Assignment3 Report

Name:余永琦 ID:120090761

Note: I have finished the bonus

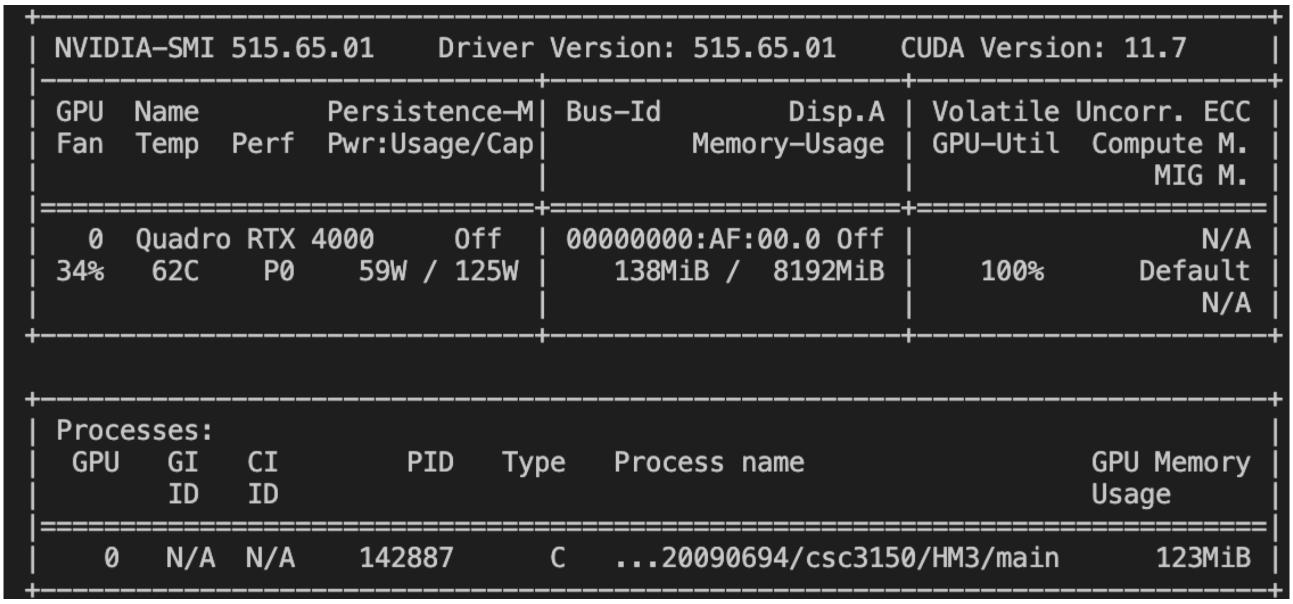
Environment

I complete and test my program on the GPU cluster provided by the course.

OS version: CentOS Linux release 7.5.1804 (Core)

CUDA version: CUDA SDK 11.7.20220729

GPU information:



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.65.01    Driver Version: 515.65.01    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Quadro RTX 4000     Off  | 00000000:AF:00.0 Off |                  N/A |
| 34%   62C    P0    59W / 125W |    138MiB /  8192MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    142887      C   ...20090694/csc3150/HM3/main      123MiB |
+-----------------------------------------------------------------------------+
```

Execution steps

One way:	“$ sbatch slurm.sh”
Another way:	“$ sh slurm.sh”

Note: For the sbatch running, you can check the result on the file result.out. For the sh running, it will output the result in terminal. Each execution of the main task takes about 1 minute to finish. **For bonus, you can only run on the sh way, since using the sbatch method will cause time out and couldn’t output the result correctly.** Each execution of bonus takes around 3 minutes.

How I designed my program?

In this assignment, we are asked to simulate a mechanism of virtual memory via GPU's memory.

We need to do CUDA programming to program the GPU.The CUDA programming is very similar to C, just need to add some keywords before we declare a variable or function to determine its location.

Now we go to the virtual memory simulation part. 48 KB (16KB for the inverted page table, and 32 for data) are allocated to the physical memory and 128KB are allocated to the disk storage.

First of all, we need to understand the paging concepts. Figure 1 shows how it works. For each program, it has a logical memory. We divide its memory into 32B blocks called pages, and store them in physical memory and disk. So, the physical memory need to be divided into blocks with same size, we call it frame. When the program needs to access data, it goes to the physical memory to find it. Hence, we need to have a table that maps the page number to the frame number, so that we can correctly access the data. In this assignment, an inverted page table is implemented to store the mapping information. Its index is the frame number, and we store a valid bit in it to indicate whether the page it store is valid, and a page number, as well as a counter for the LRU algorithm(explain later). The following shows how the inverted page table works: for a given page number, we traverse the whole table and find whether it is stored in the table. If so, we can return the index as our frame number. Otherwise, we will have a page fault.

Once the page fault happens, we need to handle it. We go to the disk memory to check if the page is stored in the disk. So, we also need a swap table to store the information. The implementation logic of my swap table is also inverted and is very similar to the page table, and I won’t explain it again. Just notice that for each term I add a number to store whether the current space is empty. Now we know how to check if the page is store in the disk. If still can not find the page, then we will just allocate a block for this page number. Then, if there is some invalid block on the physical space, we directly swap the memory block from disk to the physical memory. Otherwise, we need to first find a victim frame and swap it out to the disk, and then we swap the memory in the disk into the physical space. Figure 2 shows the swapping procedure. Since we need to first swap out, there must always be an empty block in the disk.

To find a victim frame, we need the Least Recent Used(LRU) algorithm. That is, the victim frame should be the one that is used least. In my implementation, I used counters in the inverted page table for each frame to indicate the use frequency, and all are initialized as 0. Each time we access a frame, we set the accessed frame counter as 0 while all other frame's counter should plus 1. Then, when we need a victim frame, we give a frame with the largest counter number.

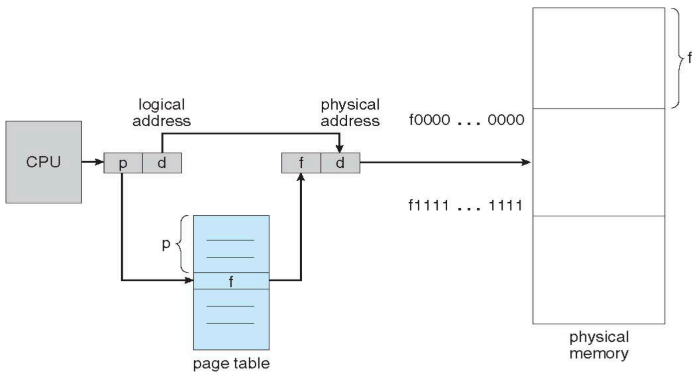


Figure 1

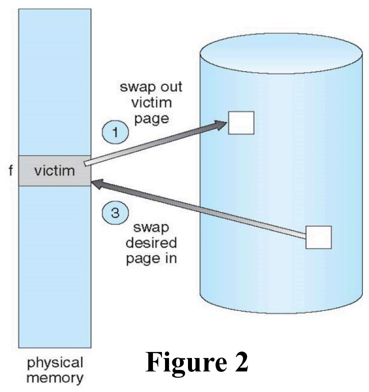


Figure 2

Bonus

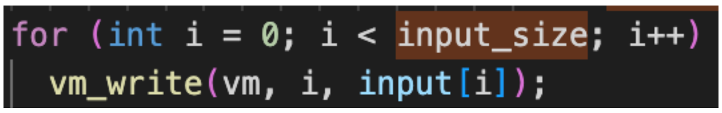
In bonus, I implemented the version 2 in the description. That is, we launch 4 threads and all the threads perform the same and so the user program is processed 4 times. To launch 4 threads we need to change "mykernel<<<1, 1, INVERT_PAGE_TABLE_SIZE>>>(input_size);" into "mykernel<<<1, 4, INVERT_PAGE_TABLE_SIZE>>>(input_size);". Then we need to use a method "__syncthreads()" to synchronize the threads. When the method is encountered in the kernel, all threads in a block will be blocked at the calling location until each of them reaches the location. This method should be called after each thread so that all the threads can be synchronized. The following website contains more details of this method: https://www.tutorialspoint.com/cuda/cuda_threads.htm .

This is the whole design of my program, for the implementation details, please read my code and check the comments.

Page fault number

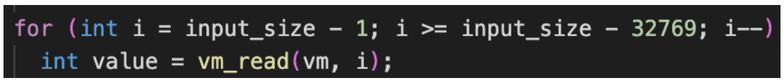
Testcase1

The page fault number of test case 1 is 8193.



```
for (int i = 0; i < input_size; i++)
  vm_write(vm, i, input[i]);
```

This contribute 4096 page fault number, because it need to write 4096 pages in total, and the table contains no page at the beginning.



```
for (int i = input_size - 1; i >= input_size - 32769; i--)
  int value = vm_read(vm, i);
```

This contribute 1 page fault number, because it contains page number from 4096 to 3071. Since the pages 4096 to page 3072 are in the physical memory now, so this only contribute 1.

Finally the vm_snapshot call reads pages form 1 to 4096 also contribute 4096 page fault number, hence the total is 4096+1+4096=8193.

Testcase2

The page fault number of test case 2 is 9215. It is very easy to calculate. The first vm_wirte contributes 4096, the second vm_wirte contributes 1023 and the vm_snapshot contributes 4096, hence in total 4096+1023+4096=9215.

Bonus

Testcase1

The page fault number is 8193 * 4 = 32772, since the four threads perform the whole process four times.

Testcase2

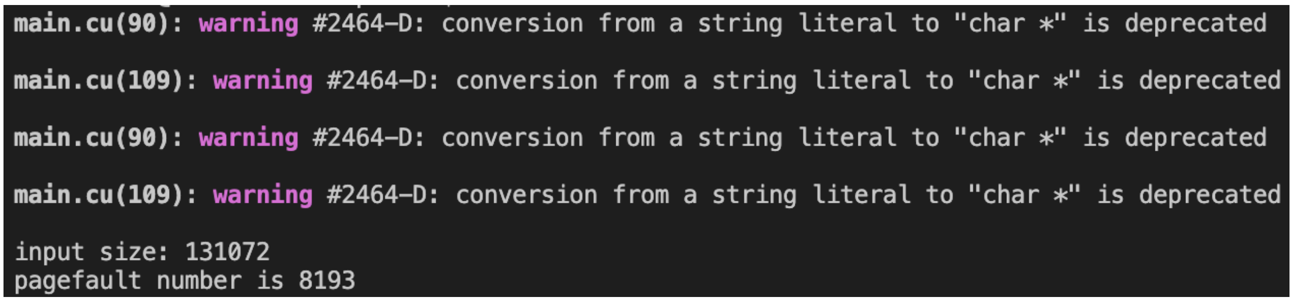
The page fault number is 9215 * 4 = 36860, since the four threads perform the whole process four times.

Problems I met in this assignment

The main problem to be solved is to map the page to the disk. According to the tutorial, I used a swap table and designed it as an inverted swap table to solve the mapping problem. Also, we need to think about how to implement the LRU algorithm, and I used counter to implement it.

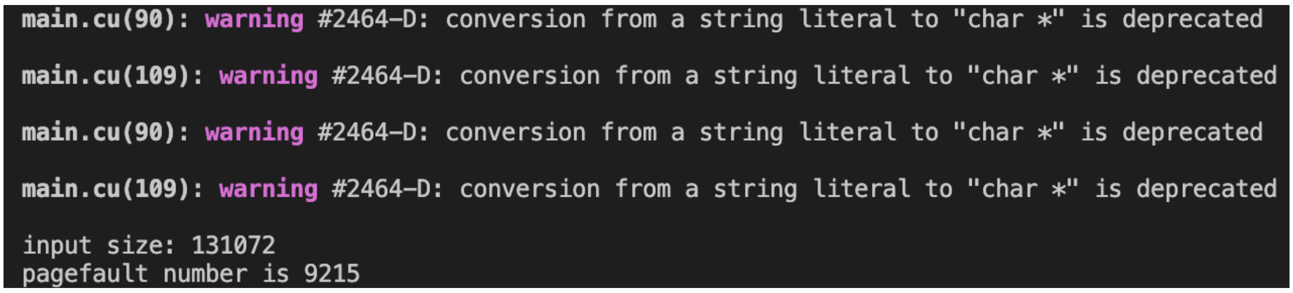
Screenshot

Testcase1



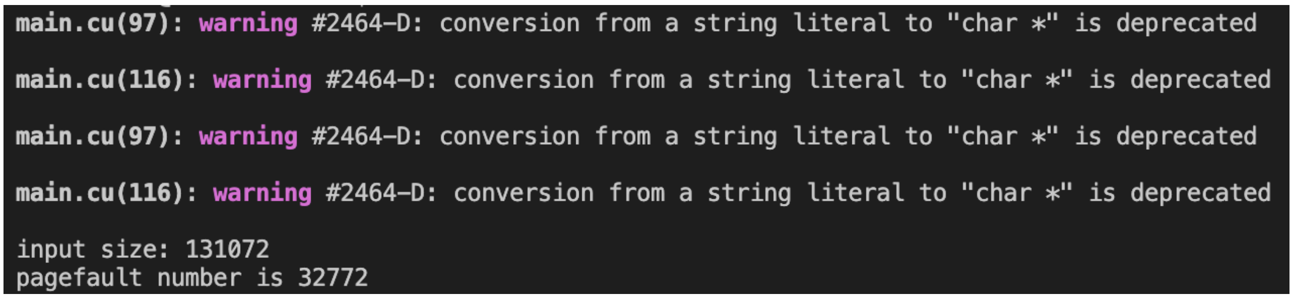
```
main.cu(90): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(109): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(90): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(109): warning #2464-D: conversion from a string literal to "char *" is deprecated

input size: 131072
pagefault number is 8193
```

Testcase2



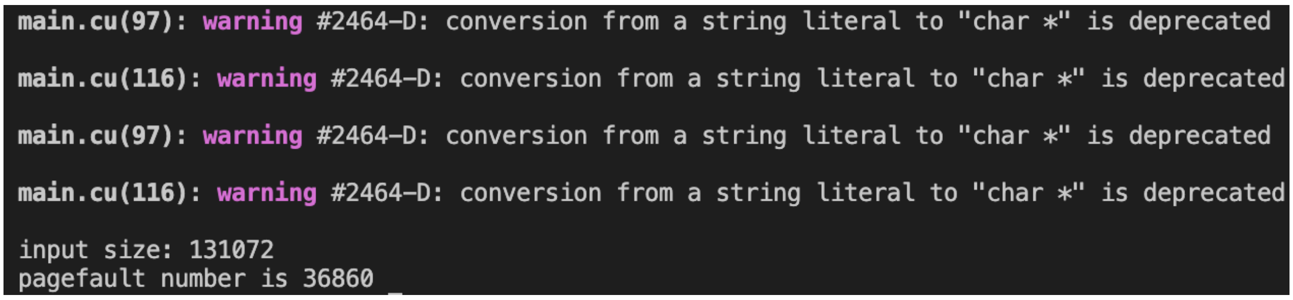
```
main.cu(90): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(109): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(90): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(109): warning #2464-D: conversion from a string literal to "char *" is deprecated

input size: 131072
pagefault number is 9215
```

Bonus
Testcase1



```
main.cu(97): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(116): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(97): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(116): warning #2464-D: conversion from a string literal to "char *" is deprecated

input size: 131072
pagefault number is 32772
```

Testcase2



```
main.cu(97): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(116): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(97): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(116): warning #2464-D: conversion from a string literal to "char *" is deprecated

input size: 131072
pagefault number is 36860
```

What did I learn?

In this assignment, I learned the basic knowledge of CUDA programming. Also, I got familiar with the mechanism of virtual memory. I knew how paging works, how it maps to physical memory, and how to handle it when we can not find pages in the physical space. Besides, I learned the LRU algorithm to find a victim frame to swap out, and mapping the page to the disk using swap table as well.

In bonus, I learned the knowledge about the multithreads programming in CUDA, including how the launch multiple threads as well as how to schedule them.