A close up of a sign

Description automatically generated

PARALLEL COMPUTER ARCHITECTURE AND PROGRAMMING

FISAC REPORT ON

**GPU Accelerated Multidirectional Sobel Edge Detection using CUDA**

SUBMITTED TO

**Department of Computer Science & Engineering**

by

Moksha Deepak Kothari

210905017

Roll Number 4

VI Semester

CSE Section A

|  |  |  |
| --- | --- | --- |
| TABLE OF CONTENTS | | |
|  | | Page No |
| Chapter 1 | INTRODUCTION | 3 |
| Chapter 2 | LITERATURE REVIEW | 4 |
| Chapter 3 | METHODOLOGY | 5 |
| Chapter 4 | RESULTS AND DISCUSSION | 8 |
| Chapter 5 | CONCLUSIONS AND FUTURE ENHANCEMENTS | 9 |
| REFERENCES | | 10 |

**INTRODUCTION**

Edge detection is a vital image processing technique used to identify margins in an image, usually done by detecting significant changes in color or intensity. These margins usually correspond to the edges of objects or regions within the image. Edge detection plays a crucial role in fields like object detection and recognition, image segmentation, feature extraction, medical image analysis, and digital image processing amongst several others. Several edge detection algorithms have been developed, including Roberts Cross Edge Detector, Prewitt Edge Detector, Sobel Edge Detector and Canny Edge Detector.

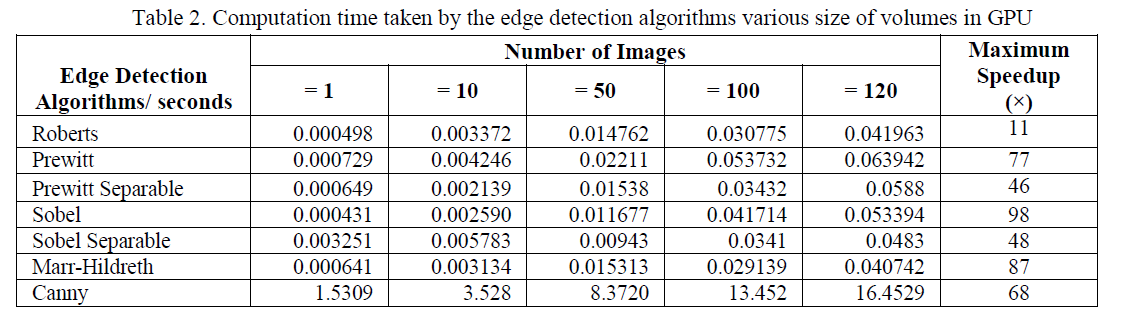
Amongst these algorithms, Sobel’s Edge Detector Algorithm stands out for its elegance and simplicity and its superior performance, particularly when parallelized. Traditionally, it employs the Sobel operator comprised of two 3x3 kernels to perform convolutions on an image to detect horizontal and vertical edges, also known as two-directional edge detection. The algorithm operates by computing the gradient magnitude at each pixel and setting the pixel value of the output image based on the gradient magnitude to obtain the edges in the image. The underlying assumption is that a sharp change in intensity (steep intensity gradient) indicates an edge. These magnitude values are compared against a threshold to control the level of detail the output image includes. Lower threshold values result in more detailed outputs while higher threshold values lead to cleaner images with fewer details. It is important to note that the algorithm works on greyscale images. In the case of a colored image, it is first converted to a greyscale image before applying the Sobel operators. The Sobel detector is also incredibly sensitive to noise in pictures, it effectively highlights them as edges.

This project extends the idea further by comparing the traditional two-directional edge detection to multidirectional edge detection in terms of details captured. Both versions of the algorithm have been parallelized using CUDA C. Multidirectional edge detection uses four 5x5 kernels to not only detect horizontal and vertical edges, but also diagonal edges. Since it detects edges along multiple directions, it captures finer details in more complex images that the 3x3 kernels may miss, leading to more comprehensive edge detection. It detects subtle changes in gradients better than the traditional kernels. In addition to this, it is less sensitive to noise since it aggregates data from multiple directions. On the other hand, it is a more complex technique that requires more memory and increases processing time.

In certain applications, the advantages of multidirectional edge detection outweigh its disadvantages, making it the preferred choice. For example, medical imaging and image analysis require fine irregular details such as tumors and fingerprints to be captured. Satellite image analysis also irregularly shaped objects and terrains may have to be outlined. industrial quality control, where complex objects have to be outlined to detect anomalies in shape.

**LITERATURE REVIEW**

Image processing is a fundamental task in image processing with several applications. Over the years, a plethora of algorithms have been developed, each with its own advantages and disadvantages, in an attempt to perform edge detection effectively. Some of these alternatives include Roberts Cross Edge Detector, Prewitt Edge Detector, Sobel Edge Detector, Marr-Hildreth Edge Detector, and Canny Edge Detector. Comparative studies have shown that the Sobel edge detection algorithm outperforms the others when parallelized on GPUs. Notably, it has been reported to work approximately 98 times faster on GPUs compared to CPU execution.

P. Sriramakrishnan et Al. (2018)

However, the conventional two-directional Sobel algorithm has limitations, notably its inability to detect diagonal edges. When we switch to the multidirectional variant of the algorithm, the increase in size and number of kernels leads to decreased speed. However, given the high efficiency of its two-directional counterpart, we can reasonably assume that the multidirectional version will also be sufficiently fast. We can improve accuracy further by using eight-directional convolution, however, this method uses eight 5x5 kernels, which leads to a much larger reduction in computational speed.

Most of these algorithms are designed to calculate the rate of change of pixel brightness, with the underlying assumption that a steep intensity gradient indicates an edge. We take the derivative of the intensity values to find the point where the derivative is maximum. We can use this idea to group filters of edge detection algorithms into two categories: Gradient and Laplacian. The gradient method involves calculating the maximum and minimum in the first derivative of the image. The Laplacian method involves identifying points where the second derivative of the image crosses zero.

The Sobel algorithm utilizes odd masks since they can be centered on a pixel to find an approximate based on its surroundings. We use these masks to calculate how rapidly intensity changes around a pixel, and hence, identify edges.

When parallelizing the code, it is important to consider the number of kernels used. A common assumption would be splitting the task into smaller pieces and creating kernel functions for each task. However, this also increases the amount of communication and data transfer between the host and devices, which in turn increases the time required for execution, rather than decreasing it. Comparative studies show, that launching three kernels, instead of two, fetching the red, green, and blue components of the colored image, conversion into greyscale, and applying Sobel’s mask for gradient calculations, reduces the speedup by 52% (Adhir Jain et Al. 2016). In addition, speedup increases as image resolution increases only up to a certain extent, after which speedup starts decreasing due to an increase in communication time between the CPU and GPU

**METHODOLOGY**

This section outlines the approach used to parallelize Sobel’s algorithm and compare the two versions of the algorithm: the traditional two-directional approach Sobel algorithm and the enhanced multi-directional Sobel algorithm.

In the two directional Sobel algorithm, the horizontal gradient (Gx) and the vertical gradient (Gy) are computed at each pixel to capture the change in intensity. A high positive Gx indicates a significant increase in intensity from left to right while a high negative Gx indicates a significant decrease in intensity from left to right. Similarly, a high positive Gy indicates a significant increase in intensity from top to bottom, while a high negative Gy indicates a significant decrease in intensity from top to bottom. This is achieved by convoluting the image with the corresponding kernels in the Sobel operator.

Next, Gx and Gy  are combined to get the gradient magnitude using the equation

However, This only considers the horizontal (0o) and vertical (90o) directions.

In the multidirectional edge detection version of the Sobel algorithm, the image is convoluted with 4 kernels to consider the diagonal gradients too.

The four gradient values to obtain the gradient magnitude using the equation

The introduction of Gd and Gdt allows us to consider the 45o and 135o directions in our gradient magnitude calculation.

Following the gradient magnitude calculation, a user-defined threshold (T) is applied to the gradient magnitude, G, to determine the edge pixels.

This sets the pixels at the edges of objects in the image to white (255), and all other pixels to black (0), highlighting the edges of the image, allowing for subsequent analysis.

To accelerate computation, the process is parallelized using CUDA C. Each pixel in the output image is computed concurrently, leveraging the parallel processing capabilities of GPU threads. This significantly reduces computation time which is particularly useful for large image datasets.

**CUDA Kernel for two-directional Sobel algorithm**

\_\_global\_\_ void sobel(unsigned char \*inputImage, unsigned char \*output, int width, int height, float \*gradientMagnitude) {

int x = blockIdx.x \* blockDim.x + threadIdx.x;

int y = blockIdx.y \* blockDim.y + threadIdx.y;

if (x < width && y < height) {

int sobel\_x[3][3] = { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 } };

int sobel\_y[3][3] = { { -1, -2, -1 }, { 0, 0, 0 }, { 1, 2, 1 } };

float gradient\_x = 0.0;

float gradient\_y=0.0;

for (int i = -1; i <= 1; i++) { //convolution operations

for (int j = -1; j <= 1; j++) {

if(x+i>=0 && x+i<width && y+j>=0 && y+j<height){

gradient\_x += sobel\_x[i + 1][j + 1] \* inputImage[(y + i) \* width + (x + j)];

gradient\_y += sobel\_y[i + 1][j + 1] \* inputImage[(y + i) \* width + (x + j)];

}

}

}

// Compute gradient magnitude

gradientMagnitude[y \* width + x] = sqrtf(gradient\_x \* gradient\_x + gradient\_y \* gradient\_y);

}

if(gradientMagnitude[y \* width + x] >100){ //adjust threshold value as required

output[y\*width+x]=255;

}

else{

output[y\*width+x]=0;

}

}

**CUDA Kernel for multidirectional Sobel algorithm**

\_\_global\_\_ void sobel(unsigned char \*inputImage, unsigned char \*output, int width, int height, float \*gradientMagnitude) {

int x = blockIdx.x \* blockDim.x + threadIdx.x;

int y = blockIdx.y \* blockDim.y + threadIdx.y;

if (x < width && y < height) {

int sobel\_x[5][5] = { {-1,-2,0,2,1},{-4,-8,0,8,4},{-16,-12,0,12,16},{-4,-8,0,8,4},{-1,-2,0,2,1}};

int sobel\_y[5][5] = { {-1,-4,-6,-4,-1},{-2,-8,-12,-8,-2},{0,0,0,0,0},{2,8,12,8,2},{1,4,6,4,1}};

int sobel\_d[5][5] = { {-6,-4,-1,-2,0},{-4,-12,-8,0,2},{-1,-8,0,8,1},{-2,0,8,12,4},{0,2,1,4,6}};

int sobel\_dt[5][5] = { {0,-2,-1,-4,-6},{2,0,-8,-12,-4},{1,8,0,-8,-1},{4,12,8,0,-2},{6,4,1,2,0}};

float gradient\_x = 0.0;

float gradient\_y = 0.0;

float gradient\_d = 0.0;

float gradient\_dt = 0.0;

for (int i = -2; i <= 2; i++) { //convolution operations

for (int j = -2; j <= 2; j++) {

if(x+i>=0 && x+i<width && y+j>=0 && y+j<height){

gradient\_x += sobel\_x[i + 2][j + 2] \* inputImage[(y + i) \* width + (x + j)]; // Fixed indexing

gradient\_y += sobel\_y[i + 2][j + 2] \* inputImage[(y + i) \* width + (x + j)]; // Fixed indexing

gradient\_d += sobel\_d[i + 2][j + 2] \* inputImage[(y + i) \* width + (x + j)]; // Fixed indexing

gradient\_dt += sobel\_dt[i + 2][j + 2] \* inputImage[(y + i) \* width + (x + j)]; // Fixed indexing

}

}

}

// Compute gradient magnitude

gradientMagnitude[y \* width + x] = sqrtf((gradient\_x \* gradient\_x) + (gradient\_y \* gradient\_y) + (gradient\_d \* gradient\_d) + (gradient\_dt \* gradient\_dt));

if(gradientMagnitude[y \* width + x] > 2500){ //adjust threshold value as required

output[y\*width+x]=255;

}

else{

output[y\*width+x]=0;

}

}

}

**RESULTS AND DISCUSSION**

When run on the same image, the two algorithms produce varying results.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  | threshold 100 | threshold 2200 |
| 2 |  | threshold 100 | threshold 2500 |
| 3 |  | threshold 100 | threshold 2200 |
| 4 |  | threshold 200 | threshold 3000 |
|  | (a) | (b) | (c) |

Results. (a) Original Image (b) Two-directional Sobel’s algorithm (c) Multidirectional Sobel's algorithm

As expected, the multidirectional Sobel algorithm demonstrates a clear advantage in terms of quality of edge detection. As observed in the first image, the two-directional Sobel algorithm cannot detect diagonal edges effectively. It is not able to capture the smooth waves in the image and introduces distortions. On the other hand, the multidirectional Sobel algorithm captures the edges in the image more effectively, resulting in a more accurate representation. It does not have disruptions and discontinuity in the diagonal lines. The second and third images demonstrate how the results of the traditional algorithm produce relatively more noisy images compared to the enhanced algorithm. Noise can be reduced by increasing the threshold; however, it would also lead to a loss in the details produced. On the contrary, the enhanced algorithm seems to achieve a better balance between noise reduction and edge preservation. This fourth image further exemplifies this observation. The original image is particularly noisy, as a result of which, the image produced by the traditional algorithm is also very noisy. The results of the enhanced algorithm, albeit containing noise, are much sharper and cleaner than its two-directional counterpart.

It is important to note the significant difference between the threshold values used in the two versions of the algorithm. Since the enhanced algorithm uses bigger and more convolution kernels in number compared to the traditional algorithm, with greater values, which are then added up and squared, during gradient magnitude calculations, the resulting gradient magnitude values are much greater than those computed in the traditional algorithm. This requires the threshold values to be greater to produce the required output.

**CONCLUSIONS AND FUTURE ENHANCEMENTS**

In conclusion, the comparison between the two-directional Sobel algorithm and the multidirectional Sobel algorithm highlights the advantages of the latter in terms of noise reduction and better precision at edge detection albeit at a slower processing speed. The multidirectional algorithm uses multiple convolution kernels to capture diagonal gradients in addition to horizontal and vertical gradients, enabling it to capture intricate details in more complex images. This is useful in scenarios where accuracy is more important.

Furthermore, the results emphasize the importance of selecting an appropriate threshold to seek a balance between noise reduction and detail preservation. While the latter does a better job at reducing noise without compromising on details, further fine-tuning threshold values could optimize performance for larger datasets. Looking ahead, machine learning techniques can be integrated to obtain optimal threshold values instead of the inefficient manual trial-and-error method.

Additionally, future enhancements can explore and parallelize noise reduction techniques to further increase accuracy. Deep learning based denoising models would enhance the adaptability of the edge detection algorithm across a larger, and more diverse dataset.

In summary, the field of image processing and edge detection is characterized by continuous research and enhancements to address the upcoming real-world challenges. By utilizing techniques from multiple disciplines, researchers can discover innovative and robust techniques to pave the way for new applications in multiple fields ranging from healthcare to technology and more.

**REFERENCES**

1. Vincent, O. R., & Folorunso, O. (2009). Descriptive Algorithm for Sobel Image Edge Detection. In Proceedings of Informing Science & IT Education Conference (InSITE) 2009.
2. Chang, Q., Li, X., Li, Y., & Miyazaki, J. (Year). "Multi-directional Sobel operator kernel on GPUs." arXiv preprint arXiv:2305.00515 [cs.DC].
3. Sriramakrishnan, P., Kalaiselvi, T., & Somasundaram, K. (2018). "Parallel Processing Edge Detection Methods for MR Imagery Volumes using CUDA Enabled GPU Machine." International Journal of Computer Sciences and Engineering, Vol-6, Special Issue-4, May 2018, E-ISSN: 2347-2693. [Online]. Available: [www.ijcseonline.org](http://www.ijcseonline.org)
4. Hao, F., Xu, D., Chen, D. et al. Sobel operator enhancement based on eight-directional convolution and entropy. Int. j. inf. tecnol. 13, 1823–1828 (2021). <https://doi.org/10.1007/s41870-021-00770-3>
5. A. Jain, A. Namdev, and M. Chawla, "Parallel edge detection by SOBEL algorithm using CUDA C," 2016 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS), Bhopal, India, 2016, pp. 1-6, doi: 10.1109/SCEECS.2016.7509360.

keywords: {Graphics processing units;Image edge detection;Kernel;Instruction sets;MATLAB;Parallel processing;Parallel Edge Detection;Parallel Computing;CUDA;Kernel;Sobel Operator;Speedup;Image Processing;MATLAB;Parallelism v/s Speedup},