PARALLELIZING PIXEL COMPUTATIONS FOR RETINAL IMPLANTS

EECS 587: PROJECT REPORT

Andrew McMullin, Neha Kumar

UM ID: 5347 3130 and 5755 4297

Department of Computer Science and Engineering

University of Michigan
Ann Arbor, MI

{mcmullin, nehark}@umich.edu

Fall 2021

Contents

1 Introduction **1**

2 Methodology **1**

3 The Parallelism **2**

3.1 GPU Processing 3

3.1.1 Initial Approach 3

3.1.2 Final Approach 3

4 Observations **4**

5 Conclusion **4**

6 Future Scope **4**

ABSTRACT

Retinal Implants are developed to substitute for the dead photoreceptors in the eyes of patients who suffer from a damaged retina. The implants, that will be connected to the optic nerve, will contain a chip that would replicate the functions of a retina in the affected eye. The output from the device will be an electrical pulse stream that sends the perceived image as a signal that the brain could comprehend. Such implants require a level of processors and parallelism in the application. In this report, we focus on different ways of parallelizing the pixel processing by mimicking the inputs that a retinal implant would encounter.

Keywords Retinal Implant · Parallel Algorithms · GPU · Shared Memory · Edge Detection Algorithms

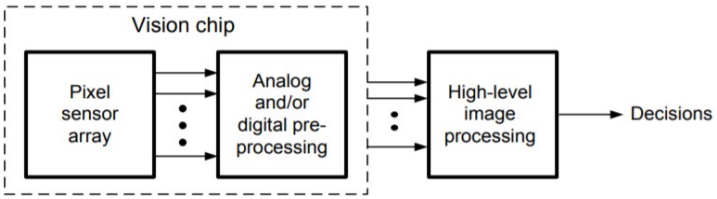

Figure 1: The Image Processing System (2)

1 Introduction

The application's requirements to develop a retinal implant are very complex. It is extensively crucial to consider the biological compatibility, the low power dissipation needed, and high error tolerance. In addition, it also requires tiny processors and massive parallelism. Our goal is to solve one of these requisites by parallelly processing the light images that are sensed by the device, by incorporating grayscaling and edge detection.

The processing can be described by the design represented in Figure 1(2). Due to the huge amount of pixel streams, we would require real-time image processing which in turn makes having extensive software resources quite essential. We propose to use parallelizing methods of Shared Memory through GPU Processing that uses techniques of grayscaling using the RGB values and Edge Detection calculation using the Robert's cross operation.

2 Methodology

We begin by establishing the input vital for the model. The implants would be able to sense the light images before processing them. So, to simulate the device, we incorporated the input as a set of real-life environment images and videos. We strategize to accept the inputs by using Python and constantly capturing the frames and converting them into a pixel image array of three dimensions. Further processing which includes parallelism, occurs using C++ Programming. Once the pixel array resolved in Python is received, after being converted to be compatible in C++, we call a function that parallelly converts the given pixel image array into grayscale using the following formula:

$$R := (0.2989 * r) + (0.5870 * g) + (0.1140 * b) \tag{1}$$

where r, g, and b represent the Red, Green and Blue values of an image represented in the three dimensional array previously received.

Moreover, after we receive the grayscale data, we need to apply edge detection, for which, we use Robert's cross operator (1) that calculates the presence of edges in the image by updating the pixel values as follows:

$$Si, j := \frac{1}{2}(|R_{i,j} - R_{i+1,j+1}| + |R_{i,j+1} - R_{i+1,j}|) \tag{2}$$

where R is the pixel array returned from the grayscale processing.

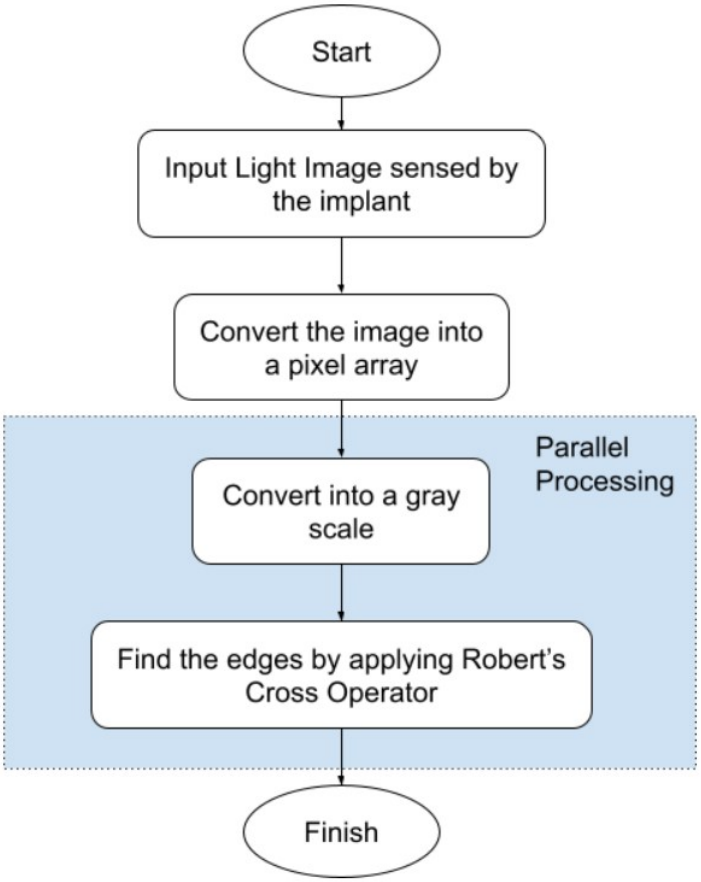

Figure 2: Model Workflow

Lastly, for more efficiency, we have also incorporated the improved Robert's Cross Operator formula which is given by:

$$y_{i,j} := \sqrt{x_{i,j}} \tag{3}$$

$$z_{i,j} := \sqrt{(y_{i,j} - y_{i+1,j+1})^2 + (y_{i,j+1} - y_{i+1,j})^2} \tag{4}$$

3 The Parallelism

We approached the application by first analysing the entire workflow (as shown in Figure 2) from a computational complexity point of view. From sensing the light images to sending a processed pulse stream to the optic nerve, we observed that the portion of the workflow that took the most time was the processing of the image pixel array. In a retinal implant, we cannot afford to have delays where real time responses are expected and required. Specifically, it was the grayscaling and edge detection that was exploiting the most amount of time. We thus incorporated parallelization to determine how much faster it would work when provided with an image input.

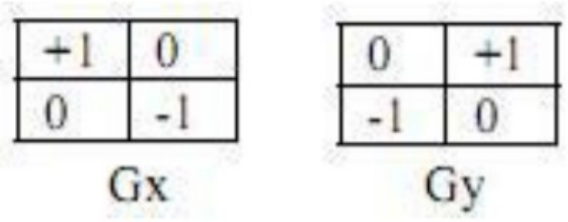

Figure 3: Robert's Cross Operator Mask (3)

Traditionally, for edge detection, there are four different gradient operators which are used to estimate the magnitude of the gradient of the test image. Namely, the Robert's cross operator, Sobel's operator, Scharr operator and Prewitt operator. We pursued the problem and went ahead with the Robert's cross operator instead of the more efficient algorithm like Sobel's as we wanted to have a clearer execution time distinction that parallelism could incorporate into the model. In edge detection using Robert's Cross Operator, the vertical and horizontal edges are brought out individually and then put together for resulting edge detection. The Roberts edge detector uses the mask as given in Figure 3 (3) to digitally approximate the first derivatives as differences between adjacent pixels.

Now, to implement the parallelism, we employed GPUs through programming in CUDA.

3.1 GPU Processing

Using the Compute Unified Device Architecture (CUDA), we can incorporate Graphic Processing Units (GPUs). In the past, GPU's have been used to execute only graphic applications, and implementing parallel processing algorithms on this platform was extremely challenging. Now, with all the innovations in parallel computing, we can utilize GPU processing through CUDA by writing and launching kernels that would be processed individually by threads in different blocks in the grids simultaneously.

For GPU processing in CUDA, we used 8 blocks and 1024 threads.

3.1.1 Initial Approach

We started off by creating a single kernel function which would execute the grayscaling and the Robert's Cross operator together. So, it would grayscale the pixels or elements in the pixel array needed for the Edge Detection, that includes the three adjacent pixels, and then run the Robert's Cross Operation on them. We realised that on following this method, we would need to update the grayscale values two times on the same pixel, which would clearly increase the time taken to process the video.

We also initially incorporated OpenMP and worked on parallelizing the model through OpenMP Shared Memory, but we were not getting the appropriate time results that a parallel process should ideally have. Thus, we went ahead with GPU Processing.

3.1.2 Final Approach

To implement the parallel model, we have thus used two kernels which execute the grayscale operation and the edge detection operation on the image array. First, the kernel to parallely process the pixel array and change it into a grayscale image is called. This kernel runs the grayscaling formula, given in formula (1),

Table 1: CUDA-Serial Processing Time Comparison

			Processing Time	
Video	Array Length	Video Length	Serial Time	CUDA Time
sample1.mp4	240 px	2 mins	4.3408 s	1.9415 s
sample2.mp4	1080 px	5 sec	7.9938 s	2.1795 s
sample3.mp4	720 px	23 sec	13.3107 s	3.5991 s

parallely on each pixel. The result of this kernel call will be a processed pixel array, which would be a Black and White image. Now, this resultant image is taken as input for the next kernel launch. This kernel runs the Robert's Cross Operator that incorporates the adjacent pixel values on the right and below as well as the diagonally adjacent bottom-right pixel. Subsequently, the resultant grayscale image is also passed through to the kernel executing the improved Robert Cross Operation given in Formula (3) and (4).

4 Observations

The simulation takes as input a mp4 file (representing what a human eye will see in some view). Then each frame is processed individually, both serially and parallely in order to make a comparison. We used CUDA programming as our parallel architecture. The results, shown in table 1, represent the time taken to process the entire video of varying resolutions and lengths. CUDA runs, on average, 4 times faster than the serial version. The time taken is calculated by summing the amount of time taken by each process, the processes being the parallel grayscaling and edge detection. Thus, the video times, given under Serial Time and CUDA Time, are the cumulative summation of each process after the next image from the video is selected.

5 Conclusion

In conclusion, we successfully reduced the latency in computing the pulse signal output stream of retinal implants using parallelization. The latency reduction was obtained by parallelizing the Grayscaling and Edge Detection in the working of the simulated device. We implemented this parallelization by using CUDA Programming. We observed that the CUDA speedup is significant when compared to the serial processing. This would help a lot in such retinal implants as the real-time processing needs to be instant and quick, with the least amount of error probability.

6 Future Scope

Improvements can be made into the following aspects:

- Reduction in power consumption in order to inhibit heat generation.
- Incorporate more sophisticated filters such as Sobel's Operator or Canny Edge Detection.
- Reducing Code footprint considering the limited space availability.

Figure 4: sample1.mp4 - Original, Robert Cross, Improved Robert Cross

Figure 5: sample2.mp4 - Original, Robert Cross, Improved Robert Cross

Figure 6: sample3.mp4 - Original, Robert Cross, Improved Robert Cross

References

[1] Robert's cross operator. https://en.wikipedia.org/wiki/Roberts_cross.

[2] C. L. Armin Alaghi and J. P. Hayes. Stochastic circuits for real-time image-processing applications. 2013.

[3] R. D. . D. G. Pinaki Pratim Acharjya. Study and comparison of different edge detectors for image segmentation. 2012.