ECSE 420 – Parallel Computing – Fall 2020

Lab 0 – Simple CUDA Processing

Due: October 2, 2020 at 11:59 pm

In this lab, you will write code for simple signal processing, parallelize it using **CUDA C/C++**, and write a report summarizing your experimental results. We will use PNG images as the test signals.

1. Image Rectification

Rectification produces an output image by repeating the following operation on each element of an input image:

$$output[i][j] = \begin{cases} input[\text{i}][\text{j}] & \text{if } input[\text{i}][\text{j}] \geq 0 \\ 0 & \text{if } input[\text{i}][\text{j}] < 0 \end{cases}$$

Figure 1 shows rectification performed on a small array.

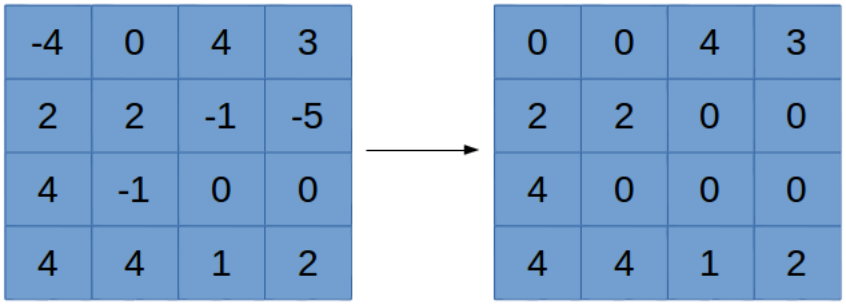

Figure 1: Rectification Example

Of course, since pixel values are already in the range [0,255], rectifying them will not change their values, so you should "center" the image by subtracting 127 from each pixel, then rectify, then add back 127 to each pixel (or equivalently, compare the pixel values to 127 and set equal to 127 if lower).

Write code that performs rectification. Parallelize the computation using CUDA kernels. Measure the runtime when the number of threads used is equal to {1,2,4,8,16,32,64,128,256}, and plot the speedup as a function of the number of threads. Discuss your parallelization scheme and your speedup plots and make reference to the architecture on which you run your experiments. Include in your discussion the *3 test*

images provided to you and *the output* of rectifying those images.

To check that your code runs properly, run the following command:

./rectify <name of input png> <name of output png> < # threads>

When the input test image is “test_1.png”, the output of your code should be identical to “test_1_rectify.png”. You can use the “test equality” code provided by TAs to check if two images are identical. The grader should be able to run your code with different numbers of threads, and the output of your code should be correct for any of those thread counts.

2. Pooling

Pooling compresses an image by performing some operation on regularly spaced sections of the image. For example, Figure 2 shows max-pooling on 2x2 squares in an image. In this case, the operation is taking the maximum value in the section, and the sections are the disjoint 2x2 squares.

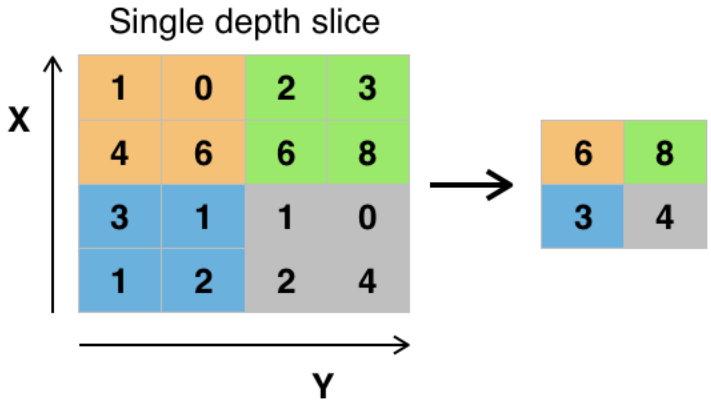

Figure 2: Max Pooling

Write code that performs 2x2 max-pooling. Analyze, discuss, and show an example image as described in the first section.

To check that your code runs properly, run the following command:

./pool <name of input png> <name of output png> < # threads>

When the input test image is “test_1.png”, the output of your code should be identical to “test_1_pool.png”. Note that if the input image is of size m by n, then the output of 2x2 max-pooling will be of size *m*/2 by *n*/2. In this exercise, do not worry about the case in

which the image cannot be divided perfectly into 2x2 squares – you may assume that the input test image will have even width and height lengths.

Info regarding the signal (PNG Images):

To help you load and produce PNG images a source code is provided. You are also allowed to use any other source code as you please.

The source code is named **lodepng.cu** (source file) and **lodepng.h** (header file). There is also an example (**lodepng_test.c**) provided on how to use the source code to load (decode) and produce (encode) png images.

It will be further discussed in the Tutorials (which will be recorded).

Submission Instructions:

Each group should submit a single zip file with the filename *Group_<your group number>_Lab_0.zip*. (Ex – Group_09_Lab_0.zip). The zip file should contain the following:

1. A **lab report** answering all the questions in the lab (please use a PDF format).
    a. Must be named *Group_<your group number>_Lab_3_Report*. (Ex – Group_03_Lab_0_Report.pdf).
    b. Must have a cover page (Include Group number, members' names and ID).
    c. Must follow the logical order for the lab discussions.
2. Y**our own source** code. For this lab **please use CUDA C/C++.** Add the whole project solution.
3. The **output test images** (raw .png) for the grader to validate.