Assignment II: CUDA Basics I

Exercise 1 - Your first CUDA program and GPU performance metrics

1. Explain how the program is compiled and run.

First, we need to compile the program(.c) with the NVIDIA CUDA compiler (nvcc)

```
nvcc -arch=sm_75 lab2exercise1.cu -o exercise1.out
```

Then we need to run the compiled executable file directly. The first execution parameter is the file name and the second is the input length of the vector we need to calculate.

```
./exercise1.out <InputLength>
```

2. For a vector length of N:

1. How many floating operations are being performed in your vector add kernel?

For the vector length of N, the total number of floating-point operations is 2N, N addition, and N assignment respectively.

At each thread, we just have one addition and one assignment.

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

There are 2N global reads performed by our kernel totally and two in each thread, since the `cudamalloc()` function will allocate vector into global space at GPU.

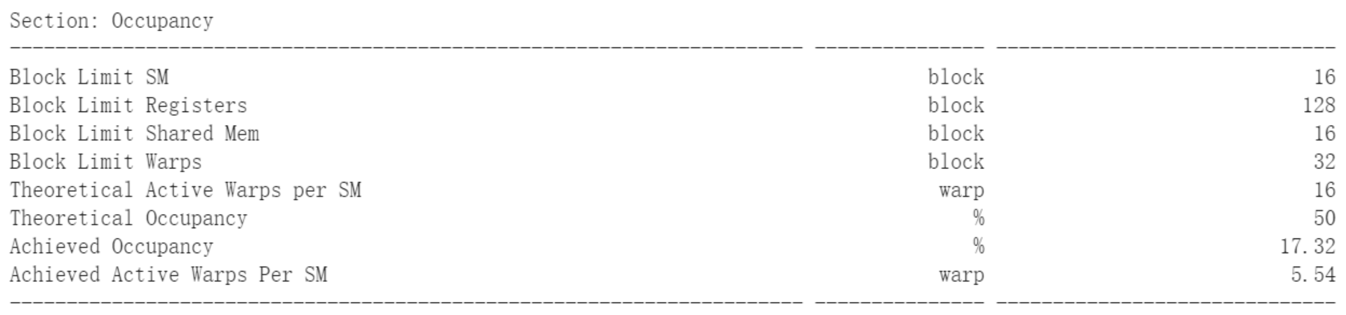
3. For a vector length of 1024:

1. Explain how many CUDA threads and thread blocks you used.

In our program, the number of thread per block is defined as 256. So the number of thread blocks that we used should be: [(1024 + 256 - 1)/256] = 4.

2. Profile your program with Nvidia Nsight. What Achieved Occupancy did you get?

As the figure below shows, the achieved occupancy is: 17.32%.



```
Section: Occupancy
---------------------------------------------------------------------- --------------- ------------------------------
Block Limit SM                                                                   block                             16
Block Limit Registers                                                            block                            128
Block Limit Shared Mem                                                           block                             16
Block Limit Warps                                                                block                             32
Theoretical Active Warps per SM                                                   warp                             16
Theoretical Occupancy                                                                %                             50
Achieved Occupancy                                                                   %                          17.32
Achieved Active Warps Per SM                                                      warp                           5.54
---------------------------------------------------------------------- --------------- ------------------------------
```

4 .Now increase the vector length to 131070:

1. Did your program still work? If not, what changes did you make?

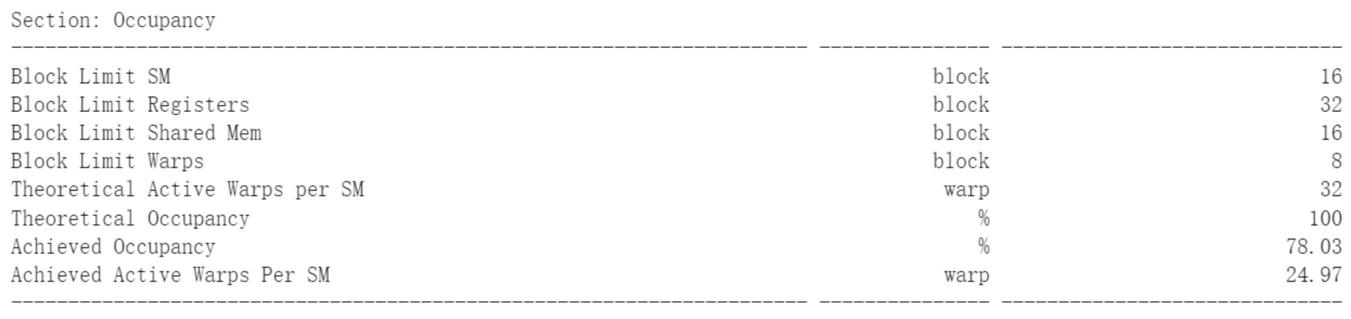
Still work. We changed the number of thread per block as: 1024 so that the number of block would not be too large.

2. Explain how many CUDA threads and thread blocks you used.

Since the number of thread per block is defined as: 1024. The number of thread block that we used is: [(131070 + 1024 - 1)/1024] = 128.

3. Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?

As the figure below shows, the achieved occupancy is: 78.03%.

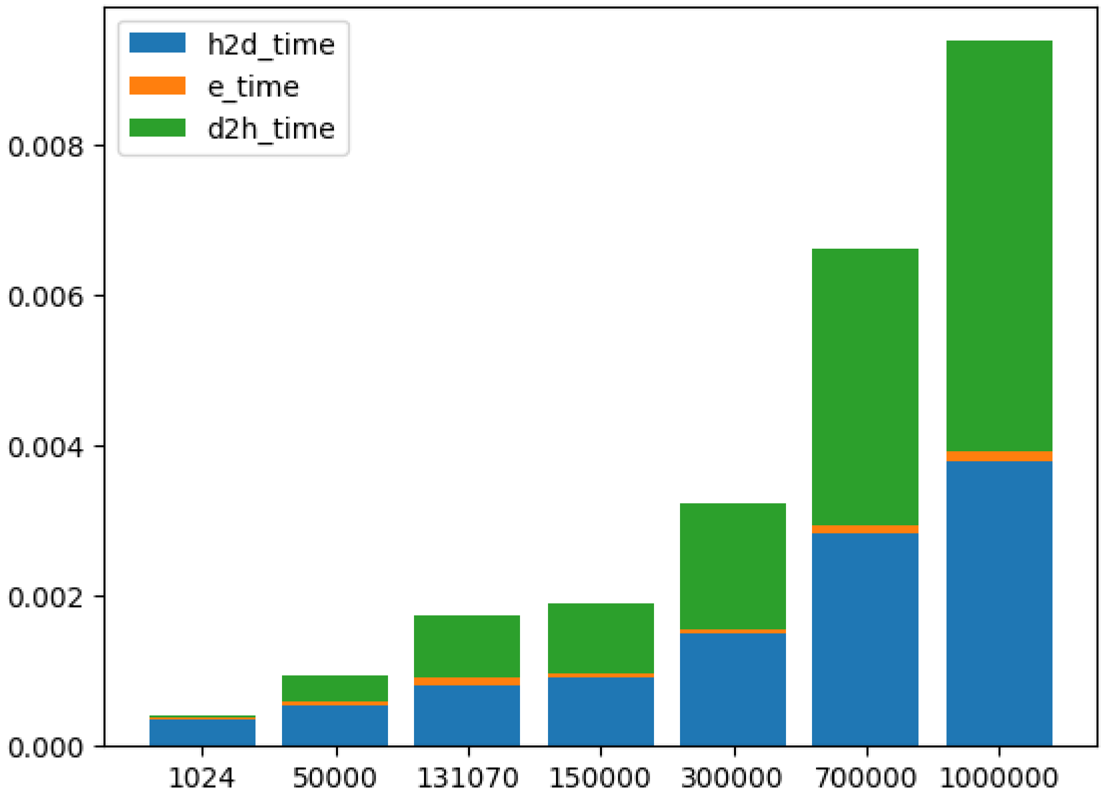


```
Section: Occupancy
---------------------------------------------------------------------- --------------- ------------------------------
Block Limit SM                                                                   block                             16
Block Limit Registers                                                            block                             32
Block Limit Shared Mem                                                           block                             16
Block Limit Warps                                                                block                              8
Theoretical Active Warps per SM                                                   warp                             32
Theoretical Occupancy                                                                %                            100
Achieved Occupancy                                                                   %                          78.03
Achieved Active Warps Per SM                                                      warp                          24.97
---------------------------------------------------------------------- --------------- ------------------------------
```

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.

As the vector length increases, the time for data copying from host to device and data copying from device to host increase almost linearly, but the execution time in the kernel remains stable with negligible time consumption.

input size	1024	50000	131070	150000	300000	700000	1000000
h2d_time	0.000353	0.000542	0.000816	0.000903	0.001501	0.002825	0.003782
execution _time	3.2e-05	5.1e-05	9.5e-05	6.5e-05	6.1e-05	9.9e-05	0.000132
d2h_time	2.5e-05	0.000338	0.000817	0.000926	0.001672	0.003674	0.005454



Exercise 2 - 2D Dense Matrix Multiplication

1. Name three applications domains of matrix multiplication.

1. Deep learning: In the convolution and fully connected layers of the neuron network, matrix multiplication is necessary.
2. digital signal processing like FFT, DCT, and so on.
3. SLAM: in the karma filter, matrix multiplication is used to update the state.

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

We assume that the sizes of the two matrices multiplied are a * b and b * c respectively, and the matrix size we obtained is a * c. For each element in the result matrix, b multiplication floating operations, (b-1) addition floating operations and 1 assignment operation are required respectively, that is, a total of 2b operations. For a * c elements, a total of 2b * a * c operations are required.

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

We assume that the sizes of the two matrices multiplied are a * b and b * c respectively, and global memory reads are a * b + b *c.

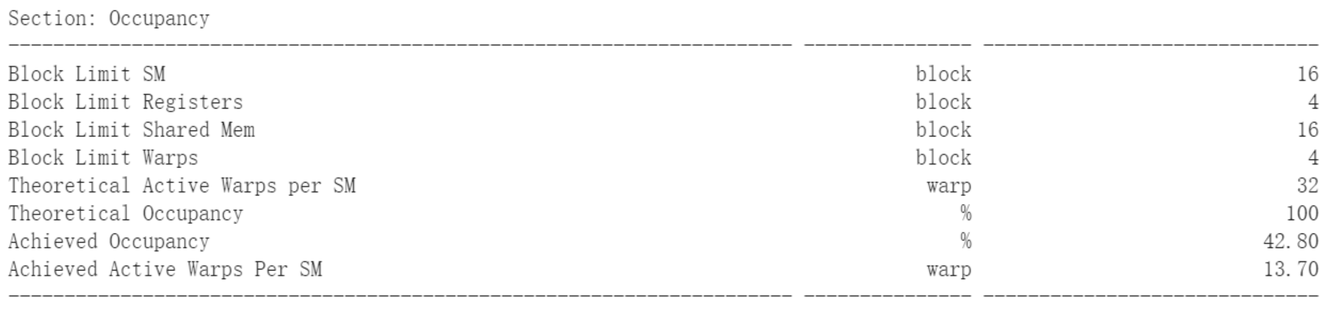
4. For a matrix A of (128x128) and B of (128x128):

1. Explain how many CUDA threads and thread blocks you used.

In the program, we define the number of thread per block as: 16 * 16 = 256. The number of block should be: [(128+16-1)/16]^2 = 64.

2. Profile your program with Nvidia Nsight. What Achieved Occupancy did you get?

As the figure below shows, the achieved occupancy is: 42.80%.



```
Section: Occupancy
---------------------------------------------------------------------- --------------- ------------------------------
Block Limit SM                                                                   block                             16
Block Limit Registers                                                            block                              4
Block Limit Shared Mem                                                           block                             16
Block Limit Warps                                                                block                              4
Theoretical Active Warps per SM                                                   warp                             32
Theoretical Occupancy                                                                %                            100
Achieved Occupancy                                                                   %                          42.80
Achieved Active Warps Per SM                                                      warp                          13.70
---------------------------------------------------------------------- --------------- ------------------------------
```

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

1. Did your program still work? If not, what changes did you make?

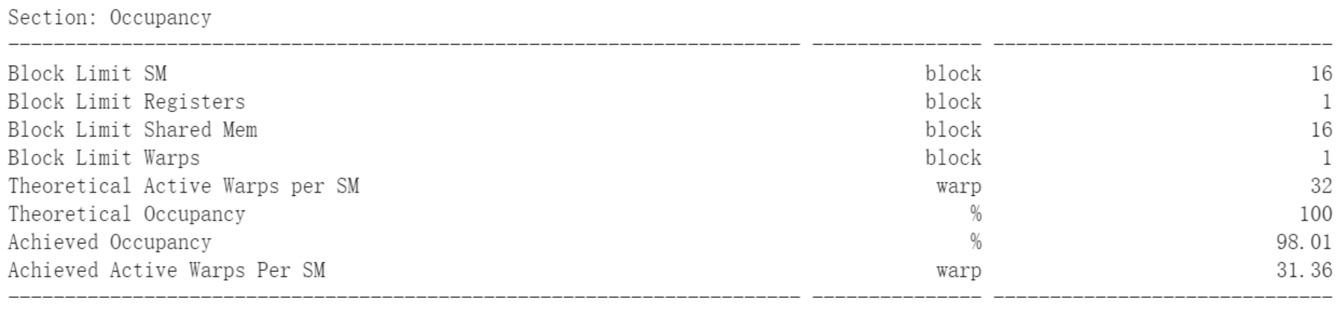
Still work. We changed the number of thread per block as: 32 * 32.

2. Explain how many CUDA threads and thread blocks you used.

The number of thread per block is defined as: 32 * 32 = 1024. The number of block should be: [(511 + 32 - 1)/32] * [(4094 + 32 - 1)/32] = 2048.

3. Profile your program with Nvidia Nsight. What Achieved Occupancy do you get now?

As the figure below shows, the achieved occupancy is: 98.01%.



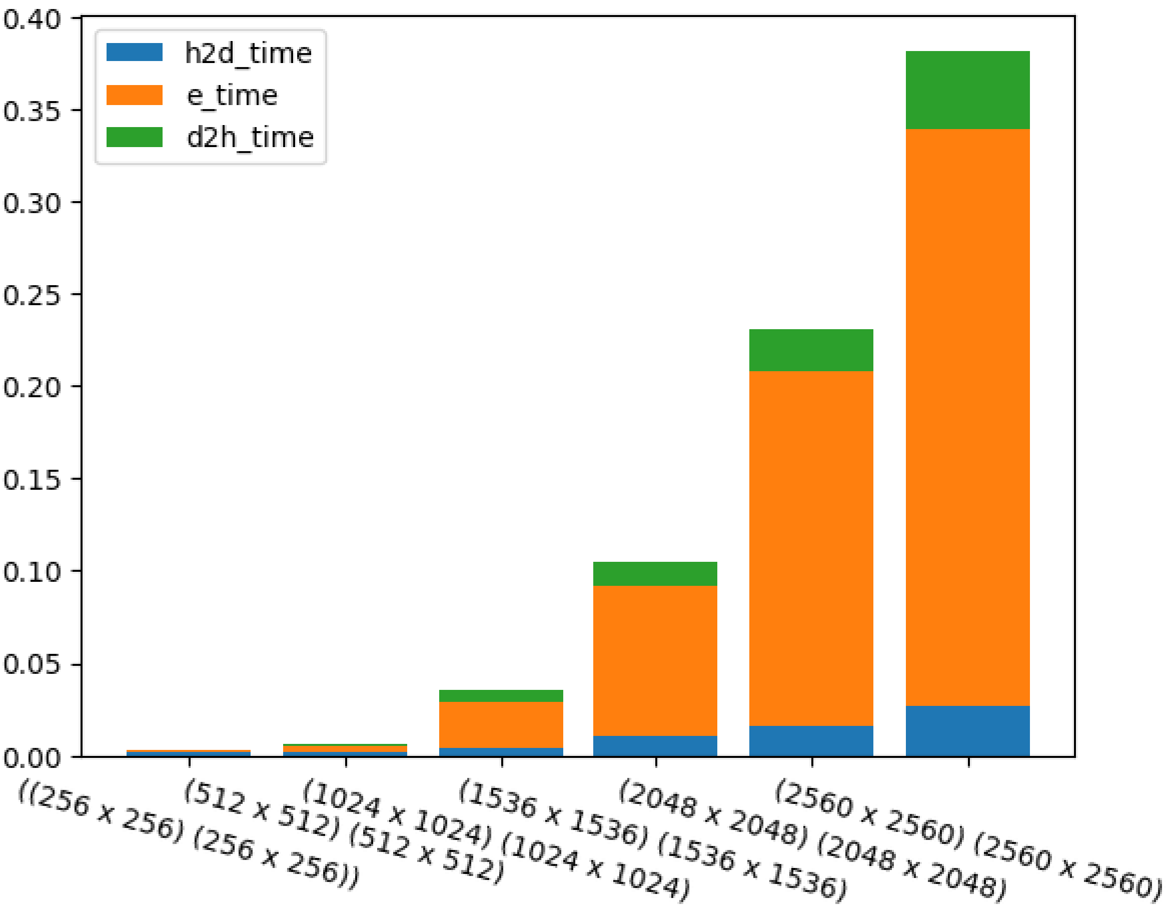
```
Section: Occupancy
---------------------------------------------------------------------- --------------- ------------------------------
Block Limit SM                                                                   block                             16
Block Limit Registers                                                            block                              1
Block Limit Shared Mem                                                           block                             16
Block Limit Warps                                                                block                              1
Theoretical Active Warps per SM                                                   warp                             32
Theoretical Occupancy                                                                %                            100
Achieved Occupancy                                                                   %                          98.01
Achieved Active Warps Per SM                                                      warp                          31.36
---------------------------------------------------------------------- --------------- ------------------------------
```

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.

The time that data copy from host to device and data copy from device to host is sounding linearly rise as the input size rises. In small input size, the data movement time takes up most of the time, but as execution time in the kernel increases dramatically, it gradually takes up a major portion of the time. This is due to the fact that we are applying a thread to calculate the sum of the products of rows and columns, and as the number of rows and columns increases the number of floating-point operations performed increases, so the execution time increases dramatically and dominates.

input size	((256 x 256) (256 x 256))	(512 x 512) (512 x 512)	(1024 x 1024) (1024 x 1024)	(1536 x 1536) (1536 x 1536)	(2048 x 2048) (2048 x 2048)	(2560 x 2560) (2560 x 2560)

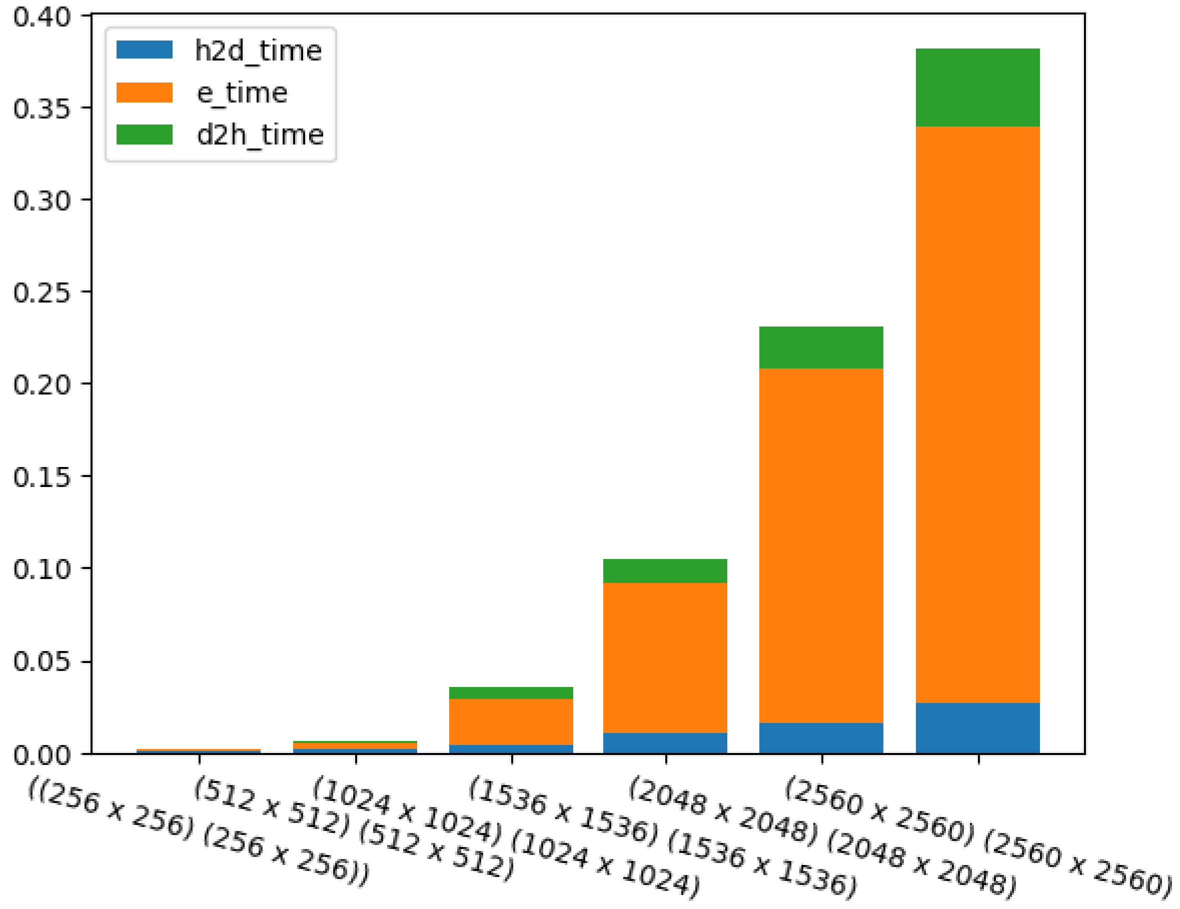
input size	((256 x 256) (256 x 256))	(512 x 512) (512 x 512)	(1024 x 1024) (1024 x 1024)	(1536 x 1536) (1536 x 1536)	(2048 x 2048) (2048 x 2048)	(2560 x 2560) (2560 x 2560)
h2d_time	0.001912	0.001346	0.00411	0.010207	0.016338	0.0272
execution _time	0.00052	0.00335	0.024283	0.081044	0.191432	0.312322
d2h_time	0.000466	0.001615	0.006646	0.013542	0.023271	0.042542



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

The overall time shares and trends are similar to those using the doule data type, but the time spent is less than using the doule data type.

input size	((256 x 256) (256 x 256))	(512 x 512) (512 x 512)	(1024 x 1024) (1024 x 1024)	(1536 x 1536) (1536 x 1536)	(2048 x 2048) (2048 x 2048)	(2560 x 2560) (2560 x 2560)
h2d_time	0.000491	0.001489	0.004224	0.00821	0.018303	0.026075
execution _time	0.000717	0.003346	0.024293	0.081021	0.191421	0.281914
d2h_time	0.000532	0.001788	0.006813	0.013132	0.023835	0.037752


0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
h2d_time
e_time
d2h_time
((256 x 256) (256 x 256))
(512 x 512) (512 x 512)
(1024 x 1024) (1024 x 1024)
(1536 x 1536) (1536 x 1536)
(2048 x 2048) (2048 x 2048)
(2560 x 2560) (2560 x 2560)