Fall 2021
ECE 5770: GPU Accelerated Computing
Prof. Shadi G. Alawneh
Deadline: Wednesday Nov 17 11:59 pm

Assignment 4

Convolution

Description

The objective of this assignment 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 assignment).

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 a 5x5 convolution filter M 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 i,j in channel c (the number of channels will always be 3 for this assignment 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 x width x channels. By interleaved, we mean that 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 the channels is always set to 3.

Instructions

Edit 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

Instructions about where to place each part of the code is demarcated by the //@@ comment lines.

Pseudo Code

A sequential pseudo code would look something like this:

```
maskWidth := 5
maskRadius := maskWidth/2 # this is integer division, so the result is 2
for i from 0 to height do
  for j from 0 to width do
    for k from 0 to channels
      accum := 0
      for y from -maskRadius to maskRadius do
        for x from -maskRadius to maskRadius do
          xOffset := j + x
          yOffset := i + y
          if xOffset >= 0 && xOffset < width &&
             yOffset >= 0 && yOffset < height then
            imagePixel := I[(yOffset * width + xOffset) * channels + k]
            maskValue := K[(y+maskRadius)*maskWidth+x+maskRadius]
            accum += imagePixel * maskValue
          end
        end
      end
      # pixels are in the range of 0 to 1
      P[(i * width + j)*channels + k] = clamp(accum, 0, 1)
    end
  end
end
```

where `clamp` is defined as

```
def clamp(x, lower, upper)
  return min(max(x, lower), upper)
end
```

Local Setup Instructions

Requirements

-=-

libwb:

https://github.com/abduld/libwb

Compile instructions (Unix)

-=-

You will need libwb (a library by the authors of the teaching kit) for all assignments in this course.

The easiest way to get everything working would be to create a directory for class assignments, and place libwb in that directory alongside the individual assignment directories.

To build libwb:

```
cd [path_to_libwb]
mkdir build
cd build
cmake ..
make
```

To build Convolution:

```
cd [path_to_ Convolution]
mkdir build
cd build
cmake ..
make
```

This should just work if the Convolution and libwb directories are in the same parent directory.

Testing

-=-

The "Dataset" directory contains some sample input / output data to test your implementation. To test a particular input, run (from the Convolution root directory):

```
./build/Convolution  -i Dataset/##/input0.ppm,Dataset/##/input1.raw
 -e Dataset/##/output.ppm  -o output.raw -t image
```

The images are stored in PPM (P6) format; this means that you can (if you want) create your own input images. The easiest way to create image is via external tools such as bmptoppm. The masks are stored in a CSV format. Since the input is small, it is best to edit it by hand.

This will produce some output, part of which should indicate whether or not your output matches the expected output.

To test all 7 examples at once, run (again from the Convolution root directory):

```
python test.py
```

Upon completion, this should tell you how many of the samples you produced the correct output for, and indicate which ones (if any) were incorrect.

Submission

-=-

You can simply submit your solution on Moodle. There is no need to include the dataset or any build files, just the Convolution.cu is fine!