Report

Zhenyu Pan 120090196

November 9, 2022

1 Environment

Since we are recommended to use the cluster to compile and test our codes, I use the cluster provided, thanks for Prof.Chung and TAs bringing the cluster into use. Below is the environment information provided in "Cluster User Guide".

Item	Configuration / Version
System Type	x86_64
Opearing System	CentOS Linux release 7.5.1804
CPU	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz 20 Cores, 40 Threads
Memory	100GB RAM
GPU	Nvidia Quadro RTX 4000 GPU x 1
CUDA	11.7
GCC	Red Hat 7.3.1-5
CMake	3.14.1

Figure 1: Environment Information

2 Execution Steps

1. Please make sure the script and the corresponding files are in the **same folder**
2. Log in to cluser and enter the folder(eg: cd source/bonus)
3. Type "sbatch ./slurm.sh" in the terminal
4. Wait a few seconds, the task will be done

3 Design of Program

For **vm_write**, first use *parse_addr* to parse out the *page_id* and *page_offset* based on the addr-parameter passed in. Then, search and judge whether the parsed *page_id* is in the *invert_page_table*; 1. if it is, we need to judge the most significant bit of *control bit* is valid or invalid: if **invalid**, meaning that the corresponding data is not loaded into the memory. Therefore, we need to read the corresponding *page_id* data from the storage, load it into the corresponding *frame*, and mark the bit valid, then write the data to the corresponding address. Update the LRU in the end; if **valid**, meaning that the corresponding data is loaded into the memory, we can directly write the data to the corresponding address, and update the LRU in the end. 2. if it isn't, then we need to use the **LRU strategy**: Find the frame that has not been used for the most recent time, if the frame is valid, flush the data to the original address of *page_id* corresponding to the frame that needs to be removed on the storage, then read the data of *page_id* we need to write from the storage to this frame. Finally modify the mapping between frame and *page_id* in *invert_page_table*, set the flag bit to valid, then write the data to the corresponding address. Don't forget to update the LRU in the end. For the LRU implementation, I use array to simulate it, since generally speaking, LRU is to find an item that has not been used for the most recent time, so we can maintain a timestamp for each data item. I define a global variable *timestamp* that is incremented by one after each update operation, and then assign the corresponding array entry to the value of this timestamp. When need remove, I find the entry with the smallest timestamp, indicating that it has not been accessed recently. For **vm_read**, it is similar to "vm_write" we have discussed above, just regard "write" above as "read" is easy to understand.

4 Page Fault Output & Brief Analysis

For basic part test1, the page fault is **8193** as expected. To verify the number of page fault, we have to check *user_program.cu* and it is consists of three processes: write, read and snapshot. I. For the write iteration, since the input size is 131072, and every time when the program

writes the page that is mod(32) remaining 0, there would be a page fault raise, so totally 131072/32 = 4096 page faults will raise. II. For the read iteration, since the iteration time is 32769 = 32(page_size)*1024+1, which means for the first 1024 iteration, there always be a hit because the reading process starts from the address is just written, and for the last process, there actually will be a page fault since the page is not in the page table. III. For the snapshot part, obviously it gives 4*1024 = 4096 page faults since it transfers the data from address 0. Therefore, in total 4096+1+4096 = 8193 page faults will be raised.

For basic part test2, the page fault is **9215** as expected. To verify the number of page fault, the process is similar as discussed above. The first process and third process is actually the "same", the only difference is that start address from 0 to 32*1024, so the first and third process also both raise 4096 page faults. For the third process, which writes (32KB-32B) data to the VM starting from 0, similar to process 1, since the input size is 32*1023, the total page faults are 32*1023/32 = 1023. Therefore, in total 4096+1023+4096 = 9215 page faults will be raised.

For the bonus Part, I choose to implement version 1, so the page fault is as the same as the test1, is **8193**.

5 Problem Met in the Process

The most tricky problem occurs when the test 2 released. After searching for a long time, I feel that there is no problem with the logic, but the number of replacement(the number of page fault) is not correct. After all, our homework is to check the number of page fault. In the end, I find it out. It turns out that I was writing in the traditional way, for example, if a page id is not there, I need to replace and remove a frame, my previous idea was to check if there were any invalid frames, and if there were any invalid frames, I would remove the invalid frames first. If all invalid then execute LRU. But this project should not ignore "invalid", so it never got it right.

6 Bonus

I choose Version 1 to implement the bonus part. Actually, the general structure of virtual memory in bonus is similar to the basic part, so we just need to launch 4 CUDA threads to realize it as mentioned in the document. Therefore, the kernel part in the "main.cu" should be modified to support 4 threads, the modification is shown below. Also, to control the threads, we need to use function *_syncthreads()* and *threadIdx.x* to get their thread ID(add them to the function vm_wirte and snapthot, no vm_read since it has return value). For more details, please check comments of the code in bonus file.

```
91  /* Launch kernel function in GPU, with single thread
92  and dynamically allocate INVERT_PAGE_TABLE_SIZE bytes of share memory,
93  which is used for variables declared as "extern __shared__" */
94  mykernel<<<1, 4, INVERT_PAGE_TABLE_SIZE>>>(input_size);
```

Figure 2: replace "1" with "4"

7 Screenshot of output

```
source > result.out
1  main.cu(89): warning #2464-D: conversion f
2
3  main.cu(108): warning #2464-D: conversion
4
5  main.cu(89): warning #2464-D: conversion f
6
7  main.cu(108): warning #2464-D: conversion
8
9  input size: 131072
10 pagefault number is 8193
11
```

Figure 3: Test 1 Result

Figure 4: Test 2 Result

Please not that if there is no information after the command "cmp file1 file2" indicates the contents of the two files are totally same(the below compare for bonus part is the same sense)

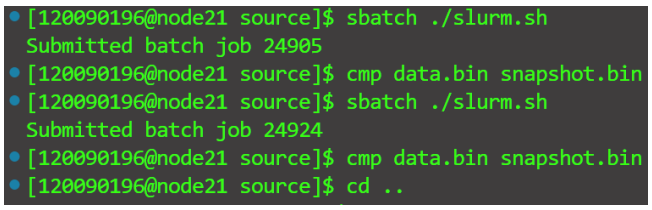

Figure 5: Test 1 & 2 Compared with data.bin

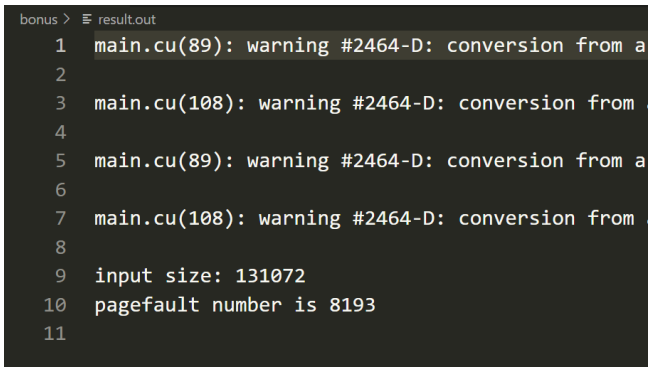

Figure 6: Bonus Result

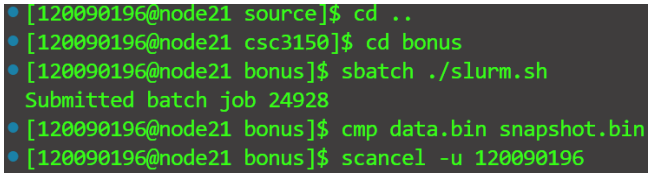

Figure 7: Bonus Compared with data.bin

8 What I learn

1. Self study is important
2. Start as early as possible
3. GDB is a good thing to debug
4. Better understanding on Memory Management
5. Learn basic CUDA language concepts
6. Understanding is more important than blind coding
7. Don't give up even the deadline time is coming