Assignment 1

Implement the following host, kernel programs and execute on an NVIDIA GPU. 2D convolution is primarily used in image processing for tasks such as image enhancing, blurring, etc. Let us consider 2D convolution operation for a 2D Matrix of floating point numbers. It is a neighborhood operation where each output element in the output matrix (OUT) is the weighted sum of a collection of neighboring elements of the input matrix (IN). The weights used in the weighted sum are typically stored in an array called the convolution mask or filter (M). The convolution formula for a 3x3 convolution filter M with a matrix IN of size nxn is:

$$P(i,j) = \sum_{a=0}^{2}(\sum_{b=0}^{2}(M[a][b]*IN[i+a-1][j+b-1])) \text{ where } 0 \leq i \leq n \text{ and } 0 \leq j \leq n$$

For Example.
Input
n=3
IN
3.0 3.0 3.0
3.0 3.0 3.0
3.0 3.0 3.0

M
 0 -1 0
-1 5 -1
 0 -1 0
OUT
9.0 6.0 9.0
6.0 3.0 6.0
9.0 6.0 9.0

Note for boundary elements, a padding of 0.0 is assumed. In the above example, for element input[0][0], placing the 3x3 mask with its centre at IN[0][0], produces OUT[0][0] = 0.0*0+0.0*-1 +0.0*0 + 0.0*-1 + IN[0][0]*5 + IN[0][1]*-1 + 0.0*0 + IN[1][0]*-1 +IN[1][1]*0 =0.0+0.0+0.0+0.0+15.0-3.0+0.0-3.0+0.0 = 9.0.

Implement a CUDA program which takes as input, two variables n and m, and a matrix N where

i) n = argv[1] is the width of the input square matrix
ii) m = argv[2] is the width of the square mask matrix (m must be odd).
iii) N is the input matrix stored in a text file, with n*n lines, each line having one element of the matrix in a row major order.

You can initialize the mask internally in your program.

The output will be an NxN matrix where each element in a row is written in a new line in the row major order.

TODO: Submit a complete archive to the TA in his email id by 25th August 9pm. Code should not be AI generated, code should not be plagiarised. If caught, you will be banned from all submissions.

Profile the program using NVIDIA profiling tools to determine the number of global memory load transactions and the total memory consumption (in bytes), and present these results along with the program's output

Report various classes of operations (branch, arithmetic, memory etc) and their relative time taken. Provide the data, charts etc in a separate report. In this report also justify whether using shared memory will help in this program.

Put both report and code in the archive along with EXACT commandline instructions on how to run your code and reproduce the profiling data.

Refer to the following input output example format.

argv[1] = 3
argv[2] = 3

Input.txt
3.0
3.0
3.0
3.0
3.0
3.0
3.0
3.0
3.0

Output.txt
9.0

6.0
9.0