CS6023 (GPU Programming): A2 (7 marks)

Deadline: 01-Mar-2020 23:55

Submission Link: https://courses.iitm.ac.in/mod/assign/view.php?id=39725

0. Aim

This assignment helps to understand/learn two concepts: Synchronization and Memory coalescing.

1. Problem Specification

Input: Matrix A of size m x n

Output: Matrix B of size (m+1) x (n+1)

Given a matrix $A_{m \times n}$. Construct a matrix $B_{p \times q}$ where p = m+1 and q = n+1 using the following:

- The last entry of i^{th} row of B should hold the rowSum(i) = sum of all i^{th} row entries of A, 1<= i <=m.

- Similarly, the last entry of j^{th} column should hold colSum(j) = sum of all j^{th} column entries, 1<=j<=n.

- Find the minimum entry (say x) from last row and last column of B (computed in previous steps).

- For other entry: B(i,j) = A(i,j) + x for all 1<= i <= m and 1<= j <= n.

- Store x as the last entry of B. That is, B(p,q) = x.

Example (m,n)=(3,4)

$$\begin{array}{cccc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{array} \quad ===== > \quad \begin{array}{cccc|c} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{24} & b_{35} \\ \hline b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \end{array}$$

Objective

Compute B matrix using three different kernels (if sticking to Section 4 guidelines) following the launch configuration and input/output specification.

- **(m*n)/k** threads where k is the number entries processed by a thread (Here, DoC = 32 – k). If k=1 then one thread operates on one entry, that is, otherwise called as fully coalesced.
- Assume the matrix A is stored as a single dimension integers in row-major order (int* A). For k > 1, **m*n** is divisible by **k**.
- B matrix (int* B) must be populated in row-major order and printed as in Section 2.2.

2. Input and Output Formats

1. Input

m n k

a1 a2 a3 .... // all m*n entries separated by space

2. Output

b_{11} b_{22} …… b_{1q}
b_{21} b_{22} …… b_{2q}
…… b_{3q}
b_{p1} b_{p2} … … b_{pq}
// Print as 2D Matrix format

3. Limits and Constraints

- $-100 <= a_{ij} <= 100$
- $2^1 <= m <= n <= 2^{13}$
- k = {1, 2, 4, 8, 16, 32}. Default k = 1.
- INT_MIN < RowSum, ColSum < INT_MAX.

4. Example (m, n) = (3, 4)

Step 0

0	1	2	3	-
4	5	6	7	-
8	9	10	11	-
-	-	-	-	-

Step 1. rowSum/colSum

0	1	2	3	6
4	5	6	7	22
8	9	10	11	38
12	15	18	21	-

Step 2. FindMin. Here x=6

0	1	2	3	6
4	5	6	7	22
8	9	10	11	38
12	15	18	21	-

Step 4. Update

6	7	8	9	6
10	11	12	13	22
14	15	16	17	38
12	15	18	21	6

3. Submission Guidelines

Please follow the submission guidelines seriously to avoid penalty.

1. The main() function reads the input and writes the output from/to stdin/out as specified in Section 2. Do NOT use file reading.

   e.g After compilation, to run you can use the input redirection. **./a.out < 1.txt**

   where 1.txt is the input file.

2. There should NOT be any other printf/scanf statement. The evaluation is semi-automated.
3. Submit a single CUDA (.cu) file containing the required kernels and main() on the Moodle.
4. File name should be ROLLNUMBER.cu. Replace ROLLNUMBER with your roll number in capital letters.
5. Download your file, and make sure it was the one you intended to submit.
6. Non-compliance to any of the above guidelines invites **penalty**.

4. Kernel Signatures and Hints

- __global__ void sumRandC (int* A, int* B, int m, int n, int p, int q, int k=1)
- __global__ void findMin (int* A, int* B, int m, int n, int p, int q, int k=1)
- __global__ void updateMin (int* A, int* B, int m, int n, int p, int q, int k=1)
- There is no strict constraint on using the above kernels, but we think this is a good /recommended design decision for the three steps/phases. Some arguments are redundant. You may use other device functions and/or device/shared variables as required.
- [Hint] Coding the CPU computation first always helps in writing the GPU kernels better.
- [Hint] It is advisable to code for k = 1 and later incorporate computation for k > 1.
- [Learning purpose only] Plot as a graph (or in a spreadsheet) with executing times of the kernels vs varying k values on the same input. Not for submission! **nvprof ./a.out < 14.txt**