CSC3150 Assignment 3

In Assignment 3, you are required to simulate a mechanism of virtual memory via GPU's memory.

Background:

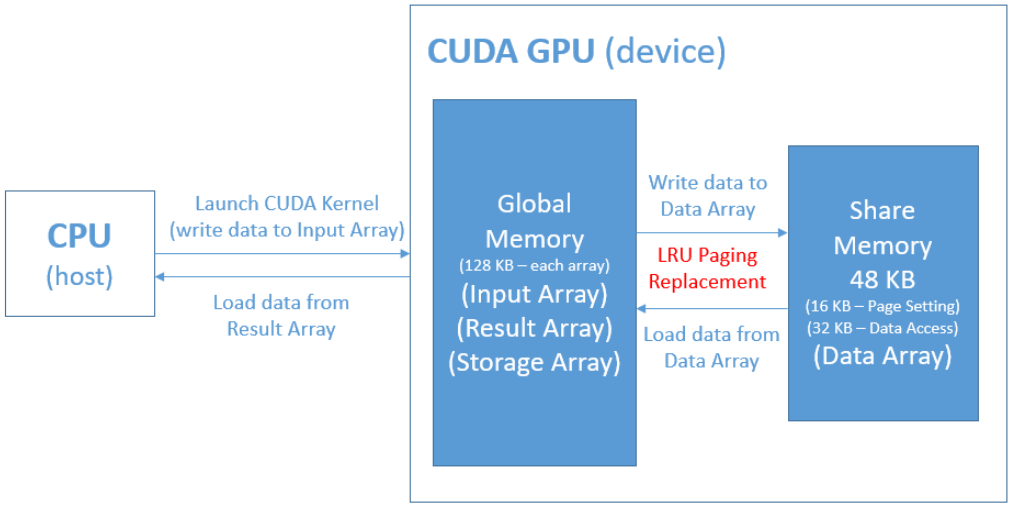
- Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory.
- In this project, you should implement simple virtual memory in a kernel function of GPU that have single thread, limit shared memory and global memory.
- We use CUDA API to access GPU. CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model.
- We don't consider any parallel computing technique in this project, only use single thread to serial access that let us focus our virtual memory implementation.
- There are many kinds of memory in CUDA GPU, we only introduce two memory (global memory and shared memory) which relate to our project.
- Global memory
    - Typically implemented in DRAM
    - High access latency: 400-800 cycles
- Shared memory
    - Extremely fast
    - Configurable cache
    - Memory size is small (16 or 48 KB)

The GPU Virtual Memory we need to design:

- Because the shared memory in GPU with small size and low latency access, we take the shared memory as the traditional CPU physical memory.
- Take the global memory as the disk storage (secondary memory).
- In CUDA, the function executed on GPU that defined by programmer, is called kernel function.
- A kernel function would no longer be constrained by the amount of shared memory that is available. Users would be able to write kernel functions for an extremely large virtual address space, simplifying the programming task.
- Implement a paging system with swapping where the thread access data in shared memory and retrieves data from global memory (secondary memory).
- We only implement the data swap when page fault occur (not in the instruction level).

Specification of the GPU Virtual Memory we designed:

- Global memory (as secondary memory)
  - 128KB (131072 bytes)
- Shared memory (as physical memory)
  - 48KB (32768 bytes)
    - 32KB for data access
    - 16KB for page table setting
- Memory replacement policy for page fault:
  - If shared memory space is available, place data to the available page, otherwise, replace the **LRU** set. Pick the **least indexed** set to be the victim page in case of tie.
- We have to map virtual address (VA) to physical address (PA).
- The valid bit of each page table block is initialized as false before first data access in shared memory.
- Page size
  - 32 bytes
- Page table entries
  - 1024 (32KB / 32 bytes)
- We assume that we can access all memory space by data[ ] array.



Template structure:

- At first, load the binary file, named **"data.bin"** to **input** buffer before kernel launch and return the size of input buffer: **input_size**.

- Launch to GPU kernel with single thread, and dynamically allocate 16KB of share memory, which will be used for variables declared as "extern __shared__"

```
166     /* Launch kernel function in GPU, with single thread
167        and dynamically allocate 16384 bytes of share memory,
168        which is used for variables declared as "extern __shared__" */
169     mykernel << <1, 1, 16384 >> > (input_size, input, results, PAGEFAULT_NUM);
```

- Initialize the page table and entries when have data access operations.

- Under **Gwrite**, you should complete the function to write data into buffer (physical memory).

- Under **Gread**, you should complete the function to read data from buffer.

- Under **snapshot**, together with Gread, you should complete the program to load the elements of **data** array (in shared memory, as physical memory) to **results** buffer (in global memory, as secondary storage), using LRU algorithm.

- Count the PAGEFAULT when executing paging replacement.

- In Host, dump the contents of binary file into "**snapshot.bin**"

- Print out PAGEFAULT when the program finish execution.

Function Requirements (90 points):

- Implement Gwrite to write to data buffer (shared memory, as physical memory) (10 points)

- Implement Gread to read from data buffer (shared memory, as physical memory) (10 points)

- Implement snapshot together with Gread to load the elements of **data** array (in shared memory, as physical memory) to **results** buffer (in global memory, as secondary storage). (10 points)

```
45  __device__ uchar Gread(uchar *buffer, u32 addr)
46  {
47      /* Complate Gread function to read value from data buffer */
48  }
49
50  __device__ void Gwrite(uchar *buffer, u32 addr, uchar value)
51  {
52      /* Complete Gwrite function to write value to data buffer */
53  }
54
55  __device__ void snapshot(uchar *results, uchar* buffer, int offset, int input_size)
56  {
57      /* Complete snapshot function to load elements from data to result */
58  }
```

- Use provided access pattern to test memory management. (5 points)

```
73      // Gread, Gwrite and snapshot are the access pattern for testing page replacement
74      for (int i = 0; i < input_size; i++)
75          Gwrite(data, i, input[i]);
76
77      for (int i = input_size - 1; i >= input_size - 32769; i--)
78          int value = Gread(data, i);
79
80      snapshot(results, data, 0, input_size);
```

- When swapping memory, you need to implement with LRU paging algorithm. (40 points)

- Print out correct page fault number. (5 points)

- Correctly dump the contents to “snapshot.bin”. (10 points)

Report (10 points)

Write a report for your assignment, which should include main information as below:

- How did you design your program?
- What problems you met in this assignment and what is your solution?
- The steps to execute your program.
- Screenshot of your program output.
- What did you learn from this assignment?

Submission

- Please submit the file as package with directory structure as below:
  - **CSC3150_Assignment_3_(Student ID)**
    - Source
      - main.cu
      - data.bin
      - snapshot.bin
    - Report
- Due date: End (23:59) of 15 Nov, 2018

Grading rules

Completion	Marks
Report	10 points
Completed with good quality	80 ~ 90
Completed accurately	80 +
Fully Submitted (compile successfully)	60 +
Partial submitted	0 ~ 60
No submission	0
Late submission	**Not allowed**