Korea Advanced Institute of Science and Technology

School of Electrical Engineering

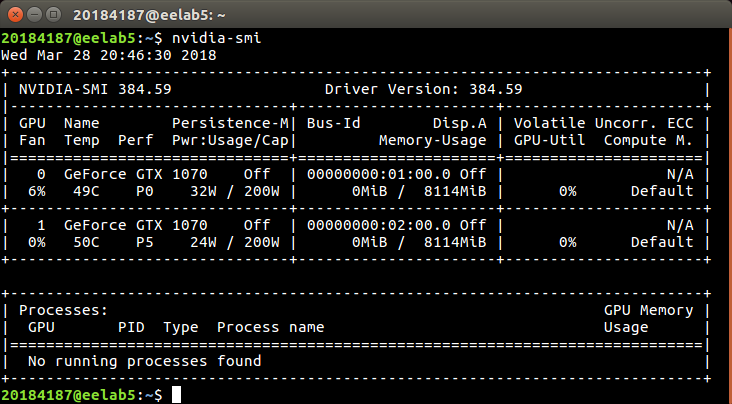
EE817 GPU Programming and Its Applications Spring 2018

Student: Dinh Vu

Student ID: 20184187

**Homework 2**

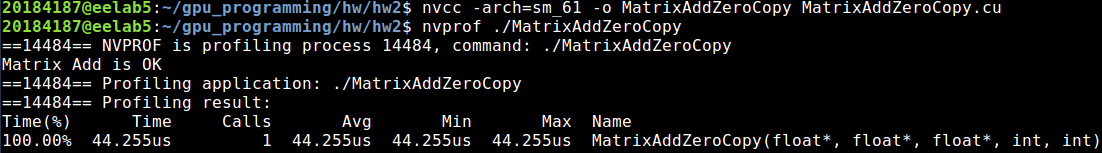
The computer used in this homework contains NVIDIA GeForce 1070 based on Pascal GP104 architecture.



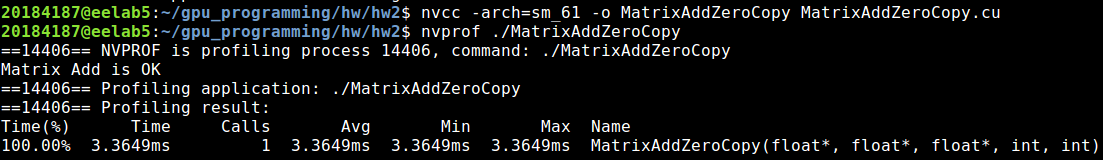
**Figure 1.** Graphic card information

1. **Homework 2.1**

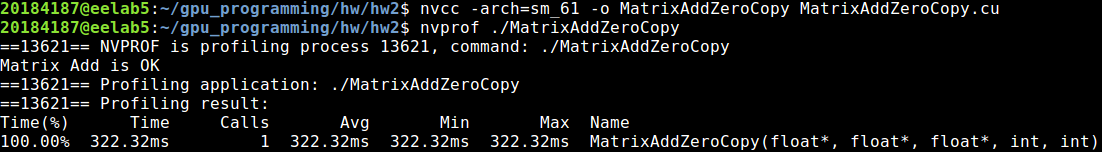
The source code of homework 2.1 is MatrixAddZeroCopy.cu. The execution time of matrix 160×160, 1600×1600 and 16000×16000 is presented in Figure 2 below.



1. Matrix 160×160



1. Matrix 1600×1600



1. 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 |

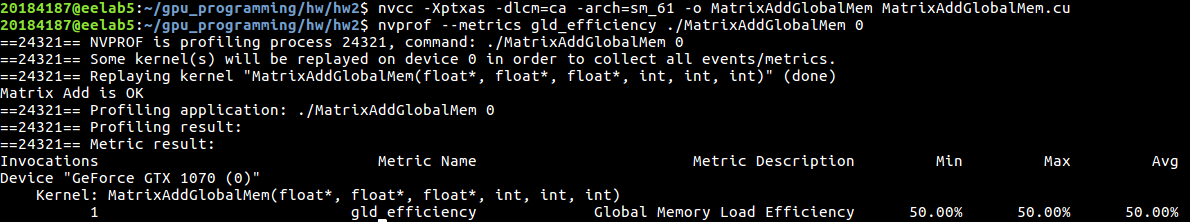
The practical result is corresponding with theorem about using zero-copy memory. Generally, the execution time increases by the size of matrix.

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.

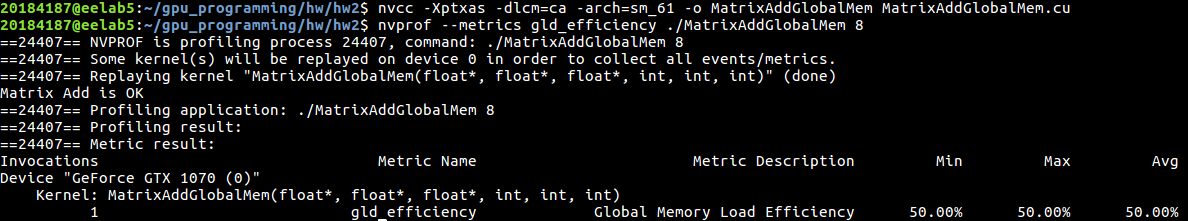
There are two reasons cause the highest execution time for matrix 16000×16000. First, the number of threads is greatest, 1.6×107 threads so it is need more time to schedule. Second, the most important reason is that reading and writing the data of 16 millions elements in each matrix, about 3×1.6×107×4 bytes ≈ 183 MB shared by CPU and GPU is limited by PCIe 3.0 bus with bandwidth 985 MB/s.

1. **Homework 2.2**

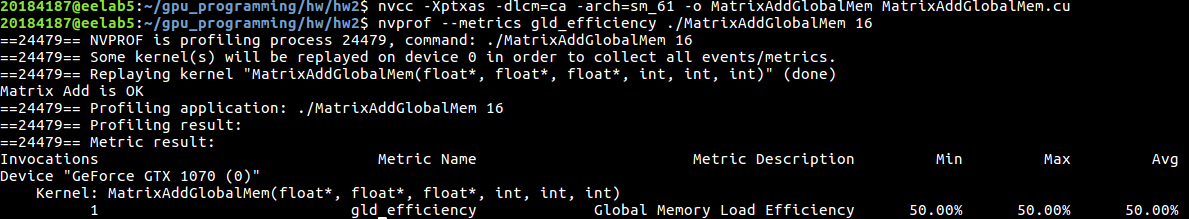
The source code for homework 2.2 is MatrixAddGlobalMem.cu. The result is display in Figure 2.1.



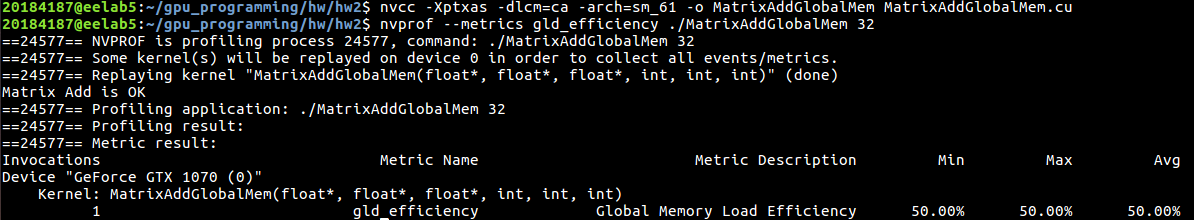
1. offset = 0



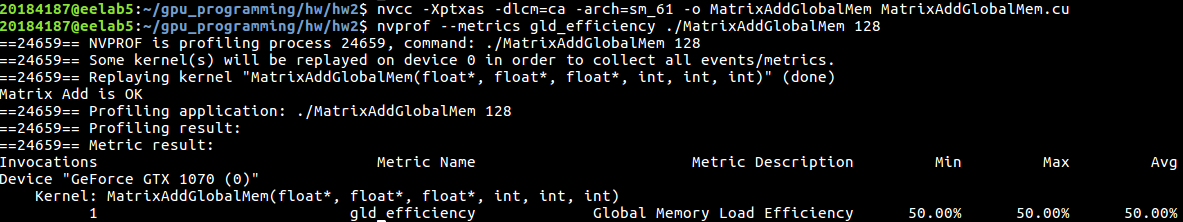
1. offset = 8



1. offset = 16



1. offset = 32



1. offset = 128

**Figure 3.** Global memory load efficiency

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

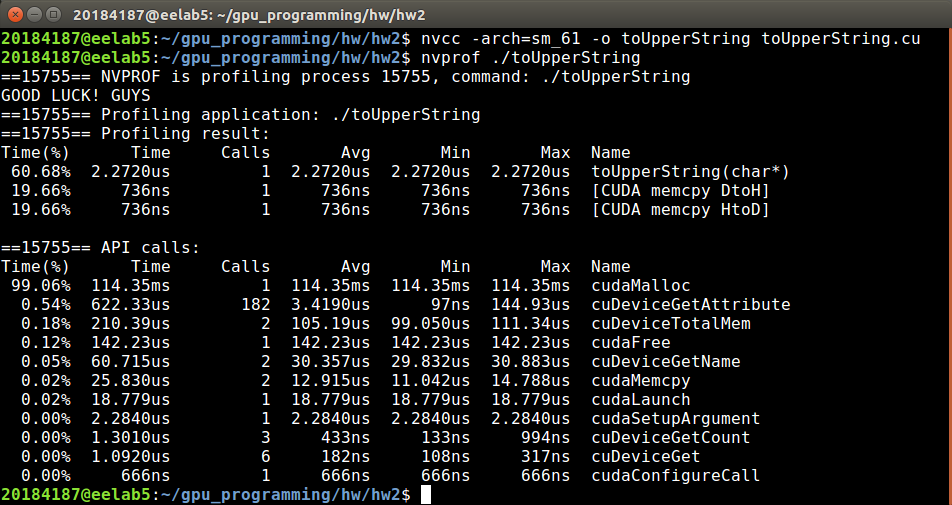
In Pascal architecture, the data access unit is 32 bytes regardless of whether global loads are cached in L1 [1]. A cache line, 128 bytes long, is splitted into four 32-byte sectors. A memory transaction, 32 bytes long, actually requests 32-byte sector read from global memory. On other hand, there are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

Requested Global Memory Load Throughput = 2 × 32 bytes = 64 bytes

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

1. **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