COMS4040A & COMS7045A Assignment 2 – Report

Shameel Nkosi and Nomasonto, 1814731, Coms Hons

May 23, 2021

Contents

This page is intentionally left blank

Introduction

Convolutions

In this report I discuss convolutions an different implementations. Convolutions in the context of this report are nothing more than processing images digital. Convolutions can however, be applied to images, video or even audio. The goal to achieve the task of as quick as possible given that having a lot of computations can have notable performance issues.

The images we are reading in are gray-scale images. Each image being read has width and height. This means that there are pixels on the x-axis as well as the y-axis. Since the picture is gray-scale, each pixel has a single value instead of a *rgb* vector of 3 values per pixel. This makes our job slightly easier.

Design Style

Initial implementation

The images here are loaded as single array of values of size *width* x *height* instead of a 2 dimensional array of dimension *(width,height)*. Initially, the design style I employed was one that worked on an *n by m* matrix which worked pretty well and was easy to maintain and keep track of the indices. This however, required me to turn the loaded image into a two dimensional array and then back to a one dimensional upon being processed so that it can be written to a file.

Current implementation

In the implementation submitted, The code works on one dimensional arrays throughout. This makes reading and writing of images easy and it also enables one to avoid making mistakes of deforming the image. As required, there are 4 different implementations done here:

- **Serial implementation:** In this implementation everything is done sequentially, one computation at a time by a single processor.
- **Parallel with global and constant memory:** The parallel code is implemented using the CUDA programming interface. Here we have a number of threads working together to achieve a common goal, executing the same instruction on different parts of the memory. Memory here is on disk, therefore for every thread executing, the thread makes a request for an item in memory.
- **Parallel with shared and constant memory:** Similar to the point above with a small, here memory is shared and therefore cached. This means that if a block is shared, a request is made only once for a block of memory, threads sharing memory use the cached or fetched memory. This avoids the traffic to and from main memory. We therefore, expect this implementation to be better than the one above.

- **Parallel with texture memory:** This is read only memory that aims to gives us better performance gains compared to any on the implementations mentioned above. Texture exploits the idea of spatial memory access, which states that if you access memory, you are likely to use the neighboring memory blocks, therefore when a request is made, neighboring memory comes with that requested memory address, which is what we need in the purposes of the our assignment.

Performance

Performance Comparisons Among All Approaches

In this section we show the performance comparisons among all approaches but not for all pictures. The reason why I will not show for all pictures that were used in this assignment is because the all pictures were of the exact size and dimensions, and therefore the output for each we very similar. This will be proven by means of pictures or screenshots and then I will explain the results.

Picturure	Serial	Global memory	Shared memory	Shared memory
Image21.pgm	15.653	0.094	0.090	0.089
Lena_bw.pgm	15.646	0.161	0.092	0.092
Man.pgm	15.555	0.092	0.092	0.090
Mandrill.pgm	15.713	0.092	0.092	0.089

Figure: 3x3 kernal generated picture for averaging time in ms

Averaging: Above are the times taken for each implementation per picture. With no doubt we can see that the Serial time is far far greater than any times tabulated. This is because every block or entry in memory for the output image is calculated one after the other. We expect the texture memory results to be the best and with no doubt they are as expected and explained in the design styles section of this report. Though the differences aren't significantly different, shared memory outperforms global memory on average but not all time. Please note that the reasons mentioned here for performance will apply to the tables we are going to list below for other kernels.

Picturure	Serial	Global memory	Shared memory	Shared memory
Image21.pgm	15.503	0.141	0.099	0.088
Lena_bw.pgm	15.585	0.09	0.1	0.088
Man.pgm	15.628	0.091	0.1	0.087
Mandrill.pgm	15.714	0.092	0.099	0.088

Figure: 3x3 kernal generated picture for sharpening in milliseconds

Sharpening: Here we can conclude that serial time and texture time are living up to what we expect of them. Though shared memory is required to outperform global at all times, we fail to achieve that. The reason is due to the initialization of the tile size memory that has to be shared. This could however been optimized had I had more time to work on the code.

Picturure	Serial	Global memory	Shared memory	Shared memory
Image21.pgm	15.608	0.137	0.106	0.088
Lena_bw.pgm	15.674	0.141	0.101	0.090
Man.pgm	15.512	0.091	0.100	0.091
Mandrill.pgm	15.744	0.111	0.100	0.088

Figure: 3x3 kernal generated picture for Edge detection in milliseconds