GPU Programming

Assignment 4 (13 marks)

Submission Deadline: April 5, 2020 (23:55 pm only on Moodle)

Release Date: March 16, 2020

1 Problem Specification

You are given a list of N processes consisting of their arrival times and execution times. Your goal is to schedule them on M cores in an optimal manner.

You will be using the multi-queue scheduling algorithm with load balancing, a multiprocessor variant of the First Come First Serve (FCFS) scheduling algorithm. There is NO pre-emption, processes once scheduled, execute until completion.

Assume each process has a unique id from 0 to $N - 1$ in the order they are read from the input.

Each core has a processing queue. A new process goes to that queue which is going to get free the earliest. Threads can read the list of processes in any order, but when populating the queues for each core, the following points need to be noted:

1. Processes are added to the queue in the sequence of their process ids.
2. Each process is added to the queue with the lowest sum of execution time remaining. Ties are broken in favor of the lower core id.
3. Assume within a timestep, that execution completes at the beginning of the timestep and process scheduling is done after that.

   For example, if a process starts execution at second 1 and has an execution time of 1 second, it can be removed from the queue at the beginning of second 2 and a process arriving at second 2 can be scheduled on that same core.

1.1 Input Format

- First line of the input contains 2 integers: N and M, the number of processes and the number of cores respectively.
- Each of the next N lines contains 2 space separated integers corresponding to the arrival time and execution time of each process.
- The input is sorted in ascending order of arrival times.
- All arrival and execution times will be integral multiples of seconds.

- We have provided a main function which invokes a `schedule()` function and passes it the following arguments:
- **N**: Number of processes
- **M**: Number of cores
- **arrival_times**: An array containing N arrival times
- **burst_times**: An array containing N execution/burst times

You need to implement the `schedule()` function.

1.1.1 Constraints

- $1 \leq$ N ≤ 20000
- $1 \leq$ M ≤ 10000
- $1 \leq$ all arrival time values and execution time values ≤ 1000
- No more than 2000 processes will have the same arrival time

1.1.2 Sample Input

```
8 4
1 4
1 5
2 3
2 4
2 6
3 4
3 8
7 3
```

1.2 Output Format

The first line of the output contains the overall turnaround time. The next M lines contain the list of the processes executed in cores 0 to $M-1$ in the order they were scheduled.

The **output is printed by the main function**. You **will NOT** print anything in the `schedule()` function. Your implementation of the `schedule()` function will **ONLY** populate the following arguments:

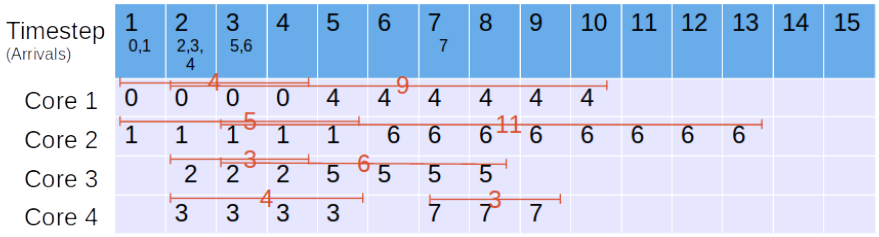
1. **cores_schedule**: Array containing M arrays of processes scheduled in each core (in order)
2. **cs_lengths**: Array containing M lengths of each core's process schedule array
3. **Return Value**: turnaround time.

which will then be printed by the main function.

1.2.1 Sample Output

45
0 4
1 6
2 5
3 7

The figure explains the scheduling at each core and the calculation of the overall turnaround time.



2 Submission Guidelines

Please follow the submission guidelines strictly to avoid penalty:

1. We have provided the main file for execution which will call a `schedule()` function. The main function handles I/O including reading the input and printing the output. Please read through and understand the main function.

2. Please submit a single `ROLLNO.cu` (in caps) with the following function implementation:

   ```
   int schedule(int N, int M, int* arrival_times, int* burst_times,
   int** cores_schedules, int* cs_lengths)
   ```

   The function header is in the provided `kernel.h` file.

3. Do not modify or add anything in any other file, since we will test with the template main function.

4. There should be no print statements in your submitted code since evaluation will be performed by an automated script.

5. We will be timing your program's execution. If your program's execution time is more than twice the average time across all submissions, you will lose 1 mark. The top two submissions having the lowest execution times will get 1 bonus mark each.