Edge detection using Sobel filter on hybrid parallel architecture

Project Report

UNIVERSITY OF MANITOBA

APRIL 30, 2022

SUBMITTED BY:
Amandeep Singh

Abstract

This project introduces parallelism at the process level using the hybrid parallel architecture of Message Passing Interface and OpenMP, to implement computer vision edge-detection task using Sobel–Feldman operator on videos. This model successfully shows the implementation and design of parallel architecture and represents the significance of parallelism when the input in terms of frames and parallel architecture in terms of thread and process increase, the problem is parallelized. The outcome of this study shows that the execution time of the algorithm decreased by enhancing parallelism.

Introduction

Image processing is referred to as steps or processes used on the images collected by camera or sensors such that they can be used in different types of computer vision tasks some of them related to applications in medical imaging, face recognition, etc. [1]. The information produced by images is stored in the pixels and usually, we want to find the patterns and some specific values which can be crucial in several applications [1] like computing the gradient of pixels for edge detection in images.

Figure 1:Left image: Orignal image, Right side image: result image after applying the Sobel filter for edge detection

We proposed the algorithm design based on the parallel architecture to parallelize the edge detection which is a hot topic in research and has been applied in a variety of applications, particularly in medical applications. The technique of recognizing and locating sharp discontinuities in an image is known as edge detection[2], and the goal of edge detection, in

general, is to significantly reduce the quantity of data in an image while retaining the structural qualities that may be utilized for future image processing.

This project studies the implementation of a Sobel-Feldman operator using parallel processes and threads. It is well studied and widely implemented for problems in many computer vision tasks called edge detection. The term "edge detection" applies to a variety of mathematical methods that aim to identify pixels in an image where the brightness changes sharply with respect to its neighboring pixels. Edge detection is a fundamental tool in computer vision and image processing used for segmentation and registration/alignment. We use the Sobel edge detection algorithm [4] to generate edge maps, but the methods in this report apply to some other edge detection algorithms as well. The Sobel method convolves the original image by two 3x3 kernels to approximate the brightness derivatives in the horizontal and vertical directions.

Problem Statement

The Sobel operator measures a 2-D spatial gradient on an image and looks for high-spatial-gradient areas that correspond to edges [3]. It's most commonly used to calculate the estimated absolute gradient magnitude at each point in a grayscale image[3]. Figure 2 shows the operator, which is made up of a pair of 3x3 convolution masks. The second mask is simply rotated by 90 degrees [3]. As shown in Figure 2 G_x kernel is used to find edges in the horizontal direction, whereas the G_y kernel is used to detect edges in the vertical direction [3]. The magnitude of the gradient that will determine the edges in both directions will be used to compute the final output. The magnitude of the gradient is calculated using the formula 1[3]:

-1	0	+1
-2	0	+2
-1	0	+1

Kernel 1

+1	+2	+1
0	0	0
-1	-2	-1

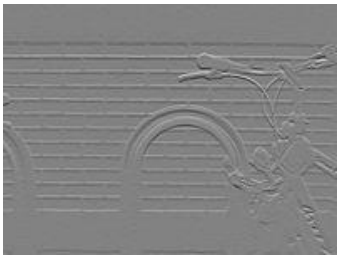
Kernel 2

Figure 2: Sobel Feldman convolution kernels (masks).

$$|G| = \sqrt{(G_x^2 + G_y^2)}$$

Formula 1: Magnitude of Gradient |G|.

(a) Input image (b) Resulting image after computing gradient using Formula 1



(c) Vertical Gradient (G_y) on the input image (d) Horizontal Gradient (G_x) on the input image

Figure 3: Sobel Feldman algorithm result image and computed gradients

The edge detection task is usually approached as a sequential problem but it can be divided into primitive tasks so that it can be approached as a distributed problem. This project emphasized the distributed aspect of the problem because as the input size of the video increases the execution algorithm time increases too and that's not efficient. As the video frames per second are increasing in size and resolution the sequential algorithm becomes inefficient. Hence the need for optimizing this problem arises. To solve this problem, we proposed a hybrid parallel architecture on the solution that is parallelized using Message Passing Interface (MPI) and OpenMP on our University's supercomputer called Mercury.

Related Work

There is a limited amount of efforts have been directed towards parallelizing the edge detection of images using the MPI approach, to achieve this process with minimum computational time using different architectures, techniques, and methods.

The Sobel-Feldman operator Edge detection mathematical method [4] uses the architecture using

two kernels convolved with the original image for horizontal and vertical changes. Analyzing its architecture the parallelization of convolution looks promising. Ziou et. al.[5] has published a detailed survey on various techniques for edge detection and understanding various architectures. After analyzing different algorithms Sobel operator method seems the best fit for parallelizing and it is also insensitive to noise in the image. Sanduja et. al. [6] present the approach of implementing the Sobel-Feldman operator on parallel architecture using FPGA architecture creating a very interesting solution that is less computation-intensive and parallelizable. Nan Zhang et. al.[7] presents an interesting and detailed approach to parallelizing the architecture of the Sobel operator to execute the edge detection in the parallel architecture of GPU and their performance results show that the parallelized approach uses proportionally less computation time in comparison to CPU. Abdul et. al. [8] perform an analysis of the Sobel edge detection method by implementing it in parallel architecture and computing its efficiency on various parameters. They concluded that with more data as input multi-core architecture performed way better than the traditional one.

Design and Solution Methodology

For this project, we designed the algorithm using both MPI and OpenMP to create a hybrid architecture. It partitions the video into n small videos. Each of the small video frames is extracted. These n videos are divided among p processors. Each processor works on computing the convolution of the frame in parallel threads. In the last step, the computed frames are again joined to form the resulting video with an edge detection mask applied. In this section, we discuss the design and implementation of the algorithm proposed in detail.

Following PCAM design methodology we divided the task of building a parallel algorithm into Decomposition, Communication, Agglomeration, and Mapping.

Decomposition

The first step in designing the parallel algorithm is to define primitive tasks. The main goal of decomposition is to identify as many primitive tasks as possible because it determines the upper bound on the parallelism we can exploit. In this algorithm, the primitive task was defined as to calculate the gradient using the 2 Sobel convolution kernels as defined in formula 1 for each pixel in the image frame.

Communication

This defines how the partitioned data and processing of data communicates among processors or threads. We formulated the decomposition in such a way that the communication overhead is minimum to speed up the parallelized version. In the defined algorithm communication is only required among the pixels of the frame as the different frames are not dependent on each other for implementing the Sobel filter convolution. So, there is communication among threads as they share the pixel values of the last row with the first row of another thread.

Agglomeration

In this phase, the task and communication structures defined in the first two stages of an algorithm design are evaluated for performance requirements and implementation costs. As it can be observed from algorithm 1, the load distribution is done with respect to the rank of the process. The determination of how to parallelize the code, and which computation is to be done how is determined in this phase.

We have implemented hybrid parallelism using both process-level parallelisms using MPI and thread-level parallelism using OpenMP. We tried implementing the task-level parallelism too at the gradient computation but due to the dependency of various elements of the task on each other, I failed to achieve that. Even with the high-resolution videos, the hybrid approach performed way better than any single one. In this way, we can reduce the idle time for processors and compute the results faster. The input video is divided into n subsections based on the number of processes. Each process works on all frames for that subsection dividing it further into threads.

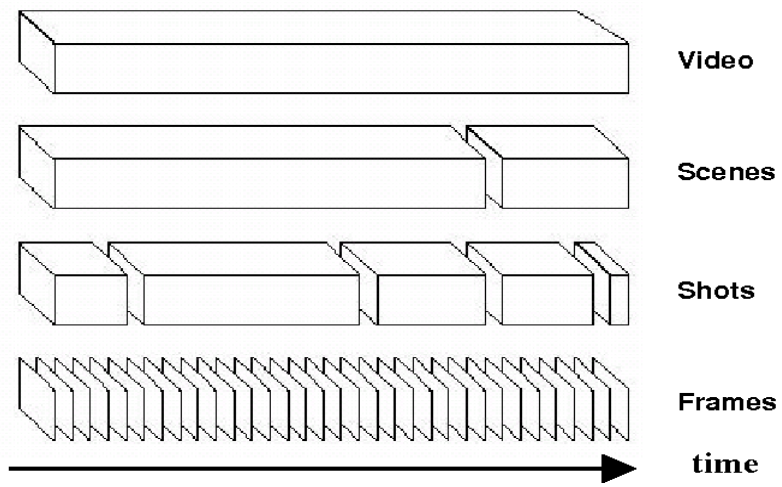


Figure 4: Extraction of frames from video

Mapping

Each task is assigned to a processor in a manner that attempts to satisfy the competing goals of maximizing processor utilization and minimizing communication costs. This can be specified statically or determined at runtime by load-balancing algorithms [9]. The root process defines the frames for the video and divides the frames among all processors. Each processor uniquely works on assigned video frames on parallel threads to compute the gradient. Finally, process 0 combines all gradients to make a convolved video using the Sobel Feldman operator.

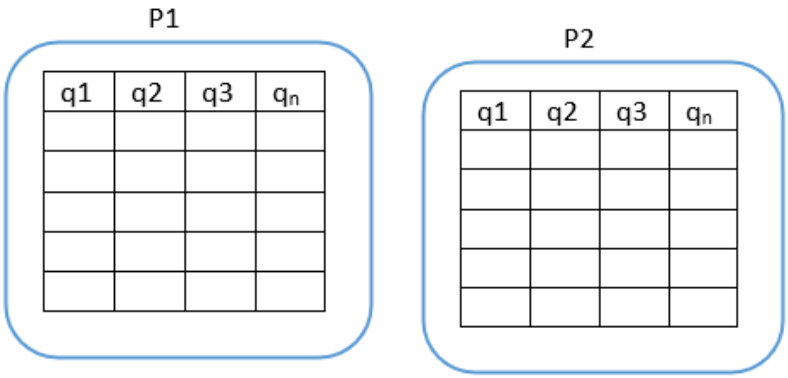


Figure 5: Thread level Architecture for each Processor (n rows of each frame divided among q threads)

Algorithm 1 Sobel Feldman operator on video using hybrid parallel architecture

```
Extract frames from the video
If (root process) then
    Number of frames < --- extracted frame
    Initialize parallel processes
    Distribute workload of frames among n processes
For each (process) in parallel do
    Divide the image frame based on rows
    For each thread in parallel do
        Compute the gradient x and gradient y for pixels assigned
        Compute the derivative of the image pixel
    Processed pixel values merged back
If (root process) then
    Save the image frames
    Create resulting video from frames
```

Algorithm 1: Sobel Feldman filter on video using hybrid parallel architecture

Empirical analysis

For evaluating the performance of this model, the variations in execution time and speedup are studied. Speedup is a way of relating the performance of a parallel program, with its sequential implementation. The execution time of solving a problem sequentially is divided by the execution time of solving the same problem, with the same size when implemented parallelly. This speedup can be further classified as Relative Speedup and Absolute Speedup.

Relative Speedup = (Time to solve the problem using a single processor) /

(Time to solve the problem, Q using P processors)

Absolute Speedup = (Time to solve the problem using the best approach on the fastest single processor)/ (Time to solve the problem using P processors)

Parallel Efficiency = 1/P x speedup= (1/P) x(T_s/T_p)

The relative speedup is when the sequential program is executed on the same parallel computer, where the program is executed parallelly as well. Whereas, the absolute speedup means to calculate the execution time for sequential, using the best-found sequential approach and implementing it on the fastest single processor. The speedup of my proposed model is discussed in the evaluation section in the next section. We also study the Parallel efficiency which measures how much use of the parallel processors we are making.

Similarly, the empirical evaluation shows that apart from better results, excessive parallelism for smaller inputs didn't have a bigger impact on the performance. In table 1, one can observe that when the number of threads is 8 and 16 the performance in terms of the execution time was bad whereas the performance at threads set 4 was better than 8 and 16. This shows that for parallelism the optimal number of threads is based on how the image frame is divided among q threads

There was a significant improvement in the execution time of the algorithm when we increased the number of processes. When we parallelize on 2 processors execution time is reduced to half of that with 1 process. As we increased the number of processes the execution time decreased significantly. To see the effect of thread parallelization we ran various settings of thread numbers for each number of processes settings. As the thread number increased we see a slight improvement in the execution time for the different number of process settings we

noticed that there is a different optimal number of threads for the different number of processes for which hybrid architecture has the least execution time. For example, when the number of processes was 4 we noticed that the optimal number of threads was 4 which gives the least execution time among the various number of thread settings for 4 processes.

Hybrid Parallelization on Video		
Processors	Threads	Execution Time (secs)
1	1	21.610545
1	4	20.131118
1	8	21.229865
1	16	20.914715
1	32	20.035563
2	1	10.753076
2	4	10.091734
2	8	9.863271
2	16	10.062005
2	32	10.131883
4	1	5.249451
4	4	5.029994
4	8	5.447537
4	16	5.184673
4	32	5.125454
16	1	1.318154
16	4	1.505105
16	8	1.35447
16	16	1.500169
16	32	1.187042
32	1	0.606979
32	4	0.477799
32	8	0.581818
32	16	0.529752
32	32	0.542428
64	1	0.334877
64	4	0.305879
64	8	0.348313
64	16	0.309057
64	32	0.334454

(a) Execution time on different threads and process settings

Sequential Execution time in seconds	28.58476
Parallel execution time in seconds	5.249451
Absolute speedup (achieved superlinear speedup)	5.445285
Parallel Efficiency	1.361321

(b). Speedup and Efficiency on 4 processes

Thread parallelization on Image		
Processors	Threads	Execution Time (secs)
1	1	0.015416
1	2	0.008801
1	4	0.004571
1	8	0.002666
1	16	0.001715
1	32	0.001903

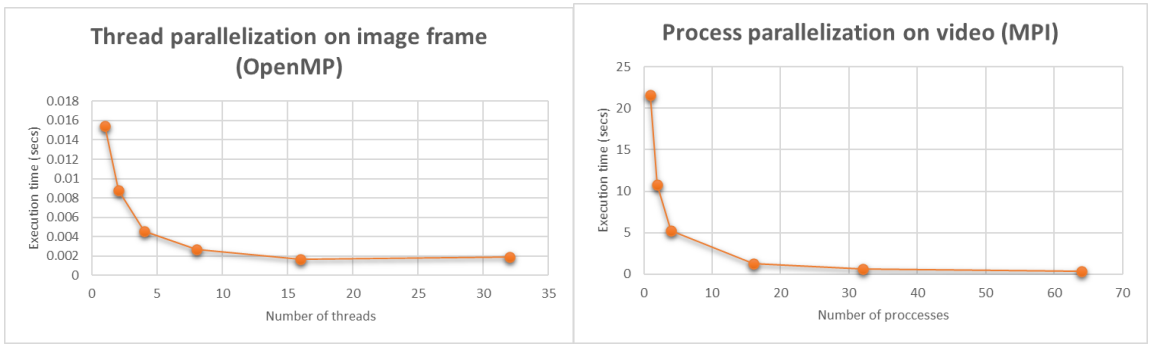
(c). Execution time for 1 process and different threads settings

MPI parallelization on Video		
Threads	Processors	Execution Time (secs)
1	1	21.610545
1	2	10.753076
1	4	5.249451
1	16	1.318154
1	32	0.606979
1	64	0.334877

(d). Execution time for 1 thread and different number of processes

Table 1: Performance results for the various number of threads and processes on hybrid parallel architecture.

To understand the effect of both MPI and OpenMP on the execution times we divided the problem to do a more detailed analysis of different parts of the architecture. First, we analyze the effect of thread parallelization on a one image computation we noticed a significant improvement in the execution time as the number of threads increased but when we reached threads = 32 the execution time increased which showed that the optimal number of threads for this frame was 16. Similarly, we dived into the processes parallelization we can infer from table 1 that as the number of processes increased and the execution reduced. Overall, we are able to reduce the execution time for our input from 21.6 secs to 0.33 secs which is a major improvement over the traditional algorithm.



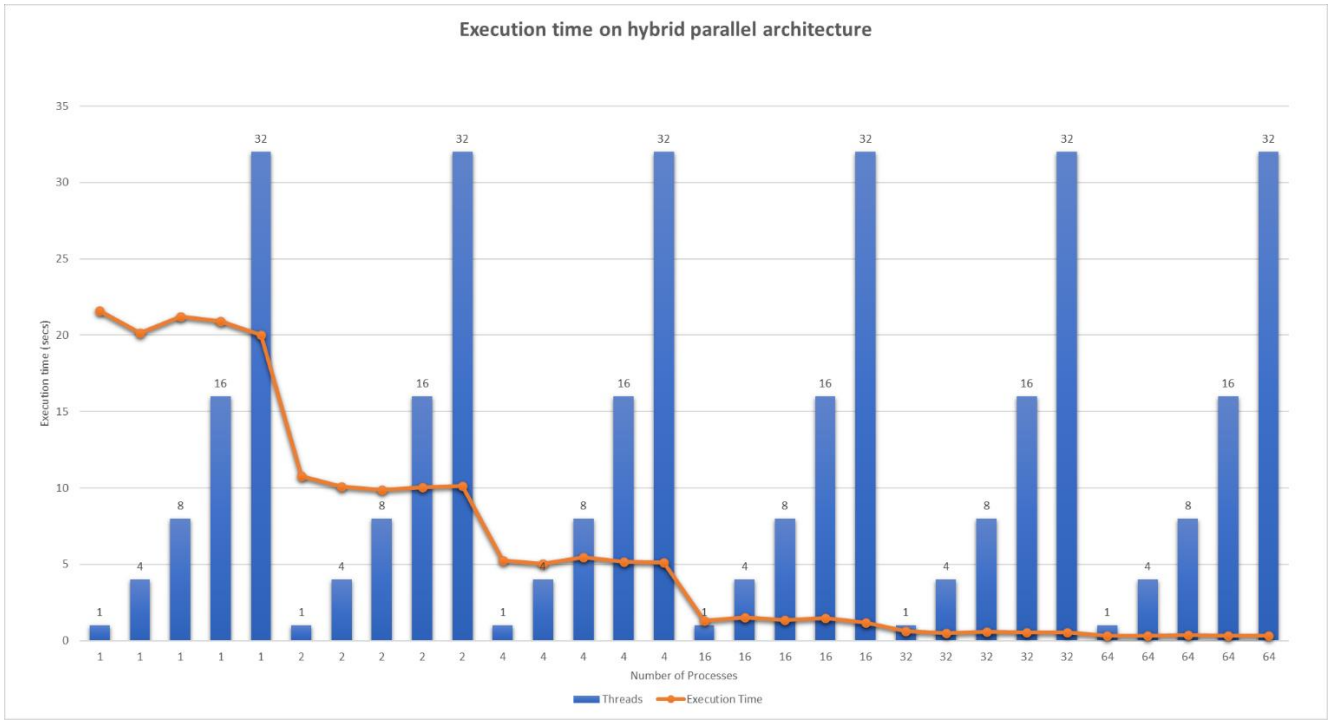


Figure 6: Represents the execution time for various settings of threads and processes for 3 settings (i) Thread parallelization only (ii) Process parallelization only (iii) Hybrid Parallelization

Overall, as computer vision tasks our computation and data-intensive, our asymptotic evaluation shows that we

are able to achieve superliner speedup by dividing the problem into subproblems and as we increased the number of processes our execution time dropped significantly approximately to half than previous processes size. We are able to achieve very good parallelization using our algorithm as seen in table 1 we are able to decrease the execution time of the algorithm from 21.6 seconds to 0.33 seconds for 64 processes and 32 threads for each process.

Conclusion

This work proposed a new algorithm that applies the Sobel Feldman operator to the input video. The algorithm is developed on hybrid parallel architecture which uses the n threads on each process among n processes. We are able to achieve very good and significant results on the performance of our algorithm without compromising the output. The algorithm developed can handle any quality of videos from low resolution to 8K resolution of videos. In future work as computer vision tasks are very data and computation-intensive this algorithm can be adapted to various other tasks in computer vision to achieve better performance.

References

[1] R. C. Gonzalez and R. E. Woods, Digital Image Processing. Pearson Prentice Hall, third ed., 2008.

[2] R. Maini and H. Aggarwal, “Study and comparison of various image edge detection techniques,” International Journal of Image Processing, 2009.

[3] W. Gao, L. Yang, X. Zhang, and H. Liu, \An im- proved sobel edge detection," Computer Science and Information Technology (ICCSIT),3rd IEEE International Conference, 2010.

[4] Sobel, Irwin & Feldman, Gary. (2015). An Isotropic 3x3 Image Gradient Operator. 10.13140/RG.2.1.1912.4965.

[5] Ziou, Djemel, and Salvatore Tabbone. "Edge detection techniques-an overview." Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii 8 (1998): 537-559.

[6] Sanduja, Varun, and Rajeev Patial. "Sobel edge detection using parallel architecture based on FPGA." International Journal of Applied Information Systems 3.4 (2012): 20-24.

[7] Nan Zhang, Yun-shan Chen and Jian-li Wang, "Image parallel processing based on GPU," 2010 2nd International Conference on Advanced Computer Control, 2010, pp. 367-370, doi: 10.1109/ICACC.2010.5486836.

[8] Abdul Khalid, Noor Elaiza & Ahmad, S.A. & Noor, N.M. & Fadzil, Ahmad & Taib, Mohd Nasir. (2011). Parallel approach of sobel edge detector on multicore platform. 5. 236-244.

[9] Foster, Ian. “Designing Parallel Algorithms: Part 1.” Dr. Dobb's, www.drdobbs.com/parallel/designing-parallel-algorithms-part-1/223100878.

Project files

Project files are available on the GitHub repository: https://github.com/akaler55/Parallel-Computing/tree/main/COMP7850_Sobel_edge_detection_MPI_OpenMP