COMS4040A & COMS7045A Assignment 2

Hand-out date: 10:00 am, April 23, 2021
Due date: 23:59 pm, May 17, 2021

Instructions

1. This is an individual assignment.
2. Before the final hand in, submit your write-up PDF file to Turnitin to produce a similarity check report.
3. Hand in the electronic files and source codes on Ulwazi course site. See more instructions in Section **Hand-in**.
4. The total marks available for this assignment is 50 for both Hons students and MSc-CWRR students.
5. Submit on time to avoid late submission penalties.

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

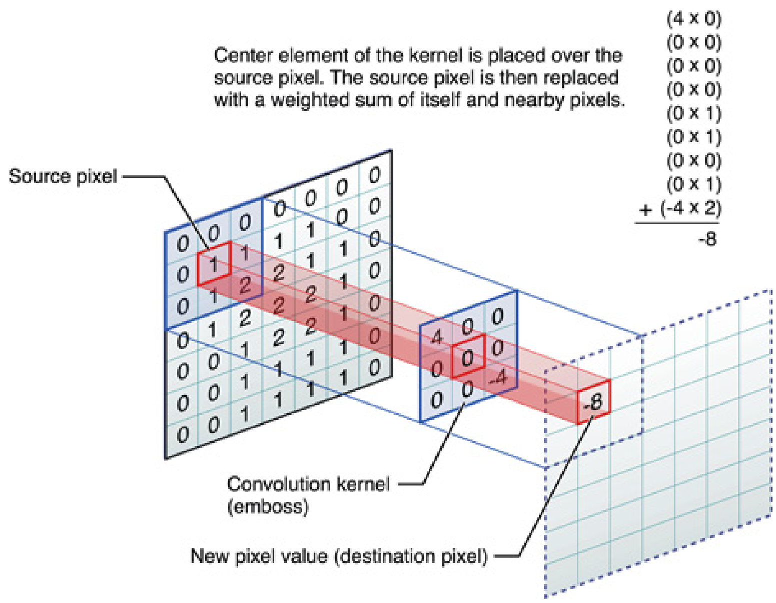

Figure 1: Performing convolution operation

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 centre 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. CUDA implementation using both global memory and constant memory [12]
3. CUDA implementation using both shared memory and constant memory [16]
4. CUDA implementation using texture memory [12]

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 pixel values outside the image boundaries should be treated as 0 values.
4. Your code should be written in a way that the size of convolution mask is arbitrary in the case of averaging filter, 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 Hand-in

1. Submit the report and source code separately via the submission links provided. Your report should be named `yourStudentNo_hw3.pdf`, and submitted to Turnitin to produce a similarity check report. Note that your report plays an important role for marking. Your source code submission should be a single compressed file named as `yourStudentNo_hw3.tar.gz` that when extracts, gives me a folder named `yourStudentNo_hw3`. In this folder, you may include subfolders of source files for different approaches, `Makefiles` to build your code, run script(s), such as `run.sh`, to run your program.
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 all approaches implemented.
   - Performances using different sizes of input images, and using different sizes of 'averaging' convolution masks. (You don't need to change the size of other types of convolution masks.)
   - Display output images from each implementation using a format with 2-4 images in a row, with clear captions and proper referrals to these in the text.
   - References, including open source code projects, should be cited properly in your report.
   - Your report can be written either in single column or double column format. Use 11pt (single column) or 10pt (double column), and single line spacing. If tables or diagrams are used, they should be clearly readable and properly sized. (Poor quality of a report will incur negative marks.)