03
    ---------------------------------------------------------------------- ----------- -------------
    WRN   The grid for this launch is configured to execute only 1 blocks, which is less than the GPU's 40
          multiprocessors. This can underutilize some multiprocessors. If you do not intend to execute this kernel
          concurrently with other workloads, consider reducing the block size to have at least one block per
          multiprocessor or increase the size of the grid to fully utilize the available hardware resources.

    Section: Occupancy
    ---------------------------------------------------------------------- ----------- -------------
    Block Limit SM                                                           block              16
    Block Limit Registers                                                    block               4
    Block Limit Shared Mem                                                   block               4
    Block Limit Warps                                                        block               1
    Theoretical Active Warps per SM                                          warp               32
    Theoretical Occupancy                                                    %                 100
    Achieved Occupancy                                                       %               96.04
    Achieved Active Warps Per SM                                             warp            30.73
    ---------------------------------------------------------------------- ----------- -------------

  convert_kernel(unsigned int*, unsigned int), 2022-Dec-20 22:05:09, Context 1, Stream 7
    Section: GPU Speed Of Light
    ---------------------------------------------------------------------- ----------- -------------
    DRAM Frequency                                                    cycle/nsecond           4.76
    SM Frequency                                                      cycle/usecond         558.57
    Elapsed Cycles                                                           cycle           2,163
    Memory [%]                                                               %                1.69
    SOL DRAM                                                                 %                1.69
    Duration                                                                 usecond          3.87
    SOL L1/TEX Cache                                                         %               12.77
    SOL L2 Cache                                                             %                1.09
    SM Active Cycles                                                         cycle          100.22
    SM [%]                                                                   %                0.59
    ---------------------------------------------------------------------- ----------- -------------
    WRN   This kernel grid is too small to fill the available resources on this device, resulting in only 0.1 full
          waves across all SMs. Look at Launch Statistics for more details.

    Section: Launch Statistics
    ---------------------------------------------------------------------- ----------- -------------
    Block Size                                                                               1,024
    Function Cache Configuration                                              cudaFuncCachePreferNone
    Grid Size                                                                                    4
    Registers Per Thread                                              register/thread            16
    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                                     byte/block                0
    Threads                                                                  thread          4,096
    Waves Per SM                                                                              0.10
    ---------------------------------------------------------------------- ----------- -------------
    WRN   The grid for this launch is configured to execute only 4 blocks, which is less than the GPU's 40
          multiprocessors. This can underutilize some multiprocessors. If you do not intend to execute this kernel
          concurrently with other workloads, consider reducing the block size to have at least one block per
          multiprocessor or increase the size of the grid to fully utilize the available hardware resources.
```

Figure 5: Potential performance issues by Nvsight report

4 Exercise 4 - A Particle Simulation Application

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

   Answer: I used Google Colab, and assigned GPU Tesla T4. In the Makefile, I changed sm_30 to sm_75. To run the simulation, I clone the repo into Google drive, and do the following commands:

   !git clone https://github.com/KTH-HPC/sputniPIC-DD2360.git

   %cd sputniPIC-DD2360

!make

!./bin/sputniPIC.out inputfiles/GEM$_2$*D.inpDescribeyourdesignoftheGPUimplementationofmover_P*

Answer: The GPU implementation of mover_PC() follows the steps of GPU programming. First of all, allocate memory on the device(GPU) and then do the kernel execution and lastly copy the value from device to host.

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

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

```
-------------------------
sputniPIC Sim. Parameters
-------------------------
Number of species    = 4
Number of particles of species 0 = 4096000       (MAX = 4096000)  QOM = -64
Number of particles of species 1 = 4096000       (MAX = 4096000)  QOM = 1
Number of particles of species 2 = 4096000       (MAX = 4096000)  QOM = -64
Number of particles of species 3 = 4096000       (MAX = 4096000)  QOM = 1
x-Length                 = 40
y-Length                 = 20
z-Length                 = 1
Number of cells (x)      = 256
Number of cells (y)      = 128
Number of cells (z)      = 1
Time step                = 0.25
Number of cycles         = 10
Results saved in: data
************************************************
**  Initialize GEM Challenge with Pertubation  **
************************************************
** B0x = 0.0195
** B0y = 0
** B0z = 0
** Delta (current sheet thickness) = 0.5
** rho species 0 = 1 CURRENT SHEET
** rho species 1 = 1 CURRENT SHEET
** rho species 2 = 0.02 BACKGROUND
** rho species 3 = 0.02 BACKGROUND
************************************************
```

Figure 6: Results of CPU

```
Number of species    = 4
Number of particles of species 0 = 4096000       (MAX = 4096000)  QOM = -64
Number of particles of species 1 = 4096000       (MAX = 4096000)  QOM = 1
Number of particles of species 2 = 4096000       (MAX = 4096000)  QOM = -64
Number of particles of species 3 = 4096000       (MAX = 4096000)  QOM = 1
x-Length                 = 40
y-Length                 = 20
z-Length                 = 1
Number of cells (x)      = 256
Number of cells (y)      = 128
Number of cells (z)      = 1
Time step                = 0.25
Number of cycles         = 10
Results saved in: data
************************************************
**  Initialize GEM Challenge with Pertubation  **
************************************************
** B0x = 0.0195
** B0y = 0
** B0z = 0
** Delta (current sheet thickness) = 0.5
** rho species 0 = 1 CURRENT SHEET
** rho species 1 = 1 CURRENT SHEET
** rho species 2 = 0.02 BACKGROUND
** rho species 3 = 0.02 BACKGROUND
************************************************

**********************
```

Figure 7: Results of GPU

```
************************************
   Tot. Simulation Time (s) = 56.0446
   Mover Time / Cycle   (s) = 3.01807
   Interp. Time / Cycle (s) = 2.19789
************************************
```

Figure 8: Time of CPU

```
************************************
   Tot. Simulation Time (s) = 35.4691
   Mover Time / Cycle   (s) = 0.85502
   Interp. Time / Cycle (s) = 2.15276
************************************
```

Figure 9: Time of GPU

As shown in the figures above, the results are the same which means that the GPU im-

plementation produces correct answers. For CPU, it executes about 56.0446 s, while GPU executes about 35.4694 s, showing that GPU improves about 36.71%.