CSUS COLLEGE OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science

CSc 196U – Parallel Programming with GPUs
Fall 2018 / Dr. Muyan

Assignment#2 (Convolution)

Due Date: Thursday, Nov 1st 11:59PM

Steps for solving the assignment (refer to "Assignment Environment" lecture slides for details):

- Download the assignment from the SacCT and unzip it (now you have "A2" directory - you must have already done this to read this a2-instructions.pdf file).
- Build the source using CMake (in "build" directory that you create under "A2" directory).
- Copy the Convolution dataset directly under "build" directory (under "A2/datasets" directory, we have only one dataset directory called Convolution, copy it to "A2/build", e.g., copy contents of "A2/datasets/Convolution" to "A2/build/Convolution").
- For the Convolution problem (which is the only problem provided in A2):
    - Set the project properties as indicated in the below "Project Setup" instructions (e.g., set command line options).
    - Update the template code (template.cu) as indicated in the below "Coding" instructions to solve the problem.
    - Build your solution under Visual Studio (right click on the project and hit "Build") and run it under Visual Studio (right click on the project and hit "Debug -> Start new instance").
    - Make sure your executable (located under "Debug" directory that resides under "build" directory) works from the command line as indicated in the below "Command-line Execution" instructions.
    - Create a new directory by giving it the name of the problem (i.e., Convolution).
    - Copy the executable file (.exe file) and the updated template code (template.cu) to this newly created directory. Also copy the pdf file that includes your answers to the questions.

- Zip the newly created directory in a file named as YourLastName-YourFirstName-a#.zip (e.g., Doe-Jane-a2.zip).
- Build (using CMake and Visual Studio) and test your assignment in a lab machine (see the syllabus for tips) and **create a TEXT (i.e., not a pdf, doc etc.) file called "readme.txt"** that indicates the lab and the lab machine you have used to test your assignment. You may also include additional information you want to share with the grader in this text file. Make sure you have the executable files (.exe files) tested on the lab machine in the zip file mentioned above.
- Submit the **TWO files you have created above (zip file and txt file) SEPERATELY to Canvas** (i.e., do NOT place the txt file inside the zip file). You will receive the grader comments on your text file when grades are posted.

Convolution dataset directory provides **five test cases**. You should test your program with all the provided test cases by updating the command line options provided in the below "Project Setup" instructions. For instance, to test *test_case_1* of Convolution (located at "build\Convolution\Dataset\1") you should update the command line option provided below (which tests *test_case_0* located at "build\Convolution\Dataset\0") by changing all subtexts that say "...\Dataset\0\..." with "...\Dataset\1\...".

Specific Instructions for Convolution Problem:

Objective

The objective of this problem 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. We will also keep the output tile size at 16x16.

To use the constant memory for the convolution mask, you should first transfer the mask data to the global memory of the device (using `cudaMalloc()` and `cudaMemcpy()`). Then (given that the pointer to the device array for the mask is named M) the signature of your kernel function should include `const float * __restrict__ M` as one of the parameters as indicated in the class notes. This informs the compiler that the contents of the mask array are constants 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 $P_{i,j,c}$ is the output pixel at position `i,j` in channel `c`, $I_{i,j,c}$ is the input pixel at `i,j` in channel `c` (the number of channels will always be 3 for this problem corresponding to the RGB values), and $M_{x,y}$ 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 problem, the `channelIndex` 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]`.

As mentioned above, the value of `channels` is always set to 3.

Pseudo Code and Approach

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
```

In this problem, you are expected to write a kernel that performs the same operation that is performed by the above sequential code (e.g., you also need to clamp your output values).

To implement your kernel you can either make your block size as big as the output tile (Design 1) or make your block size as big as the input tile (Design 2).

Coding

Edit the code in the code tab 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 when loading input list elements into the shared memory
  clamp your output values

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

Project Setup

To test your program on test_case_0:

- Right click on the Convolution_Template project -> Properties -> Configuration Properties -> Debugging -> Commmand Arguments and enter the following:

```
-e .\Convolution\Dataset\0\output.ppm

-i

.\Convolution\Dataset\0\input0.ppm,.\Convolution\Dataset\0\input
1.raw

-o .\Convolution\Dataset\0\myoutput.ppm -t image >

.\Convolution\Dataset\0\result.txt
```

(all in one line - do not forget to put spaces between the sub-lines - above you see five sub-lines)

Here, `input0.ppm` is the input image and `input1.raw` is the mask.

You will see the output of your execution in `result.txt` which resides under `build\Convolution\Dataset\0\`

Command-line Execution

The executable generated as a result of compiling the project (`build\Debug\Convolution_Template.exe`) can be run from the command-line using the following command (make sure you are in `build` directory):

```
.\Debug\Convolution_Template

-e .\Convolution\Dataset\0\output.ppm

-i

.\Convolution\Dataset\0\input0.ppm,.\Convolution\Dataset\0\input
1.raw

-o .\Convolution\Dataset\0\myoutput.ppm -t image >

.\Convolution\Dataset\0\result.txt
```

(all in one line - do not forget to put spaces between the sub-lines - above you see six sub-lines)

Questions

1) Name 3 applications of convolution.

2) How many floating operations are being performed in your convolution kernel in terms of variables that indicate mask width, input image size, and number of channels (utilize the same variable names used in the template.cu e.g., utilize *MASK_WIDTH* to indicate mask width)? explain.

3) How many global memory reads are being performed by your kernel in terms of variables that indicate output tile width, mask width, input image size, and number of channels (utilize the same variable names used in the template.cu e.g., utilize *O_TILE_WIDTH* to indicate output tile width)? explain. Assume that ghost elements also require global reads.

4) How many global memory writes are being performed by your kernel in terms of variables that indicate input image size and number of channels (utilize the same variable names used in the template.cu e.g., utilize *imageWidth* to indicate image width)? explain.

5) What do you think happens as you increase the mask width (e.g., you increase it from 5 to 1024) in your kernel while you keep your output tile width at 16? What do you end up spending most of your time doing? Do you think using a tiled convolution approach would still be a good solution in this case? Hint: think of the shared memory size.

6) Do you have to have a separate output memory buffer, e.g., called P, in your kernel? Put it in another way, why can't you perform the convolution in place (by just using the input memory buffer, e.g. called N)?

7) What is the identity mask? What would be the values in it for a mask width of 5 for 2D convolution?