Convolution

GPU Teaching Kit — Accelerated Computinq

OBJECTIVE:

The lab's objective is to implement a tiled image convolution using both shared and constant memory. We will have a constant 5x5 convolution mask, but will have arbitrarily sized image (assume the image dimensions are greater than 5x5 for this Lab).

To use the constant memory for the convolution mask, you can first transfer the mask data to the device. Consider the case where the pointer to the device array for the mask is named M. You can use *const float *__restrict__ M* as one of the parameters during your kernel launch. This informs the compiler that the contents of the mask array are constants and will only be accessed through pointer variable M. This will enable the compiler to place the data into constant memory and allow the SM hardware to aggressively cache the mask data at runtime.

Convolution is used in many fields, such as image processing for image filtering. A standard image convolution formula for 5x5 convolution filter H with an Image I is:

$$P_{i,j,c} = \sum_{x=-2}^{2} \sum_{y=-2}^{2} I_{i+x,j+y,c} * M_{x,y}$$

where Pi,j,c is the output pixel at position i , j in channel c, Ii,j,c is the input pixel at the i,j in channel c (the number of channels will always be 3 for this MP corresponding to the RGB values), and Mx,y is the mask at position x , y.

INPUT DATA:

The input is an interleaved image of height * width * channels. By interleaved, we mean that the the element I[y][x] contains three values representing the RGB channels. This means that to index a particular element's value, you will have to do something like:

```
index = (yIndex*width + xIndex)*channels + channelIndex;
```

For this assignment, the channel index is 0 for R, 1 for G, and 2 for B. So, to access the G value of I[y][x], you should use the linearized expression I[(yIndex*width+xIndex)*channels + 1].

For simplicity, you can assume that channels is always set to 3.

INSTRUCTIONS:

Write the code to perform the following:

- allocate device memory
- copy host memory to device
- initialize thread block and kernel grid dimensions
- invoke CUDA kernel
- copy results from device to host
- deallocate device memory
- implement the tiled 2D convolution kernel with adjustments for channels
- use shared memory to reduce the number of global accesses, handle the boundary conditions in when loading input list elements into the shared memory