Parallel Image Processing

Divin Visariya
Computer Science
Rochester Institute of Technology,
Rochester, United States
djv9999@rit.edu

Sreeprasad Govindankutty
Computer Science
Rochester Institute of Technology,
Rochester, United States
sxg4222@rit.edu

Varun Goyal
Computer Science
Rochester Institute of Technology,
Rochester, United States
vvg1074@rit.edu

***Abstract* —** *This paper presents the effect of parallel computing on image processing. We present various image processing problems such as converting a Color image into a Black and White image, Sepia image, Negative image, the flipped image and performing edge detection on the image. All these operations are computationally intensive as they process every pixel of the image individually, thus has a degrading performance effect if performed sequentially. This paper presents how to perform these calculations using parallel computing and analyzing the time difference of the two methods. Parallel computing here implies two different methods those are Shared Memory Processor and Cluster.*

I. INTRODUCTION

Image Processing is a computationally intensive operation that requires a lot of CPU cycles for simple image transformation. Image processing involves manipulation of every pixel of an image to perform a transformation to a new image. Transformation such as converting a color image into black and white requires selecting every pixel in the image and manipulating its RGB values. The image can be divided into smaller chunks and the same transformation operations implemented on each chunk. Thus, the Image processing is a good candidate for running on parallel processor to improve the speed of computation when there are multiple images to be processed. Image processing is thus a massively parallel problem.

In this paper, we will study some simple image transformation and perform a complex operation like edge detection. Specifically we study how the image processing performance can be improved using parallel processing.

II. SOLUTION DESIGN

A. *Black and White*

The only way to convert the color image to a black and white version is by making changes to every pixel of the image. One needs to extract the Red, Green and Blue part of the pixel and perform a mathematical operation to get the grayscale value. The formula for the determining the gray scale value is:

*Gray Scale Value = red * 0.3 + green * 0.59 + blue * 0.11*

With help of the Gray Scale Value we can make the resultant matrix of pixel and write to the file as a Black and White version of the color image.

B. *Flip image*

The input image can be flipped horizontally by copying the left pixels to the right and vice versa. The flip operation just copy the pixel as it is and makes the new image which has the image in the mirror format. Thus, the image is being flipped. Sepia of Image

C. *Negative of Image*

The input image can be inverted making the new pixels values as (255-original pixel). Thus we get the inverted format or negative format of the image. This image is the one which we had when we used to use the films for photographs.

D. *Sepia of the image*

The input image can be converted into the sepia equivalent by modification of every pixel using the sepia value which decides the amount of change to be made. The Red and Green values are altered by the sepia value. We also have the sepia intensity which increases the effect. This intensity value is use to alter the Blue part of the pixel.

E. *Edge Detection*

There are many ways to determine the edge from a given image. In this paper, we study the most commonly used Edge detection called Canny Edge detection. Canny Edge detection has better edge detection and good localization than other edge detection algorithms and hence is widely used. Initially we smoothen or even the picture with a two dimensional Gaussian. Two-dimensional Gaussian is given by the equation below:

$$G(x,y) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\left[\frac{(x^2+y^2)}{2\sigma^2}\right]}$$

From this distribution, smoothing masks can be generated. The width of the generated smoothening masks depends upon the standard deviation value. The standard deviation determines the spread of the filter. Then by means of normalization, contrast of the picture decreased. This is required to eliminate the intrinsic noise present in the image. Gradient of the image with respect to intensity is taken before non-maximal suppression of the image is completed. Then by means of hysteresis the threshold of the edges is calculated and the final image is rendered as the output file.

III. ANALYSIS

A. *Image Modifications to various formats*

The main logic behind this modification is that we visit every pixel of the input image. Initially the input image is read into an array by using the PixelGrabber. With the array of the input image, the algorithm visits every pixel of the image and makes the modifications as required. The results are stored in their respective data structure. Finally, with all the modifications done we have the results stored in the different images as the output.

For the SMP version of the modification algorithm, we have the Parallel Job which speeds up the computation. The slicing among the cores are done row wise. Every core performs the job on their respective cores. After the computation we have the final images stored in the similar way.

For the Cluster Version, the image is divided into parts by using ranges. Every Processor computes the result images and sends the result back to the process 0 in the form of reduction. The process 0 performs the final merging of the results into the matrix and the results are stores in images as output.

B. *Canny Edge Detection SMP*

Here, we share the computation process equally amid different processes of a multi core processor. For this the image's pixel data array is sliced corresponding to the height of the image. Each process is responsible for processing its own slice of array at the same time. Once the processing is done the system being of same shared memory, the collated processed new data array is ready. This array is then written onto a new image file generating the canny edge image

C. *Canny Edge Detection Cluster*

Here, we share the computation process equally amid different processors. For this the image is sliced corresponding to the height of the image. Each processor is sent a small chunk of the image for processing. The pixels of the each chunk are grabbed using pixel grabber by each processor. The pixels so grabbed are saved to one-dimensional array. The dimensions of the array are the dimensions of the image slice shared between each processor. Next, the luminance of every pixel is computed and stored in the array. Each processor will store this in its local array. Histogram is applied to normalize the contrast. This is necessary to diminish the inherent noise presented in the image. The resultant image is then remapped by means of data available from the histogram. Next the slope of the image is computed to even or level the boundaries. Each processor then performs hysteresis to identify the edges. The results from each processor are communicated by reduce function to process 0. Process 0 finally writes the output image to a file.

D. *Challenges faced during implementation*

- Speed up achieved is restricted due to loads of message passing amid multiple processors.
- Every image is split to slices corresponding to the image size and each processor is assigned a slice to compute. This means each processor must receive an image slice from process 0. This involves lot of data transfer over the network and consequently a lesser more time for computing edges of the image.

IV. PERFORMANCE EVALUATION

A. *Image Modifications to Various Formats*

The modification of every pixel of the image becomes computation intensive when the image is large and the number of modification increases. Thus, the operation is time-consuming and can be improved using parallel processing. Since, we visit all the pixel of the image, and perform the calculations on every pixel independently; we don't have any dependencies among the pixels. We can parallelize the calculations among multiple processors or cores.

The SMP version of the modification algorithm has a very good performance improvement since it divides the job of modifications among the multiple processors and then finally obtains the results from all to get the correct output.

The Cluster Version of the modification algorithm splits the image into parts and sends to multiple processors. The computation part has a performance improvement, but the message passing between different processors caused the overall performance to degrade. We have different output for every image and this algorithm makes lot of output which has to be passed to processor 0 for the final output. Thus, a lot of data has to be passed after the computation.

Overall, the best improvement was achieve in the SMP version of the modification algorithm

B. *Canny Edge detection in SMP*

Canny Edge detection is a good contender for parallelizing in multi core processor. This is because of it being massively parallel program. There is a lot of processing to be done on each and every individual pixel. Now when processing a picture of resolution 1600 * 1600 it implies that number of pixels in such an image are 2,560,000 which is a lot of pixels, which makes it embarrassingly parallel problem. To reduce the processing, instead of storing the pixel data in a matrix we store it in a 1-D array.

Thus, we parallelize certain parts of the program, such as, the calculation of gradients by dividing the calculation in equal blocks of Gaussian width, each block to be computed by individual process thus calculating the x & y convolution and finally calculating the gradient magnitude. Similarly we parallelize the normalization process that is, the histogram method however the problem being a reduction is needed here to accurately calculate the normalized pixels. Thus a reduction is performed and we get a good speed up. With guided scheduling for load balancing, the speedup becomes even better.

C. *Canny Edge detection in Cluster*

Canny Edge detection is not a fine contender for executing in cluster for a number of rationales. Principally there are lots of message communications between the processors that

necessitate a lot of time. In the beginning, the image is split corresponding to its height and each processor is sent out a chunk or slice of the image to be processed independently. If the chunk size is huge then effectiveness depends on network load and size of image chunk being transmitted. The processor independently computes the luminance of every pixel present in its own slice of image and stores the value it its own local one-dimensional array. The dimension of this array is the dimensions of the image chunk being transmitted to every processor. The image may intrinsically have noise, which must be removed before detecting the edges. For this we normalize the contrast using histogram. Each processor then computes the gradient of the image by performing hysteresis. The reduce function is used to send this data to process 0 which then writes the edges to image output file.

	Canny Edge Detection SMP version			
Ideal Speed up	***Time (msec)***	***Actual Speed up***	***Efficiency***	***EDSF***
1	19132	1	1	
2	9652	1.982179859	0.99108993	0.008990174
3	6497	2.944743728	0.981581243	0.509687111
4	5329	3.590167011	0.897541753	0.760299625
5	4395	4.353128555	0.870625711	0.780915744
6	3765	5.081540505	0.846923417	0.827986348
7	3345	5.719581465	0.817083066	0.869853918

TABLE 1

	Canny Edge Detection Cluster version			
Ideal Speed up	***Time (msec)***	***Actual Speed up***	***Efficiency***	***EDSF***
1	7902	1	1	
2	5608	1.409058488	0.704529244	0.419387497
3	5079	1.555818074	0.518606025	0.858505706
4	4887	1.61694291	0.404235727	0.949596377
5	5016	1.575358852	0.31507177	1.032995703
6	4994	1.582298759	0.26371646	0.994736842
7	5552	1.423270893	0.203324413	1.130356428

TABLE 2

Figure 1. *Original Image*

Figure 2. *Norm of the gradient format*

Figure 3. *Thresholding to get Edges.*

V. CONCLUSION

Image processing is not good contender for cluster processing for the reason that so much message passing involved between the computing processors affects efficiency in terms of speed and time. In SMP, there is no message passing drawn in and the image is present within the memory boundaries of single processor. Consequently, we are able to accomplish better speed and completely render the edges of the image in a relatively lesser time compared to sequential version. Parallel Image Processing can be used in cluster version, if the size the problem being solved has no sequential dependencies and there is minimum message passing involved between the computing processors.

REFERENCES

[1] Canny Edge Detector: Wikipedia
http://en.wikipedia.org/wiki/Canny_edge_detector

[2] Canny Edge detector Reference by Tomgibara
http://www.tomgibara.com/computer-vision/canny-edge-detector

[3] Canny Edge detector Java Implementation by Suraj
http://suraj.lums.edu.pk/~cs436a02/CannyImplementation.htm

[4] Prof. Roger S. Gaborski slides at: http://www.cs.rit.edu/~rsg/