

Assignment3

Name: Xinyu Xie

Student ID: 119020059

Virtual Memory Management

Environment of running my program

Operating system: Windows 11

IDE: Visual Studio 2022

CUDA Version: 11.8

GPU Information: Nvidia GeForce GTX 1050

HPC environment

Operating system: CENTOS-7

CUDA version: 11.7

GPU information: Nvidia Quadro RTX 4000

Steps to execute my program

The following are the steps to execute the program:

1. Go to root directory
2. Run sbatch ./slurm.sh

How did I design my program

From the virtual memory specification given, we can calculate that page offset needs 5 bits, physical page number needs 10 bits, and virtual page number needs 13 bits.

Page Table

Each entry in the page table has 32 bits, which contains virtual page number (first 13 bits), valid bit (last bit) with 1 denoting valid and 0 denoting invalid, physical page number (second last 10 bits).

The page table entries forms a linked list, the head is MRU entry and the last is LRU entry. When a page is read or write, the corresponding entry will be moved to the head of linked list.

vm_write

1. Firstly, given a virtual page number, this function goes through the inverted page table to check whether there exists an entry matching the required virtual page number.
2. If the corresponding entry is found, then we have page hit, then the iteration will terminate and write data into the corresponding physical page number directly.
3. If we have page fault, the iteration will go through all the elements to find empty page. If there is an

empty page, the program will directly swap the corresponding page in the secondary storage to the frame in physical memory. Otherwise, it will first use LRU to find the recently least used page and swap that page back to secondary memory. After that, the program would swap the corresponding frame in the physical memory. Then the function can directly write the data into the new page. And update the address of that page in the page table. After writing back the data, the function would set the LRU node of the corresponding element to the front of the LRU list. And also update the head and tail information in the page table.

vm_read

1. Firstly, given a virtual page number, this function goes through the inverted page table to check whether there exists an entry matching the required virtual page number.
2. If the corresponding entry is found, then we have page hit, then the iteration will terminate and read from corresponding physical page number directly.
3. If we have page fault, the iteration will go through all the elements to find empty page. If there is an empty page, the program will throw an error. Otherwise, the program need to swap out the data in main memory to disk, then swap in the page residing in the disk into the main memory, and perform read operation. And also update the head and tail information in the page table.

Page fault number of my output

Program 1

The total page fault number is 8193. The page fault number is 4096 for write operation, 1 for read operation, and 4096 for snapshot operation.

The 4096 page faults for write operation is because each time start write a new page of data, a page fault occurs. Since there are 128K data with 32 bytes pagesize in each page, by calculation we have 4096 page faults because there are 4096 pages.

The 1 page fault caused by read operation is because the amount data read is 1 byte more than the capacity of the page table, with the first 32768 byte to read being the last 32768 byte written into the main memory. Therefore, the extra 1 byte cause 1 page fault.

The last 4096 page faults is because of snapshot operation. The program reads all data from the lowest address of virtual memory to the highest address of virtual memory. Each page will generate one page fault.

Program 2

Using similar method mentioned above, the total page fault number is $4096 + 1023 + 4096 = 9215$.

Problems I met

I met the problem of how to design page table with efficient space usage. I figured out the solution of packing multiple fields into one int to largely save spaces, and extract each field using bit-wise operations when necessary.

Screenshot of my program output

User Program 1 (Test Case 1)

```
input size: 131072
pagefault number is 8193
```

User Program 2 (Test Case 2)

```
input size: 131072
pagefault number is 9215
```

What did I learn from the tasks

In this task, I learned the following knowledge:

1. I get better understanding of memory management through implementing virtual memory, including page table, LRU, page replacement policy.
2. I learned some basic knowledge about cuda programming like declaring variables and functions in different positions using *device* and *host*.
3. I also recall some knowledge of the data structure through Investigating different data structures when implementing the LRU.

Bonus

Environment of running my program

Operating system: Windows 11

IDE: Visual Studio 2022

CUDA Version: 11.8

GPU Information: Nvidia GeForce GTX 1050

HPC environment

Operating system: CENTOS-7

CUDA version: 11.7

GPU information: Nvidia Quadro RTX 4000

Steps to execute my program

The following are the steps to execute the program:

1. Go to root directory
2. Run sbatch ./slurm.sh

How did I design my program

The 4 threads will run sequentially, with order thread 0, thread 1, thread 2, thread 3. I use `__syncthreads` to set up thread barrier and synchronize all threads, which ensures their running order.

I use `thd_id` to denote the current running thread. When a new thread starts running, the function will firstly clear everything related to page table and swap table, after that performs read and write operations as shown in the main task.

Page fault number of my output

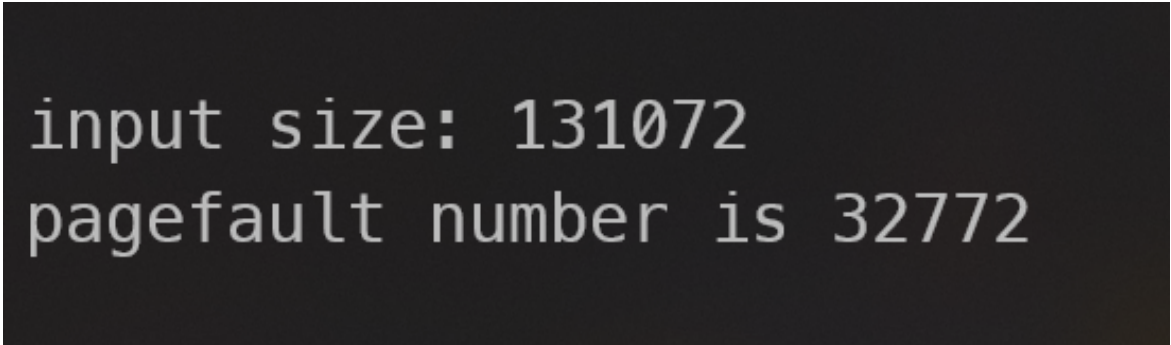
The page fault number = $8193 * 4 = 32772$.

Since the thread clears all data written by other threads, for 4 threads in total, the total page fault number should be four times of single thread page fault number.

Problems I met

I have problems about how to schedule CUDA threads and do synchronization. And after searching online and learning from Nvidia Guide, I find the function `__syncthreads` which could be used to do threads synchronization.

Screenshot of my program output



```
input size: 131072
pagefault number is 32772
```

What did I learn from the tasks

Besides the things I learned in the main task related to virtual memory management and CUDA programming, I also learned different ways to design page table for managing multiple threads with only one page table instead of associated page tables.