Parallelization of Sobel Edge Detection Algorithm

Haneen Tartory
Master of Informatics
Palestine Polytechnic University
Hebron, Palestine
haneen@student.ppu.edu

Mohammed ALDasht
Assistant professor at IT department
Palestine Polytechnic University
Hebron, Palestine
mohammed@ppu.edu

May 23, 2012

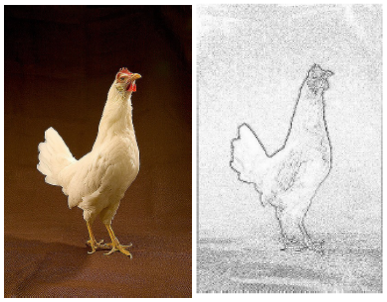
Abstract

The size of images in image processing considered a critical point in processing the images, so we must process the large size of images in small time especially in medical applications.In this paper, we present a new design of parallelizing Sobel edge detection algorithm in order to decrease the computation time. The parallel algorithm is implemented using MPI library and decentralized architecture. The new methodology depending on domain and function decomposition to improve the results. Experimental is done in one machine and the results demonstrate that the new design gives a good results.

1 Introduction

Image processing is a technique to enhance the images received from camera or a sensor, where image processing is used in different applications such as: medical imaging, remote sensing, document processing, graphic art, and others[1]. The output of image processing is information that is critical in some applications[1].

The size of images in image processing considered a critical point in processing the images, so we must find methods to process the large size of images in small time. Recently, some studies focused on using parallel algorithms for image processing.



(a) Before (b) After

Figure 1: Image before and after edge detection[3]

In this paper, we proposing a design to parallelize the edge detection which is an active research filed and has used in many applications in different areas, especially in medical applications. Edge detection refers to the process of identifying and locating sharp discontinuities in an image[2], where the purpose of edge detection in general is to significantly reduce the amount of data in an image, while preserving the structural properties to be used for further image processing[2], see Figure 1.

To detect the edge of image we need to apply specific computation on each pixel in the image, and that process will take a large time to do it especially for large size images, so in this paper we propose a parallel algorithm for edge detection, in order to reduce the computation time of the edge

detection operation.So in this paper we propose a new approach to parallelize the sobel edge detection algorithm depending on domain and function decomposition and decentralization architecture of processors.

This paper is organized as follows: Section 2 gives a brief introduction on the Sobel edge detection algorithm, while section 3 gives introduction of parallel computing using massage passing interface and describes the architecture of the new algorithm, section 4 presents the related works of parallelization the sobel edge detection algorithm, section 5 describes the parallel design, section 6 discusses the experiment and results and finally, section 7 outlines some conclusions.

2 Sobel Edge Detection

In this paper we choose the sobel edge detection algorithm, because it is the most common, and that refers to two reasons: it is less sensitivity to noise, and because it is the differential of two rows or two columns, so the elements of the edge on both sides has been enhanced, so that the edge seems thick and bright[4].

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial gradient that correspond to edges [4]. Typically it is used to find the approximate absolute gradient magnitude at each point in an input greyscale image[4].

The operator consists of a pair of 3x3 convolution masks as shown in Figure 2. One mask is simply the other rotated by angle 90 [4]. The Gx mask is

-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 convolution masks (kernels)

used to determine the edges in the horizontal direction while the Gy mask is to identify the edges in the vertical direction [4]. The resulting output will be calculated from the magnitude of the gradient that will determine the edges in both directions. The magnitude of the gradient are calculated using the formula [4]:

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

Although typically, an approximate magnitude is computed using[4]:

$$|G| = |G_x| + |G_y|$$

which is much faster to compute.

Based on the above equation, it seem that edge detection is trivial task, but it still adds to the overall computation time if it were to be operated on big size images because it involves huge number of iterations. Therefore finding means to accelerate the process can contribute to faster overall image processing time.

3 Parallel Computing Using Massage Passing Interface

A single computer processor can only do one thing at a time in sequential order. A parallel computer, which can perform multiple computations at once, can be used to solve simple problems in a minutes that normally take hours or days on a single processor [3]. The most basic concept behind parallel computing is the distribution of the workload between the individual processors that are working together to perform computations.
There are two approaches to writing parallel programs, these approaches are [5]:

1. Use of a directives-based data-parallel language.
2. Explicit message passing via library calls from standard programming languages.

In this paper we have used the Message Passing Interface (MPI) which is a message passing library interface specification due to the nature of our problem in order to distribute data between processors

and allow for inter processor communication, so we can work on large size of image without being restricted by the size of a global shared memory pool. MPI is designed for high performance computing on parallel machines or cluster of workstations [6], and it consist of multiple instances of a serial program that communicate by library calls. These calls divide into four classes [7]:

1. Calls used to initialize, manage, and finally terminate communications.
2. Calls used to communicate between pairs of processors.
3. Calls that perform communications operations among groups of processors.
4. Calls used to create data types.

Also, in this paper we depended on decentralization architecture for processors to achieve message passing approach, where the one of processor work as master and others as workers, but also there are a communications between the workers this communication is based on message passing approach, see Figure 3

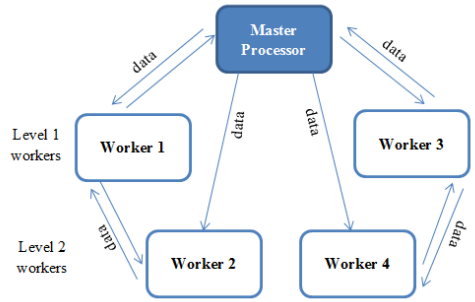


Figure 3: Decentralization architecture

4 Related Works

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

In their work, Haron et al. (2009)[6] designed a parallel Sobel edge detection algorithm using Foster's methodology; the methodology consists of four steps: partitioning, communication, agglomeration and mapping. In the partitioning stage, they have chosen data decomposition approach where they break the image into smaller pieces and each piece is associated with one task, they have identified that only local communication exists in this parallel algorithm, since the task requires values from only its neighboring pixels. In Mapping they assigned tasks to each processor, they have assigned each agglomerated task to each processor [6]. They reached to good results compared to sequential algorithm.

In other hand, Abdul Khalid, et al. (2011) [8], presented the parallel multicore Sobel edge algorithm which parallelizes the traditional sequential Sobel edge detection algorithm on a parallel multicore platform. They used the MPI where the algorithm is implemented on various thread [8], and they depended on data decomposition, and they reached to good method not only for image processing by depending on threads.

5 Design and Implementation

In this section we describes the design and implementation phases for the methodology used in this paper.

5.1 Decomposition

The first step in designing a parallel program is to break the problem into discrete "chunks" of work that can be distributed to multiple tasks [9]. There are two options on how partitioning can be done; domain or functional decomposition. 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 domain decomposition the data associated with a problem is decomposed, and each parallel task then works on a portion of the data. Functional decomposition focus on the computation that is to be

performed rather than on the data manipulated by the computation, the problem is decomposed according to the work that must be done, each task then performs a portion of the overall work [9]. In this paper we have used these two options; Domain and functional decomposition.

- **Domain Decomposition:** Due to the nature of edge detection mechanism where the computation is carried row by row in an image, we have chosen the domain decomposition by breaking the image into smaller element and calculating the gradient for each pixel in the element, so the domain decomposition consider the main decomposition in this problem. The element in this paper are rows of images, each level 1 workers and it's corresponding level 2 workers work on different rows of image, see Figure 4. This decomposition in this paper is done by dividing the height of the image on the number of level 1 workers, and the master works remaining rows if there exists.

Figure 4: Example of decomposition for the image where the number of worker processors equals 8

  For example, if we have 4 worker processor (2 workers are level 1, and 2 workers are level 2), each level 1 workers will work on different rows, where each level 1 worker and it's corresponding level 2 worker work on the same rows. and for each rows the level1 worker and it's corresponding level 2 worker will compute the gradient of image for both directions x and y by applying the previous kernels for each pixel in each rows.

- **Function Decomposition:** In edge detection algorithm, for each pixel we find the gradient for each direction x and y. In this paper we have decomposed the compute of gradient into two processor each processor compute the gradient in one direction. For example in Figure 4 if we have 8 processor, so processor 1 (level 1) and 5 (level 2) will work on the same rows (rows 1) but processor 1 will compute the gradient on x direction by applying the kernel 1 (Figure 4) for each pixel, and processor 2 will compute the gradient on y direction by applying the kernel 2 (Figure 4) for each pixel. Also processors 2 and 6, 3 and 7, 4 and 8 will work as same as processor 1 and 5.

5.2 Communication

There are two types of communication that exist in any parallel algorithms: local and global [6]. Local communication exists when a processor requires values from only a small number of other processor in order to perform a computation, while global communication occurs when a processor requires values from significant number of other processors [6]. In our method we have a local communication because our architecture is decentralized, and we don't need a global communication.

5.3 Synchronization

There are three types of synchronization: First is the barrier which is a basic mechanism for synchronizing processes, inserted at the point in each process where it must wait, and all processes can continue from this point when all the processes have reached it [10], another type of synchronization is deadlock which occur if both processes perform the send, using synchronous routines first. This is because neither will return; they will wait for matching receives that are never reached [10]. The third type of synchronization is the synchronization of communication operations, which involves the tasks executing a communication operations [10]. In this paper we have a synchronization of communication depending only on send and receive message between the processor. In this paper the synchronization for Sobel edge detection was done as follow:

1. The master send to each worker the size of segment they will work on it.
2. The master send to each worker the width of image.
3. The master sends to level 1 worker the lower bound for different rows from the image.
4. The workers received all previous data from master.
5. Each level 1 worker send his data to corresponding level 2 worker to work on different kernel.
6. The level 2 workers after applying the second kernel on segment of image, it returned the segment of image after computation to it's corresponding kernel 1.
7. The level 1 worker receive the data from it's corresponding level 2 worker after applying the kernel 2 on the segment.
8. The level 1 worker send the segment after finding the magnitude to the master.
9. The master receive previous segments from each level 1 worker, and end the edge detection algorithm.

5.4 Load Balancing

Load balancing refers to the practice of distributing work among processor so that all tasks are kept busy all of the time, in order to minimize the idle time of processor [11]. load balancing algorithms can be categorized as static or dynamic [11]. The static load balancing algorithms distribute the tasks to processing elements at compile time, while dynamic algorithms bind tasks to processing elements at run time [11]. In this paper we used the static load balancing, and we achieve it at two stages:

- First stage was done by equally partition the work between processor because we have the same power of processor, where each worker processor work on the same size of segment from image, and also the master work on some exceed rows of images to find the gradient.
- Second stage was done by equally distribution the segment after finding the gradient between the level1 workers to find the magnitude in order to distribute the work of the master.

5.5 Serial and Parallel Sobel Edge Detection Algorithm

The serial algorithm for the sobel edge detection is illustrated in Algorithm 1.

Algorithm 1 Serial Sobel edge detection algorithm

Read the grayscale image into array
Apply convolution kernel 1 on each pixel of image by multiplication, the output is Gx.
Apply convolution kernel 1 on each pixel of image by multiplication, the output is Gy.
Find the absolute value for each pixel in Gx.
Find the absolute value for each pixel in Gy.
Sum the Gx and Gy matrix, the output is M.
Read suitable threshold (t) from user.
for all $pixel(i) in M$ **do**
  if $M(i) > t$ **then**
    $newImage(i) = 1$
  else
    $newImage(i) = 0$
  end if
end for
Write newImage.

The Parallel algorithm for the sobel edge detection is illustrated in Algorithm 2, depending on master and workers processors.

Algorithm 2 Parallel Sobel edge detection algorithm

```
Initialize MPI
Finding the number of processor will work on different data, noprocessor = (no of processor)/2 +1
Load the image, for all processors less than noprocessor.
if master then
    Finding the size of remain rows of image.
    Finding the size of segment for each worker processor, by dividing the height of image on the number of salve processor after substract the remain rows.
    Finding the lower bound of segment for each worker processor.
    Sending the lower bound of segment and size of segment for each worker processor.
    if remainrows > 0 then
        Applying kernel 1 on remain rows.
        Applying kernel 2 on remain rows.
        Finding the magnitude for segment, and storing it in edge matrix at it's lower bound index
    end if
    For each worker less than noprocessor, receiving the segment after Finding the magnitude and the size of segment and storing it as following :
    storing receiving data in edge matrix at the lower bound index of sending processor.
    Writing the edge matrix as output.
else
    Receiving the size of each segment, and the lower bound of the segment from the master.

    if rank < noprocessor then
        Reading the data of segment depending on the lower bound received from master
        Sending the data(segment) to the myrank + (noprocessor − 1)
        Applying kernel 1 on the segment
        Recieving the segment after applying the kerne2 from myrank + (noprocessor − 1)
        Finding the magnitude of segment.
        Sending the segment after finding the magnitude to master
    else
        Receving the data(segment) from the (noprocessor − 1) − myrank.
        Applying kernel 2 on the segment
        Sending the segment after applying the kernel to (noprocessor − 1) − myrank.
    end if
    Finalize the MPI
end if
```

5.6 Performance Evaluation Method

The evaluation of the parallel execution performance is measured by speedup, performance improvement and efficiency with reference of time taken for both sequential and parallel processing [6].

- Speedup is a measure that captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processing element to the time required to solve the same problem on a parallel computer with P identical processing elements [7], the equation of speedup is:

$$speedup = \frac{Sequential(time)}{Parallel(time)}$$

- The performance improvement measures the relative improvement that the parallel algorithm has over the sequential. This performance is measured as following formula [8]:

$$Improvement = \frac{Sequential(time) - Parallel(time)}{Sequential(time)}$$

- Efficiency of a parallel program is a measure of processor utilization and is calculated using the following formula [8]:

$$Efficiency = \frac{Sequential(time)}{no.processor \times Parallel(time)}$$

6 Experiment and Results

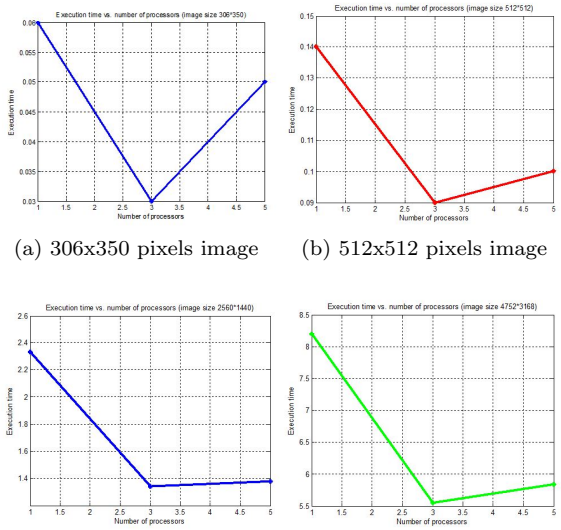
6.1 Experimental Setup

The experiment was implemented on one machine. This machine have 2 processors (Core 2 Due), 2GB RAM, and 2.1 GHz processor's speed. The software required to perform the parallel process are Ubuntu 11.10, lam-MPI library and CPU monitoring software for performance measure.

The experiment was done on 306x350 pixels, 512x512 pixels, 2560x1440 pixels, 4752x3168 pixels images.

6.2 Results

The Parallel results are discussed based on speedup, performance improvements and efficiency that mentioned before. Figure 5 shows the execution time of both serial algorithm and parallel algorithm for different size of images with different number of processors.



(a) 306x350 pixels image (b) 512x512 pixels image

(c) 2560x1440 pixels image (d) 4752x3168 pixels image

Figure 5: Execution time for 4 different images size using different number of processors

As shown in Figure 5 the total execution time for all images in the figure is reduced significantly at 3 processors. In Figure 5, in (a) until 5 processors the parallel algorithm is less than serial algorithm, but in (b) the parallel algorithm is suitable than serial until 7 processors, also in (c) the parallel algorithm is suitable than serial until 29 processors, and in (d)the parallel algorithm is suitable than serial until 29 processors. This difference in results returns to two reasons: the size of the image and the actual number of processors on the machine.

Figure 6 shows the speedup results against number of processors for 4 image sizes.

In Figure 6, all images gain good speedups using 3 number of processors, that because of the experiment is done using dual core machine (actual number of processors equals 2). However, all reached saturation point at different processors related to the size of image, and is predicted to have less difference in speedup as number of processors increases.

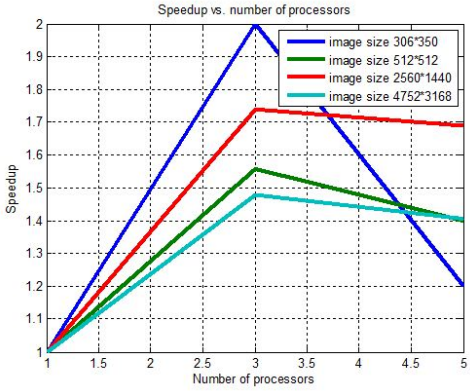


Figure 6: speedup for 4 different images size using different number of processors

Figure 7 shows the performance improvement of parallel algorithm for different size of images with different number of processors. In this Figure we can see that the best improvement is done at 3 number of processors for all image sizes.

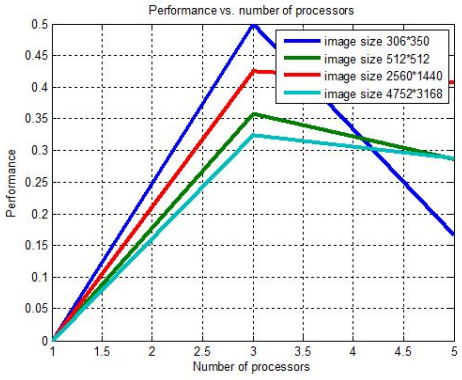


Figure 7: Performance improvement for 4 different images size using different number of processors

Figure 8 shows the efficiency of parallel algo-

rithm for different size of images with different number of processors. In this figure we can see that the efficiency of the parallel algorithm decreases as the number of processors increases for all images sizes.

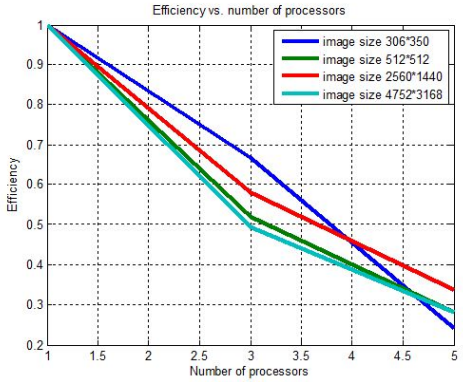


Figure 8: Efficiency for 4 different images size using different number of processors

7 Conclusion

In this paper we have presented a design of parallel edge detection algorithm, this design was based on domain and function decomposition and decentralization architecture. The results shows that the parallel algorithm improves the serial Sobel edge detection algorithm. Our design gives a good speedup for different image sizes until specific number of processor, that related to the environment of the experiment and the communication time for the proposed architecture, so we hope in the future to apply this new algorithm on a Beowulf cluster to improve our results, and improve the results for the large images.

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] A. L. Jackson, "A parallel algorithm for fast edge detection on the graphics processing unit," *Honor thesis, Washington and Lee University*, 2009.

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

[5] P. T. Group, *Introduction to MPI*. NCSA Access, Board of Trustees of the University of Illinois, third ed., 2001.

[6] N. Haron, R. Amir, I. A. Aziz, L. T. Jung, and S. R. Shukri, "Parallelization of edge detection algorithm using mpi on beowulf cluster," *IEEE International Conference on Systems, Computing Sciences and Software Engineering*, 2009.

[7] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing*. Addison Wesley, second ed., 2003.

[8] N. E. A. KHALID, S. A. AHMAD, N. M. NOOR, A. FIRDAUS, A. FADZIL, and M. N. TAIB, "Analysis of parallel multicore performance on sobel edge detector," *Proceedings of the 15th WSEAS international conference on Computers*, 2011.

[9] A. Vajda, "Programming many-core chip," *Springer*, 2011.

[10] I. Foster, *Designing and Building Parallel Programs*. Addison Wesley, 1995.

[11] A. Mandal and S. C. Pal, "An empirical study and analysis of the dynamic load balancing techniques used in parallel computing system," *Proceedings of ICCS*, 2010.