Korea Advanced Institute of Science and Technology

School of Electrical Engineering

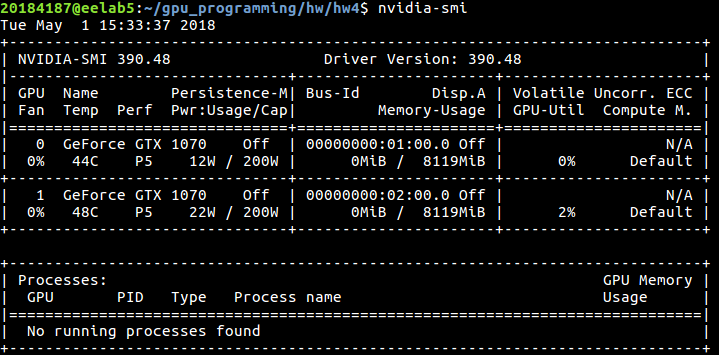
EE817 GPU Programming and Its Application Spring 2018

Student’s Name: Dinh Vu

Student’s ID: 20184187

**Homework 4**

The computer, used in my homework 4, contains NVIDIA GeForce GT 1070 based on Pascal GP104 architecture.



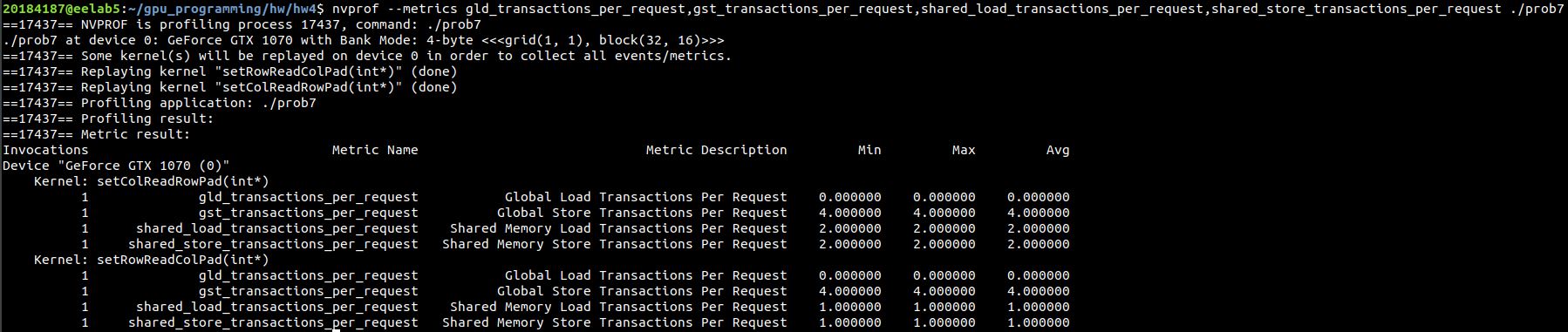
**Figure 1.** Graphic card information

1. **Problem 7**

The source code for problem 7 is prob7.cu file and the results are shown in Figure 1.1 and Table 1.1 below with thread block 32×16 using 4-byte access mode. There is not any load transaction from global memory, so Global Load Transactions Per Request equals to 0.

By padding IPAD = 2 elements in each row of the tile in the kernel setRowReadColPad, the even-column elements and the odd-column elements are distributed among even banks and odd banks, respectively. So, both writing by row-major and reading by column-major are conflict-free. It means the number of transactions for a request from shared memory is 1.00.

However, with the same padding, the kernel setColReadRowPad has a 2-way bank conflict in both reading and writing operation. The mapping from words to banks is illustrated at Figure 1.2 and Figure 1.3.



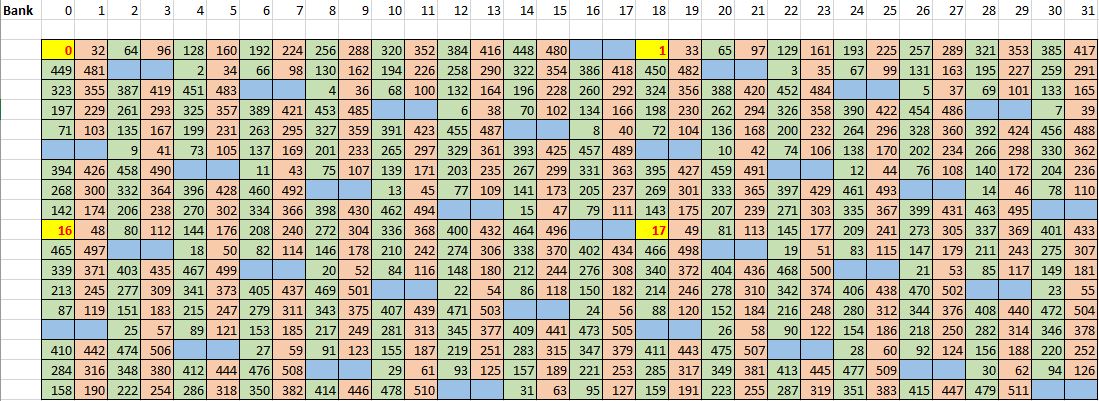
**Figure 1.1.** Results of problem 7

**Table 1.1.** Transaction Metrics

|  |  |  |
| --- | --- | --- |
| **Kernel** | setColReadRowPad | setRowReadColPad |
| **Global Load Transactions Per Request** | 0.00 | 0.00 |
| **Global Store Transactions Per Request** | 4.00 | 4.00 |
| **Shared Memory Load Transactions Per Request** | 2.00 | 1.00 |
| **Shared Memory Store Transactions Per Request** | 2.00 | 1.00 |



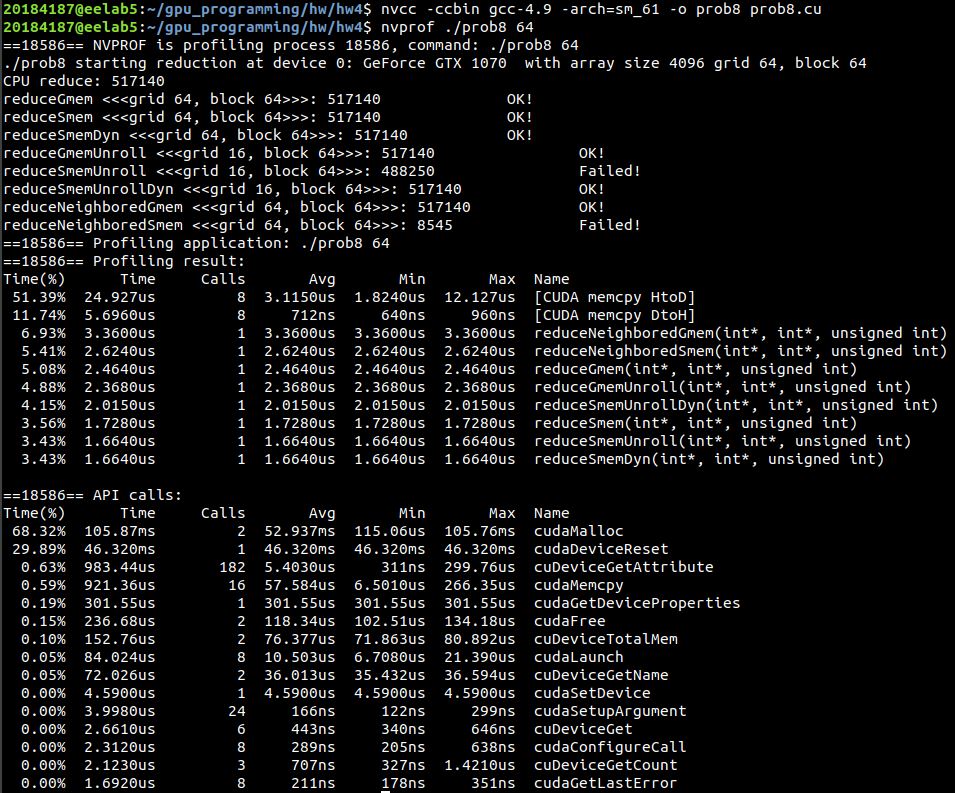
**Figure 1.2.** setRowReadColPad with bank conflict-free



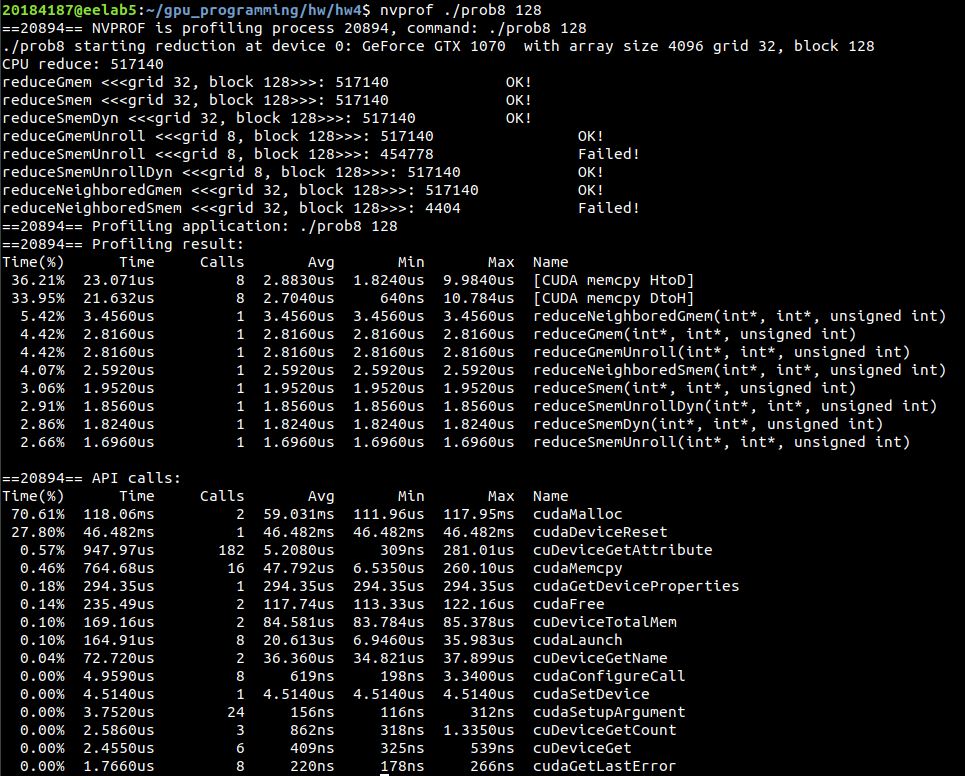
**Figure 1.3.** setColReadRowPad with bank conflict at yellow elements

1. **Problem 8**

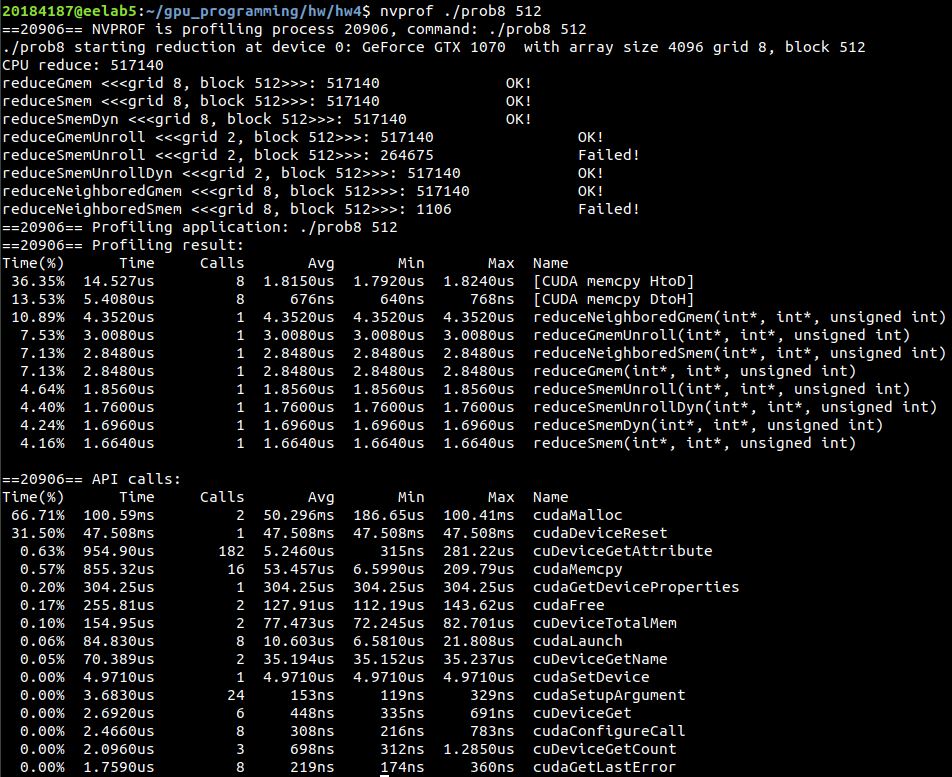
The source code for problem 8 is prob8.cu file and its results are display in Figure 2.1, Figure 2.2, Figure 2.3 and Figure 2.4 corresponding to block sizes of 64, 128, 512 and 1024. The summary of the results is typed in Table 2.1.



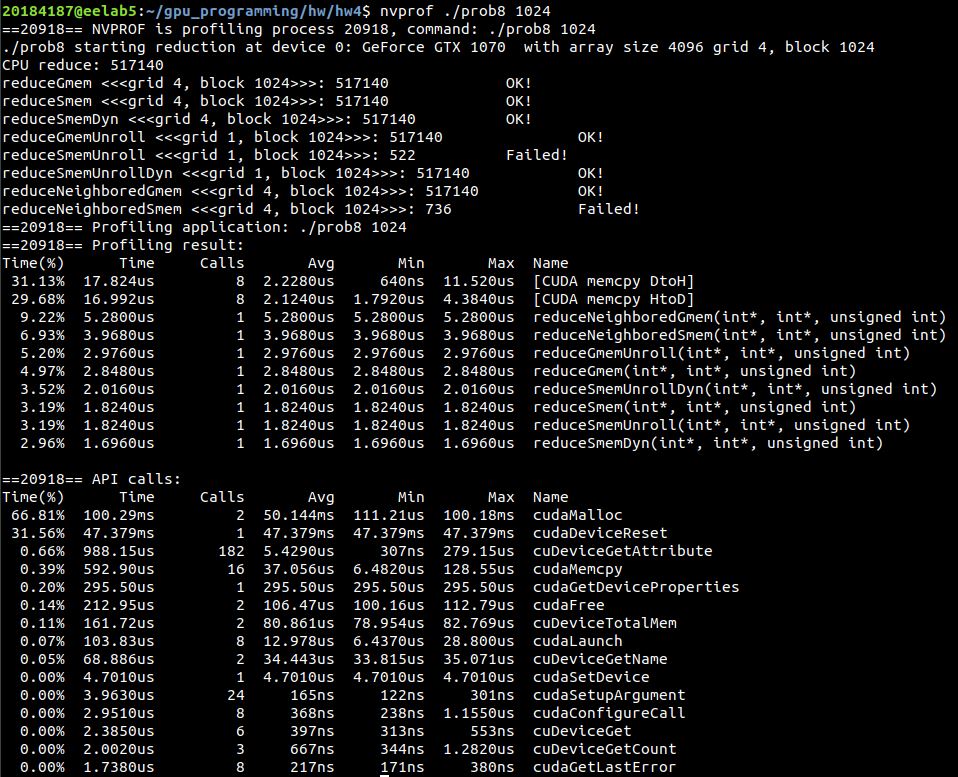
**Figure 2.1.** Test block size of 64



**Figure 2.2.** Test block size of 128



**Figure 2.3.** Test block size of 512



**Figure 2.4.** Test block size of 1024

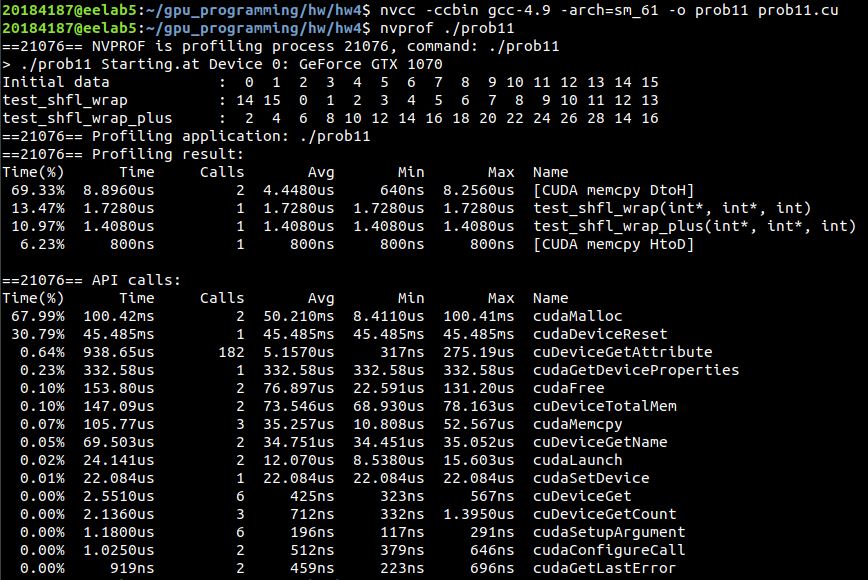
**Table 2.1.** Performance on various block sizes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Kernel** | **Execution time in µs** | | | |
| **64 threads** | **128 threads** | **512 threads** | **1024 threads** |
| reduceGmem | 2.4640 | 2.8160 | 2.8480 | 2.8480 |
| reduceSmem | 1.7280 | 1.9520 | 1.6640 | 1.8240 |
| reduceSmemDyn | 1.6640 | 1.8240 | 1.6960 | 1.6960 |
| reduceGmemUnroll | 2.3680 | 2.8160 | 3.0080 | 2.9760 |
| reduceSmemUnroll | 1.6640 | 1.6960 | 1.8560 | 1.8240 |
| reduceSmemUnrollDyn | 2.0150 | 1.8560 | 1.7600 | 2.0160 |
| reduceNeighboredGmem | 3.3600 | 3.4560 | 4.3520 | 5.2800 |
| reduceNeighboredSmem | 2.6240 | 2.5920 | 2.8480 | 3.9680 |

In general, with Pascal architecture, the best execution configuration is 64 threads per block, though reduceSmem, reduceSmemDyn and reduceSmemUnrollDyn achieve the best performance with 512 threads per block. The execution time with block size of 1024 is always highest.

1. **Problem 11**

For problem 11, the source code is prob11.cu file and the results, depicted in Figure 3.1 below. Based on the kernel test\_shfl\_wrap, the kernel test\_shfl\_wrap\_plus increases the current thread’s value by the value of the thread, which is two indexes greater. Because the width of the shuffle operation is still 16, so the first two initial threads warp around to the bottom two threads. The operation of the kernel test\_shfl\_wrap\_plus, presented in Figure 3.2.



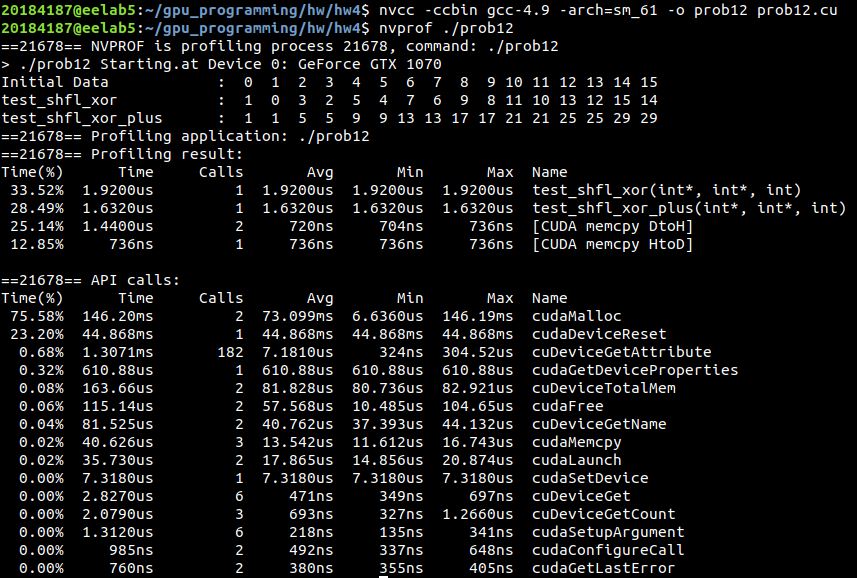
**Figure 3.1.** The results of problem 11



**Figure 3.2.** The operation of the kernel test\_shfl\_wrap\_plus

1. **Problem 12**

The prob12.cu file is the source code of problem 12 and its results are shown in Figure 4.1. Based on the kernel test\_shfl\_xor, the kernel named test\_shfl\_xor\_plus simply performs \_\_shfl\_xor function by passing one as the mask and increases the current thread’s value with the received value. Every even thread adds the value of the odd thread above it and every odd thread receives the value of the even thread below it.



**Figure 4.1.** The results of problem 12



**Figure 4.2.** The operation of the kernel test\_shfl\_xor\_plus