Korea Advanced Institute of Science and Technology

School of Electrical Engineering

EE817 GPU Programming and Its Applications Spring 2018

Student: Dinh Vu

Student ID: 20184187

Homework 2

The computer used in this homework contains NVIDIA GeForce 1070 based on Pascal GP104 architecture.

```
20184187@eelab5: ~
20184187@eelab5:~$ nvidia-smi
Wed Mar 28 20:46:30 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.59                 Driver Version: 384.59                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 1070    Off  | 00000000:01:00.0 Off |                  N/A |
|  6%   49C    P0    32W / 200W |      0MiB /  8114MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  GeForce GTX 1070    Off  | 00000000:02:00.0 Off |                  N/A |
|  0%   50C    P5    24W / 200W |      0MiB /  8114MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
20184187@eelab5:~$
```

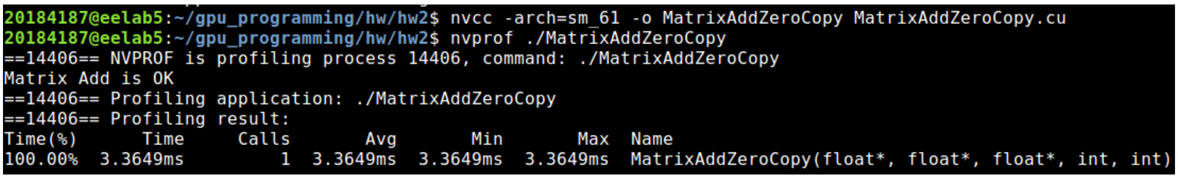
Figure 1. Graphic card information

1. Homework 2.1

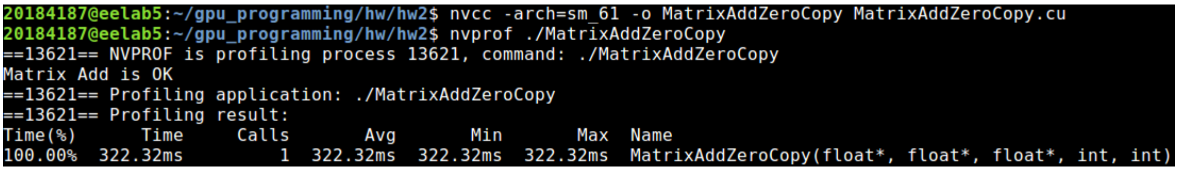
The source code of homework 2.1 is MatrixAddZeroCopy.cu. The execution time of matrix 160×160, 1600×1600 and 16000×16000 is presented in Figure 2 below.

```
20184187@eelab5:~/gpu_programming/hw/hw2$ nvcc -arch=sm_61 -o MatrixAddZeroCopy MatrixAddZeroCopy.cu
20184187@eelab5:~/gpu_programming/hw/hw2$ nvprof ./MatrixAddZeroCopy
==14484== NVPROF is profiling process 14484, command: ./MatrixAddZeroCopy
Matrix Add is OK
==14484== Profiling application: ./MatrixAddZeroCopy
==14484== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
100.00%  44.255us         1  44.255us  44.255us  44.255us  MatrixAddZeroCopy(float*, float*, float*, int, int)
```

(a) Matrix 160×160



(b) Matrix 1600×1600



(c) Matrix 16000×16000

Figure 2. The execution time of addition function for each matrix size

Table 1. The execution time of matrices

Matrix size	160×160	1600×1600	16000×16000
Execution time	44.255 μs	3.3649 ms	322.32 ms

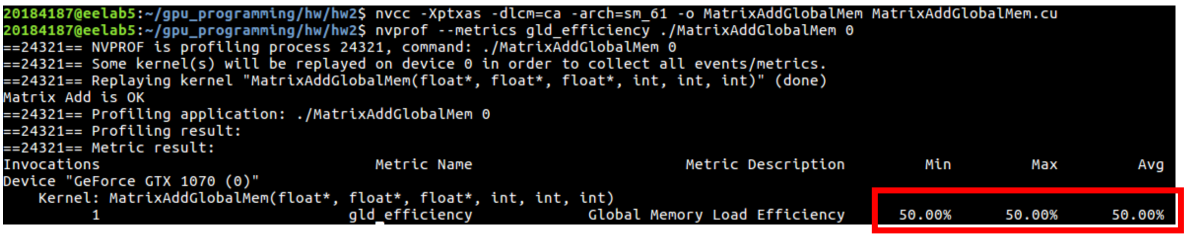
The practical result is corresponding with theorem about using zero-copy memory. Generally, the execution time increases by the size of matrix.

The execution time of matrix 160×160 is lowest because the number of threads of addition matrices 160×160 is lowest and amount of data is acceptable with the bandwidth of zero-copy memory.

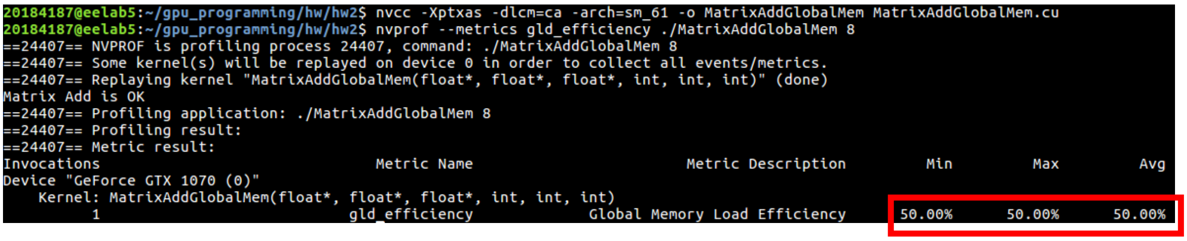
There are two reasons cause the highest execution time for matrix 16000×16000. First, the number of threads is greatest, 1.6×10^7 threads so it is need more time to schedule. Second, the most important reason is that reading and writing the data of 16 millions elements in each matrix, about $3\times1.6\times10^7\times4$ bytes ≈ 183 MB shared by CPU and GPU is limited by PCIe 3.0 bus with bandwidth 985 MB/s.

2. Homework 2.2

The source code for homework 2.2 is MatrixAddGlobalMem.cu. The result is display in Figure 2.1.

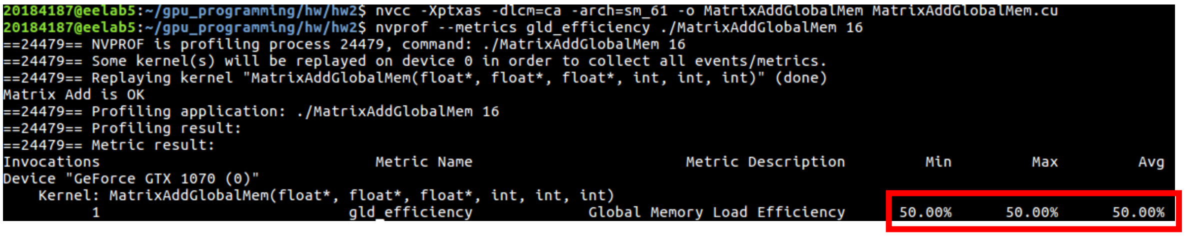


(a) offset = 0



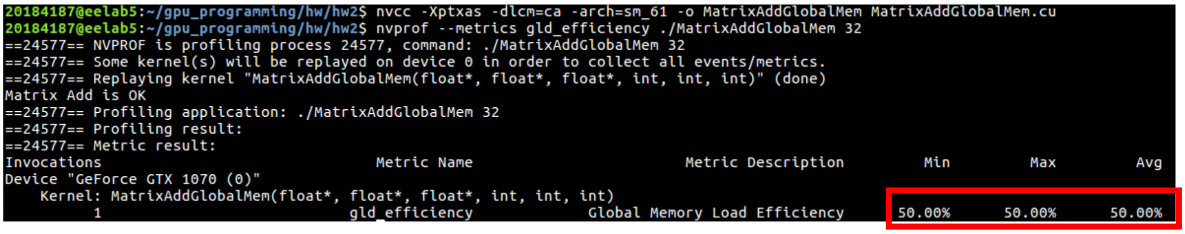
```
20184187@eelab5:~/gpu_programming/hw/hw2$ nvcc -Xptxas -dlcm=ca -arch=sm_61 -o MatrixAddGlobalMem MatrixAddGlobalMem.cu
20184187@eelab5:~/gpu_programming/hw/hw2$ nvprof --metrics gld_efficiency ./MatrixAddGlobalMem 8
==24407== NVPROF is profiling process 24407, command: ./MatrixAddGlobalMem 8
==24407== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==24407== Replaying kernel "MatrixAddGlobalMem(float*, float*, float*, int, int, int)" (done)
Matrix Add is OK
==24407== Profiling application: ./MatrixAddGlobalMem 8
==24407== Profiling result:
==24407== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GTX 1070 (0)"
    Kernel: MatrixAddGlobalMem(float*, float*, float*, int, int, int)
          1                            gld_efficiency             Global Memory Load Efficiency      50.00%      50.00%      50.00%
```

(b) offset = 8



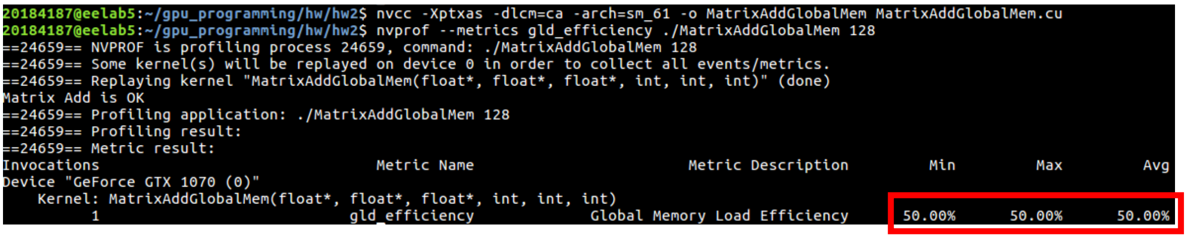
```
20184187@eelab5:~/gpu_programming/hw/hw2$ nvcc -Xptxas -dlcm=ca -arch=sm_61 -o MatrixAddGlobalMem MatrixAddGlobalMem.cu
20184187@eelab5:~/gpu_programming/hw/hw2$ nvprof --metrics gld_efficiency ./MatrixAddGlobalMem 16
==24479== NVPROF is profiling process 24479, command: ./MatrixAddGlobalMem 16
==24479== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==24479== Replaying kernel "MatrixAddGlobalMem(float*, float*, float*, int, int, int)" (done)
Matrix Add is OK
==24479== Profiling application: ./MatrixAddGlobalMem 16
==24479== Profiling result:
==24479== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GTX 1070 (0)"
    Kernel: MatrixAddGlobalMem(float*, float*, float*, int, int, int)
          1                            gld_efficiency             Global Memory Load Efficiency      50.00%      50.00%      50.00%
```

(c) offset = 16



```
20184187@eelab5:~/gpu_programming/hw/hw2$ nvcc -Xptxas -dlcm=ca -arch=sm_61 -o MatrixAddGlobalMem MatrixAddGlobalMem.cu
20184187@eelab5:~/gpu_programming/hw/hw2$ nvprof --metrics gld_efficiency ./MatrixAddGlobalMem 32
==24577== NVPROF is profiling process 24577, command: ./MatrixAddGlobalMem 32
==24577== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==24577== Replaying kernel "MatrixAddGlobalMem(float*, float*, float*, int, int, int)" (done)
Matrix Add is OK
==24577== Profiling application: ./MatrixAddGlobalMem 32
==24577== Profiling result:
==24577== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GTX 1070 (0)"
    Kernel: MatrixAddGlobalMem(float*, float*, float*, int, int, int)
          1                            gld_efficiency             Global Memory Load Efficiency      50.00%      50.00%      50.00%
```

(d) offset = 32



```
20184187@eelab5:~/gpu_programming/hw/hw2$ nvcc -Xptxas -dlcm=ca -arch=sm_61 -o MatrixAddGlobalMem MatrixAddGlobalMem.cu
20184187@eelab5:~/gpu_programming/hw/hw2$ nvprof --metrics gld_efficiency ./MatrixAddGlobalMem 128
==24659== NVPROF is profiling process 24659, command: ./MatrixAddGlobalMem 128
==24659== Some kernel(s) will be replayed on device 0 in order to collect all events/metrics.
==24659== Replaying kernel "MatrixAddGlobalMem(float*, float*, float*, int, int, int)" (done)
Matrix Add is OK
==24659== Profiling application: ./MatrixAddGlobalMem 128
==24659== Profiling result:
==24659== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "GeForce GTX 1070 (0)"
    Kernel: MatrixAddGlobalMem(float*, float*, float*, int, int, int)
          1                            gld_efficiency             Global Memory Load Efficiency      50.00%      50.00%      50.00%
```

(e) offset = 128

Figure 3. Global memory load efficiency

The global memory load efficiency is 50 % for all five cases.

In Pascal architecture, the data access unit is 32 bytes regardless of whether global loads are cached in L1 [1]. A cache line, 128 bytes long, is splitted into four 32-byte sectors. A memory transaction, 32 bytes long, actually requests 32-byte sector read from global memory. On other hand, there are two L1 caches in each Streaming Processor (SM). Hence:

$$\text{Requested Global Memory Load Throughput} = 2 \times 32 \text{ bytes} = 64 \text{ bytes}$$

However, one cache line contains data from one 128-byte memory line, so Required Global Memory Load Throughput equals to 128 bytes

$$\text{gld\_efficiency} = \frac{\text{Requested Global Memory Load Throughput}}{\text{Required Memory Load Throughput}} = \frac{64 \text{ bytes}}{128 \text{ bytes}} = 50\%$$

3. Homework 2.3

The source code of homework 2.3 is toUpperString.cu and the result is shown in Figure 3.1 below.

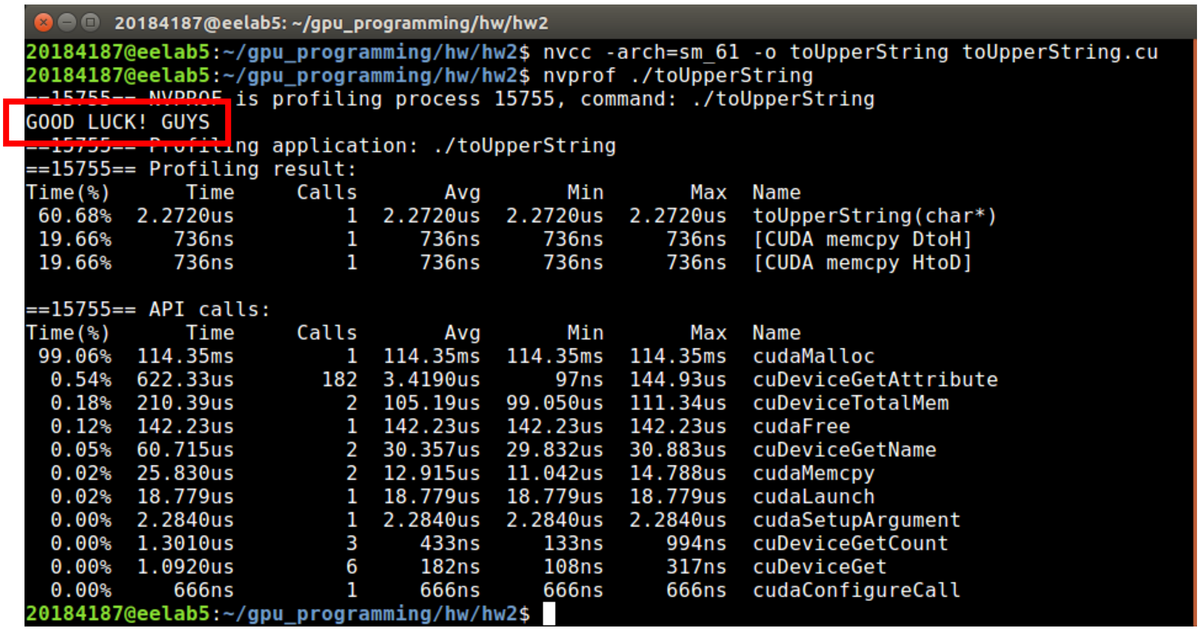


Figure 4. The result of homework 2.3

Explanation for 2 function d_islower() and d_toupper():

```
__device__ int d_islower(char c) {
  if (c > 96 && c < 123) return 1;
  else return 0;
}

__device__ int d_toupper(char c) {
  if (c > 64 && c < 91) return c;
  else return c - 32;
}
```

In ASCII table, the coding numbers of the lower Latin letters from “a” to “z”, is from 97 to 122. So, if 96 < c < 123, then c is lower.

As the lower case, the coding number of the upper Latin characters from “A” to “Z”, ranges from 65 to 90. Hence, if 64 < c < 91, then c is upper.

The ASCII number of the lower letter always greater than the corresponding upper letter by 32 units in decimal. Therefore, transforming lower letter to upper letter is minus 32.

Reference

[1] NVIDIA, Tuning CUDA Applications for Pascal, version 9.1, March 2018