CSC3150 Project 3 Report

Jiakun Fan

120090316

1. Program Environment

- Operating system: CentOS Linux release 7.5.1804
- CUDA version: 11.7
- GPU: Nvidia Quadro RTX 4000 GPU x 1

2. Execution Steps

Before executing, make sure your device has **CUDA**.

Source

Under `/source` directory:

```
nvcc --relocatable-device-code=true main.cu user_program.cu virtual_memory.cu -o
test
srun ./test #see the output
```

Bonus

Under `/bonus` directory:

```
nvcc --relocatable-device-code=true main.cu user_program.cu virtual_memory.cu -o
test
srun ./test #see the output
```

3. Program Design

source

```
void init_invert_page_table(VirtualMemory *vm);
```

Initialize the page table, with all entries set to invalid

```
void vm_init(VirtualMemory *vm, uchar *buffer, uchar *storage);
```

Initialize the virtual memory, as well as the stack storing the information of least recent used page, implemented by an array.

```
int vm_get_LRU_idx(VirtualMemory *vm);
```

Obtain the least recent used index (main memory address) of the LRU page,

extracting this information from the Page Number array. The least recent used

index is the index with the min time counter.

```
int vm_search_vpn(VirtualMemory *vm, int vpn);
```

Iterate through the page table to find the index of a page number. If found,

then this page is available, and page fault if not found.

```
int vm_search_swap_table(VirtualMemory *vm, int vpn);
```

Iterate through the swap page table to find the index of a page number. If found,

return the index, else return -1.

```
void vm_update_queue(VirtualMemory *vm, int idx);
```

Each time the program accesses a page, the corresponding index in array should be assigned the current time counter value to indicate that it is the most recent used one and the time counter add one to itself.

Therefore, this function updates the page number array to keep the lru index.

```
void vm_write(VirtualMemory *vm, u32 addr, uchar value);
```

Write the data into the main memory by locating the correct main memory address via page table. If the demanding page number is in the page table, then extract the corresponding main memory address (index of the inverted page table) as well as write data to this location of main memory. If not, then page fault. In this case 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 write operation. At the same time, two page tables and page number queue need to be updated.

```
uchar vm_read(VirtualMemory *vm, u32 addr);
```

Read the data from the main memory by locating the correct main memory address via page table. If the demanding page number is in the page table, then extract the corresponding main memory address (index of the inverted page table) as well as read data from this location of main memory. If not, then page fault. In this case 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. At the same time, two page tables and page number queue need to be updated.

```
void vm_snapshot(**struct** VirtualMemory *vm, uchar *results, int offset, int input_size);
```

Basically extract all the information of the input binary file.

Bonus

bonus is based on source, I implement version 3.

- Add the thread index with vpn when updating two page table and search with thread index.
- the content of `vm_read` and `vm_write` are transferred to `vm_atomic_read` and `vm_atomic_write` .

  And `vm_read` and `vm_write` are used to schedule 4 threads.

```
uchar vm_read(VirtualMemory *vm, u32 addr) {
  int val;
  if (threadIdx.x == 0) {
    val = vm_atomic_read(vm, addr);
  }
  __syncthreads();
  if (threadIdx.x == 1) {
    val = vm_atomic_read(vm, addr);
  }
  __syncthreads();
  if (threadIdx.x == 2) {
    val = vm_atomic_read(vm, addr);
  }
  __syncthreads();
  if (threadIdx.x == 3) {
    val = vm_atomic_read(vm, addr);
  }
  __syncthreads();

  return val;
}

void vm_write(VirtualMemory *vm, u32 addr, uchar value) {
  if (threadIdx.x == 0) {
      vm_atomic_write(vm, addr, value);
    }
    __syncthreads();
    if (threadIdx.x == 1) {
      vm_atomic_write(vm, addr, value);
    }
    __syncthreads();
    if (threadIdx.x == 2) {
      vm_atomic_write(vm, addr, value);
    }
    __syncthreads();
    if (threadIdx.x == 3) {
      vm_atomic_write(vm, addr, value);
    }
    __syncthreads();

}
```

4. Page Fault Number

`user_program.cu`

```
void user_program(VirtualMemory *vm, uchar *input, uchar *results,
                              int input_size) {
  for (int i = 0; i < input_size; i++){
    vm_write(vm, i, input[i]);
  }

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

  vm_snapshot(vm, results, 0, input_size);
}
```

source

- test 1 8193 page faults

```
void user_program(VirtualMemory *vm, uchar *input, uchar *results,
                              int input_size) {
  for (int i = 0; i < input_size; i++){
    vm_write(vm, i, input[i]);
  }
  // writing 0-4095 entries occurs 4096 faults
  // page table contains 3071-4095
  for (int i = input_size - 1; i >= input_size - 32769; i--){
    int value = vm_read(vm, i);
  }
  // read 3070-4095 entries occurs 1 faults (3070 not in page table)

  vm_snapshot(vm, results, 0, input_size);
  // read 0-4095 entries occurs 4096 faults

  // total number = 4096 + 1 + 4096 = 8193
}
```

- test 2 9215 page faults

```
void user_program(VirtualMemory *vm, uchar *input, uchar *results,
  int input_size) {
    // write the data.bin to the VM starting from address 32*1024
    //4096
    for (int i = 0; i < input_size; i++)
      vm_write(vm, 32*1024+i, input[i]);
    // write (32KB-32B) data  to the VM starting from 0
    // 1023
    for (int i = 0; i < 32*1023; i++)
      vm_write(vm, i, input[i+32*1024]);
```

```
    // readout VM[32K, 160K] and output to snapshot.bin, which should be the same
with data.bin
    //4096
    vm_snapshot(vm, results, 32*1024, input_size);

}

//expected page fault num: 4096*2+1024 = 9215
```

bonus

Four threads perform the same operation four times, but the disk is scaled to 512KB so that the four threads can write on different pages of the virtual machine without overwriting each other. When the final snapshot is made, the result buffer stores the 128KB written by the last thread.

The execute sequence: 0->1->2->3

35844 page faults

```
void user_program(VirtualMemory *vm, uchar *input, uchar *results,
                            int input_size) {
  for (int i = 0; i < input_size; i++){
    vm_write(vm, i, input[i]);
  }
  // writing 0-4095 entries 4 times occurs 4*4096 faults
  // page table contains 3839-4095(because the dram is used by four thread, so
every
  // thread only have 8KB(512 entries) dram to use)
  for (int i = input_size - 1; i >= input_size - 32769; i--){
    int value = vm_read(vm, i);
  }
  // read 3070-4095 entries 4 times occurs 4*769 faults (3070-3838 (1024 - 256 +
1) not in page table)

  vm_snapshot(vm, results, 0, input_size);
  // read 0-4095 entries occurs 4 * 4096 faults

  // total number = 4 * 4096 + 4 * 769 + 4 * 4096 = 35844
}
```

- version 3 bonus don't support test 2. Because for every single thread, it only supports 4096 + 256(1024/4) entries. In test case 2, 1023 + 4096 entries are written which result in no space.

5. Problem Encountered

It's hard to debug in **CUDA**. I use lots of printf and valgrind to debug.

6. Screenshots

- source

```
[120090316@node21 source]$ ls
data.bin  main.cu  slurm.sh  snapshot.bin  test  user_program.cu  virtual_memory.cu  virtual_memory.h
[120090316@node21 source]$ nvcc --relocatable-device-code=true main.cu user_program.cu virtual_memory.cu -o test
main.cu(93): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(112): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(93): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(112): warning #2464-D: conversion from a string literal to "char *" is deprecated

[120090316@node21 source]$ srun ./test
input size: 131072
pagefault number is 8193
[120090316@node21 source]$
```

```
[120090316@node21 source]$ nvcc --relocatable-device-code=true main.cu user_program.cu virtual_memory.cu -o test
main.cu(93): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(112): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(93): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(112): warning #2464-D: conversion from a string literal to "char *" is deprecated

[120090316@node21 source]$ srun ./test
input size: 131072
pagefault number is 9215
[120090316@node21 source]$
```

- bonus

```
[120090316@node21 bonus]$ ls
data.bin  main.cu  slurm.sh  snapshot.bin  test  user_program.cu  virtual_memory.cu  virtual_memory.h
[120090316@node21 bonus]$ nvcc --relocatable-device-code=true main.cu user_program.cu virtual_memory.cu -o test
main.cu(93): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(112): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(93): warning #2464-D: conversion from a string literal to "char *" is deprecated

main.cu(112): warning #2464-D: conversion from a string literal to "char *" is deprecated

[120090316@node21 bonus]$ srun ./test
input size: 131072
pagefault number is 35844
[120090316@node21 bonus]$
```

7. Things learned

I learned the working principle of memory.