CS6023: GPU Programming

Assignment 4 (20 marks)

Deadline : April 24, 2022, 23:55 on Moodle

1 Problem Statement

The aim of this assignment is to write a *Train Ticket Booking* application. There are N trains with the train numbers being $0, 1, 2, \cdots N-1$. Each train has a variable M number of classes (the classes being $0, 1, 2, \cdots M-1$) with each class having a maximum capacity. Each train is also associated with a source station (src) and a destination station ($dest$). The src station is **never** the same as the $dest$ station.

The list of station that the train goes through is either $[src, src+1, src+2, \cdots, dest]$ or $[dest, dest-1, dest-2, \cdots, src]$ depending on whether $src < dest$ or $src > dest$ respectively.

The booking requests are split across batches and they are processed in the following manner:

- All the requests in Batch i are processed before the requests in Batch $i+1$ are processed.
- All the requests within the same batch belonging to the same train and same class are to be processed in order of their *RequestID*. (See Input Format below)
- Requests belonging to different trains / same train and different classes are to be processed in parallel so as to maximise the number of requests being processed at a given time.

2 Input and Output

2.1 Input Format

- First line contains N // the number of trains.
- For each of the N trains, the following is given:
  - Train Number, Number of Classes (M), Source (src), Destination ($dest$)
  - The next M lines contains Class Number, Max Capacity
- Next line contains B //the number of batches
- For each of the B batches, the following is given:
  - R //Number of requests in the given batch
  - Each of the R requests is of the form: Request ID, Train Number, Class, Source, Destination, Number of Seats

2.2 Output Format

For each request in the batch, print either *success* or *failure* depending on whether the request could be processed or not. After each batch is processed, the following is to be printed in different lines:

- Number of requests that succeeded <space> Number of requests that failed
- Total number of seats that were booked in the given batch.

The total number of seats that are booked when a particular request is processed can be calculated in the following manner:

- Say, the request is for booking n seats from src to $dest$.
- Then the number of seats booked is $n \times abs(dest - src)$ where abs is the *absolute* function.

2.3 Constraints

- $1 \leq N \leq 100000$ (number of trains)
- $1 \leq M \leq 25$ (number of classes in each train)

- $1 \leq B \leq 5000$ (number of batches of requests)
- $1 \leq R \leq 5000$ (number of requests in a batch)
- $1 \leq C \leq 100$ (max capacity of any class)
- $1 \leq abs(dest - src) \leq 50$
- $src \geq 0$, $dest \geq 0$

3 Sample Testcase

3.1 Sample Input

```
2                                          //Number of Trains
0 2 57 54          //(Train Num - 0, Num Classes - 2, Src - 57, Dest - 54)
0 10                                       //(Class - 0, Max Capacity - 10)
1 12                                       // ...
1 3 45 49          //(Train Num - 1, Num Classes - 3, Src - 45, Dest - 49)
0 12                                       //(Class - 0, Max Capacity - 12)
1 5                                        // ...
2 10                                       // ...
2                                          //Number of batches of requests
4                                          //4 requests in 1st batch
0 0 0 56 54 4      //(RequestID, TrainNum,ClassNum,Src,Dest,NumSeats)
1 0 0 56 54 4                              // ...
2 0 0 56 54 3                              // ...
3 1 2 45 49 12                             // ...
1                                          //1 request in 2nd batch
0 0 0 57 56 5      //(RequestID, TrainNum,ClassNum,Src,Dest,NumSeats)
```

3.2 Sample Output

```
success
success
failure
failure
2 2
16
success
1 0
```

5

The first three requests in the $1st$ batch of requests try to book seats in the $0th$ class of the $0th$ train. Since the maximum capacity for the mentioned class is only 10, the first and second request succeeds and the third requests fails. The fourth request in the $1st$ batch also fails because the number of seats asked for is 12 whereas the maximum capacity for the asked class is only 10 (train 1, class 2). Thus, in the $1st$ batch there are 2 requests which succeeds and 2 requests which fails. Hence, 2 2 in the $5th$ line of the output. The total number of seats booked is 16 $((4 \times 2) + (4 \times 2))$.

There is only 1 request in the $2nd$ batch and it succeeds because the requirement is satisfied.

4 Points to be noted

- Say, a booking request is from *src* station to *dest* (and $src < dest$) station for n seats succeeds. Then, all the n seats are occupied from station *src* to station $(dest - 1)$ and the seats become free at the *dest* station. For example, if the max capacity for a particular class is 50, *src* station is 0, *dest* station is 10, then both the following requests succeed:
  - 50 seats from 0 to 5.
  - 50 seats from 5 to 10.
- For this assignment, the reading/writing of input/output, use of data structures, kernel prototypes has been completely left to your choice.
- Sequential implementation (apart from where absolutely needed) of the above functionality will lead to '0' marks on the assignment.
- Test your code on large inputs.

5 Submission Guidelines

- Name the file as 'RollNumber.cu' which contains the implementation of the above-described functionality.

- Submit **only** the 'RollNumber.cu' file to Moodle. For example, if your roll number is CS17B049, then the name of the file to be submitted on Moodle is 'CS17B049.cu'.
- Read input from standard input and print the output to standard output so that we can use I/O redirection for the evaluation.
- After submission, download the file and make sure it was the one you intended to submit.
- Non-adherence to the above guidelines **will** lead to a penalty.

6 Learning Suggestions

- Write a CPU-version of code achieving the same functionality. Time the CPU code and GPU code separately for large matrices and compare the performances.
- Exploit shared memory as much as possible to gain performance benefits.