School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS4040A & COMS7045A Assignment 2

Hand-out date: 10:00 am, March 14, 2019
Due date: 24:00 pm, April 14, 2019

Instructions

1. This is an individual assignment.
2. The due date is strictly before 24:00 pm, April 14, 2018. Late submission will be penalized by 10% to 50% of the total marks.
3. Hand in the electronic files and source codes on Sakai course site.
4. The total marks for this assignment is 52 for both Hons students and MSc students. The weight of this assignment towards the final mark is 13%.

Outcome

1. Write programs in CUDA for CPU + GPU systems;
2. Optimize the performance of a CUDA program by using CUDA memory hierarchies efficiently.

1 Image Convolution Using CUDA C

1.1 Introduction

Convolution is an array operation where each output data element is a weighted sum of a collection of neighbouring input elements. The weights used in the weighted sum calculation are defined by an input mask array, referred to as the convolution mask here. The same convolution mask is typically used for all elements of the array. Convolution is commonly used in various forms in signal processing, digital recording, image processing, video processing, and computer vision. In these application areas, convolution is often performed as a filter that transforms signals and pixels into more desirable values.

Convolution typically involves a significant number of arithmetic operations on each data element. For large data sets such as high-definition images and videos, the amount of computation can be very large. Each output data element can be calculated independently of each other, a desirable trait for parallel computing. On the other hand, there is substantial level of input data sharing among output data elements with somewhat challenging boundary conditions. This makes convolution an important use case of sophisticated tiling methods and input data staging methods.

When applied in image processing tasks, convolution leads to results such as noise removal, edge detection and sharpening of details etc. If an image is represented as a 2D discrete signal $Y \in \mathbb{Z}^{M\times N}$, we can perform the (discrete) convolution

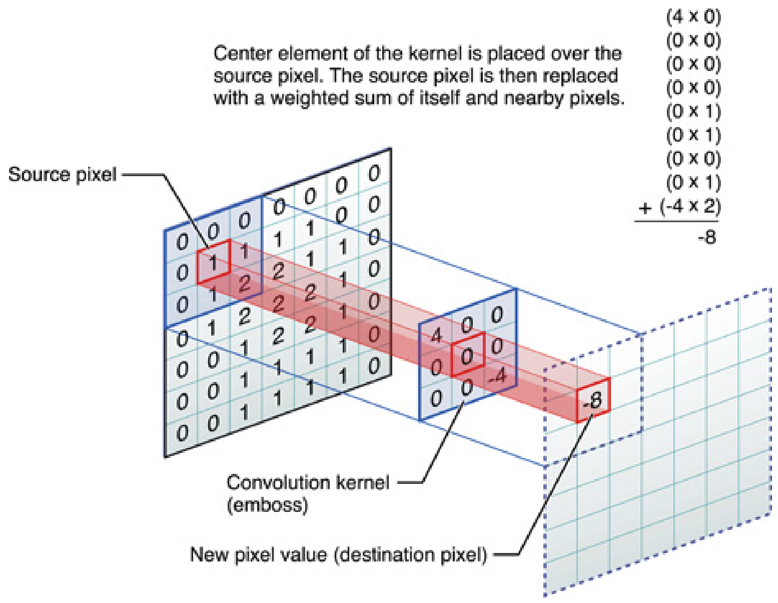
Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Figure 1: Performing convolution operation

in 2-dimension using a kernel $F \in \mathbb{Z}^{L \times P}$ as

$$(Y * F)[i,j] = \sum_{l=0}^{L-1} \sum_{p=0}^{P-1} Y[i-l, j-p] F[l,p], \quad 0 \leq i < M,\ 0 \leq j < N. \tag{1}$$

The convolution operation in Equation (1) is actually a scalar product of the filter weights and all pixels of the image within a window that is defined by the extent of the filter and a center pixel. Figure 1 illustrates the convolution using a small 3×3 kernel. The design of the convolution filter requires careful selection of the weights to achieve desired results.

1.2 Implementation Considerations

Image convolution can be efficiently implemented on massively parallel hardware, since the same operator gets executed independently for each image pixel. A naive implementation would simply execute the convolution for each pixel by one CUDA thread, read all values inside the filter area from the global memory, and write the result. (Note that this approach is inefficient.)

In this assignment, you are going to implement image convolution using the following methods.

1. Serial computation **10%**
2. The naive parallelization approach **25%**
3. Using shared memory **30%**
4. Using constant memory **10%**
5. Using texture memory **25%**

In your implementation:

1. The filters in Table 1 can be used for testing the convolution result.

2. A `pgm` test image is given. To load a `pgm` image or write a `pgm` image, you may use the relevant CUDA SDK functions. A CUDA SDK sample program is provided in this regard.
3. The filtered values outside the image boundaries, referred to as 'halo cells', should be treated as 0 values.
4. Your code should be written in a way that the size of convolution mask is arbitrary, that is, you should test it on different sizes of convolution masks, e.g., 3 by 3, 5 by 5, 7 by 7 etc.

Sharpening	Edge Detection	Averaging
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Table 1: Convolution masks

2 Submission

1. Your submission should be a single compresses file named as `yourStudentNo_hw2.tar.gz` that when extracts, gives me a folder named `yourStudentNo_hw2`. In this folder, you may include subfolders of source files for different approaches, `Makefiles` to build your code, a `readme` file on building and running your program, as well as a report named `report.pdf`. This document will be the main source of evaluation of the quality of your work.
2. Here is the list of contents and discussions you should put in your report.
   - A description of the design choices you tried and how did they affect the performance.
   - Performance comparison among the serial and different parallel approaches. In particular, include discussions on the following:
     (a) Performances using different sizes of input images, and using different sizes of 'sharpening' and 'averaging' convolution masks. (You don't need to change the size of edge detection convolution mask.)
     (b) How many global memory reads are being performed by your kernel?
     (c) What is the measured floating-point computation rate for the CPU and GPU kernels in this application? How do they each scale with the size of the input?
     (d) How much time is spent as an overhead cost for using the GPU for computation? Consider all code executed within your host function with the exception of the kernel itself, as overhead. How does the overhead scale with the size of the input?
   - A discussion on using different types of CUDA device memory.
   - References, including open source code projects, should be cited properly in your report.
   - **Your report should be in single column format, use 11pt, single line spacing, and if any tables or diagrams are in it, they should be clearly readable. Finally, write in a style of scientific writing, and please avoid putting in non-technical staff in your report.**