

Implementation of Selected Image Processing for Single and Multiple Processors

Wafaa Bazzi

American University Of Science & Technology

Abstract

Image processing contributes on many of the technological advancements today. It is widely used by the developing institutes as well as a source of important data.

One of the main things that are considered through this process would be the length of time on dealing with the application of different routines, on these images. Thus, time, being a valuable criterion for the efficiency of the systems, demands a better way for how these images will be managed. With the given situation, the idea of integrating a number of computers to perform image manipulation is considered. This uses the idea of parallel computing.

Images obtained through various sources are rarely of perfect quality. They may be degraded or noisy due to poor scanning technique or poor impression condition. Hence enhancement techniques are applied on images in order to obtain a clear image for display.

This Thesis focuses on implementing different image processing routines integrated into a system that will execute it on single and distributed processors. Initially it will deal with routines such as threshold, brightness, contrast, smoothing, edge detection and enhancement Gaussian Filter, secondly the Thesis considers two important expensive resources which are memory and time efficiency. It then explains how the existing codes which I implement them by using visual basic net 2008 , has improved to visual basic net 2010 which included many parallel programming features ,and if these are used, they reduce Programming complexity .

The system will be designed to execute the routines on image processing, as well as, to present the statistics in relation to the time required for processing images through single and multiple processors, these statistics will help evaluate the performance between single and distributed computing.

Experiments using different image processing routines are conducted to enhance the images and to show a comparative result in terms of time used to enhance image by sequential method and time used to enhance

image by parallel method on each enhanced image. Experimental results out performs well to overcome the counterpart of enhancement technique.

Introduction and Problem Statement

Image processing contributes to many of the technological advancements today. It is widely used by the developing institutes to provide visual information as well as a source of important data which gives way to the development of more flexible and intelligent systems.

One of the main things that are considered through this process would be the length of time for dealing with the application using different routines. Thus, time, being a valuable criterion for the efficiency of these systems, requires better ways to be minimized upon managing these images.

Images are looked upon as mappings from a set of coordinates(pixels). These pixels are commonly manipulated through these perspectives along with some of the basic routines. Image operator classes include point operators, local operators, and global operators which are usually executed by a single processor on a single computer. Although these operations can work on a single platform, the performance is at stake. From a real world point of view, the performance of the system is taken into consideration given that the images it would be handling range from very small to very large. With the given situation, the idea of integrating a number of computers to perform image manipulation is considered. This uses the idea of parallel computing.

There are different routines which are commonly used for image processing. Some of these operators that will be included in this study are threshold, brightness, contrast, smoothing, edge detection and enhancement, erosion and dilation, opening and closing, and fill and connected.

This study focuses on developing a system that will handle the algorithms for sequential and parallel operations. The system will be designed to execute the routines on image processing, as well as, to present the statistics in relation to the time required for processing images through single and distributed processors.

1.1 Historical Background

Several years ago [1, 2], the digital image was used in several domains including military, academic, geographical, and industrial research laboratories. Day after day, the need to digital image is increased as all students download digital images from the web, and as photographs are being stored on a digital CD.

1.2 Statement of the Problem

The noise in the Image is the most important factor in determining image quality. In image processing, it is usually necessary to remove noise from the image in order to obtain a clear image for display.

The topic first addresses some digital image's algorithm such as spatial filter, sharpness, Laplacian filter, threshold and histogram equalization, secondly the topic considers two important expensive resources which are memory and time efficiency. It then explains how the existing codes which I implement them by using visual basic net 2008, has improved to visual basic net 2010 which included many parallel programming features, and if these are used, they reduce Programming complexity .

1.3 Objectives

The major objective of this study is to design and implement a system that will handle different image processes for including sequential and parallel implementations. Specifically:

- 1) To develop a system for implementing and evaluating different image processing routines
- 2) To implement the system on single and also on distributed processors
- 3) To incorporate the algorithms for sequential and parallel using VB2010 and Parallel tools which are available when using NET FRAME-Work 4.
- 4) To generate statistics with relevance to the performance of the system on single and distributed processors

1.4 Review of Literature

Image processing [1, 2] is gaining a larger importance in a variety of application areas. It requires extensive computational power to be able to operate in real time. For that reason, there has been an increasing interest in the development and the use of parallel algorithms in image processing.

The emphasis on algorithm design has shifted from sequential algorithms to parallel algorithms because the more computers are increasingly incorporating some form of parallelism.

Parallel algorithms are designed and analyzed with the goal to get exact bounds on their speed-ups on real machines. They consider "fast solution for gray scale image" as a basic problem in image processing. Their algorithms imply that the range becomes larger as the input size grows.

In 2001, the focus was on the implementation of low level image processing for parallel active vision systems. The image operator classes included in his study were point operators, local operators, dithering, smoothing, edge detection, morphological operators, and image segmentation.

Mathematical operation is usually used as the basis for image processing. It is aimed at speeding up some of the image algorithms by means of improvements on the original algorithms, or by the use of parallel computing techniques, it presented a method for introducing parallelism into an image processing application. It was based on algorithms for low, medium and high level image processing operations which provided an easy-to-use parallel programming interface.

In 2005, a multi-agent technique was used for the processing time comparison between sequential edge detection and a parallel edge detection application. These studies have encouraged the researcher to pursue on the development of a system for implementing and evaluating image processing routines on single and distributed processors.

By filtering the images, it is believed that the symptoms of image noise can be detected. The detection of noise involves a sequence for processing data within the image, and on observation of image enhancements methods.

[3, 4] Image enhancement is basically improving the perception of information in images for human viewers and providing 'better' output. The principal objective of image enhancement is to modify attributes of an image to make it more suitable for a given task and a specific observer. During the processes, one or more attributes of the image are modified so the result is that the noise of images is removed, and the three

characteristics for processing high resolution images are achieved, the system is designed to reduce computing latency, memory usage and stream processing of the input data..

In spatial domain techniques, we directly deal with the image pixels. The pixel values are manipulated to achieve the desired enhancement, the project focuses on spatial domain techniques for image enhancement, with particular reference to point and histogram processing methods. Some of these methods are Intensity Enhancement, Histogram, Histogram Equilization, Spatial Filter, Edge Detection, and Gaussian Filter. These enhancement operations are performed in order to modify the image brightness, contrast or the distribution of the grey levels, and will be discussed in chapter two .

In addition, this project focuses on the design of a parallel program for entering digital images, which involves looking for the mechanism that allows for image enhancement, and searching for a new conception of parallel computing by a self developed parallel image processing algorithm ,this will be discussed in chapter three .

Moreover, in chapter four we deal with the noise problem by the methods Intensity Enhancement, Histogram, Histogram Equilization, Spatial Filter, Edge Detection, and Gaussian Filter, also we deal with the expensive resources which are memory and time efficiency, Additionally, the problem is not well defined, as there is no objective measuring for the image quality. As result, we discuss few recipes that have shown to be useful both for the human observer and/or for the machine recognition. These methods are very problem-oriented: a method that works well in one case may be completely inadequate for another.

The test on greyscale image using 8 CPU cores (CORE i7 running at a frequency of 1.6 GHz) will show an enhancement using 8 Threads.

Furthermore, in chapter five we calculate the performance of my algorithms and make comparison between the sequantial methods and the parallel methods.

Finally, in chapter six, we conclude that if a modification in the sequantial algorithms will remove the noise completaly, and if a parallel version of solution is created will speed up the processes of enhancement.

Existing Solutions

The field of Image Processing and Computer Vision has been growing at a fast pace. The growth in this field has been in both the breadth and depth of concepts and techniques. This chapter aims at providing the fundamental techniques of Image Processing and the Computer Vision, and at showing how the image is save and manipulated inside the memory of a computer. In addition this chapter explains the techniques and operations on single images that deal with pixels and their neighbors (spatial operations)(normally removing noise by reference to the neighboring pixel values). There are also discussions on edge detection and on Enhancement by guassian filter.

2.1 Digital Image Definitions

Because the digital images [2, 5, 6, 7, 8] are stored and processed on digital computers, they become a discrete range of signals; these signals are represented as matrices containing the light intensity or color information at each point.

The Digital Image is seen as 2D Vectors $a(x,y)$, where x and y are spatial coordinates, and the amplitude of a at any pair of coordinates (x,y) is called the intensity or gray level of the image at that point . Note that a digital image is divided into N rows(width) and M columns(height)(finite number of elements) .The intersection of a row and a column is termed a *pixel*(picture element) . The value assigned to the integer coordinates $[m,n]$ with $\{m=0,1,2,\dots,M-1\}$ and $\{n=0,1,2,\dots,N-1\}$ is $a[m,n]$. As example the following figure 2.1:

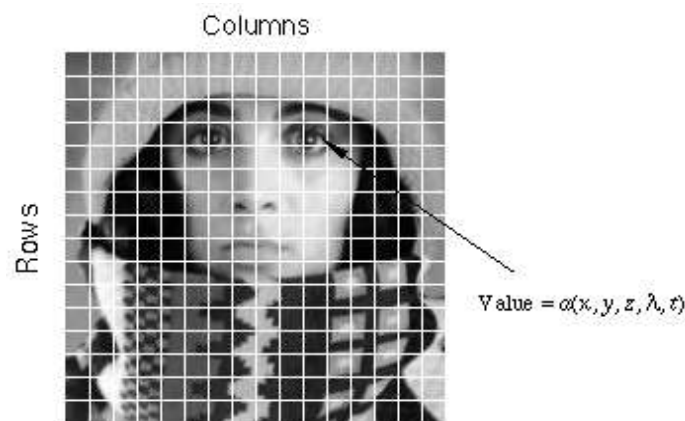


Figure 2.1 - Digitization of a Continuous Image.

The pixel at coordinates $[m=10, n=3]$ has the integer brightness value 110. The image shown in Figure 2.1 has been divided into $N = 16$ rows and $M = 16$ columns.

2.2. Different Categories of Image

Images are categorized [5, 6, 7] according to their source which are , Gamma rays, X-rays, Ultraviolet, Visible, Infrared , Microwaves, and Radio waves.

As I introduced in the previous section, images are of the function $a(x,y)$.

The function $a(x, y)$ may be characterized by two components : [5 , 7 , 8]

1. The amount of source illumination.
2. The illumination reflected by the objects in the scene.

$$A(x, y) = i(x, y) r(x, y)$$

2.3 Image Enhancement Reason

Often images obtained via [7] photography, digital photography, flatbed scanning, or other sensors can be of low quality due to poor image contrast. In order to enhance the image, suppose that I is the original image, we compute J the enhanced image. For each pixel in I :

- $J(x, y) = f(I(x, y))$. the function f is different from method to another .

2.4 . Proposed Solutions

Over decades, many image-enhancement techniques have been proposed. I implemented some of these techniques, and the program results are shown and discussed. The techniques can be divided into four categories: operations based on the image histogram, on simple mathematics, on convolution, and on mathematical morphology.

2.5. Image Transforms Classification

The Image can be changed by using [9] one of the following methods. The Point transform method when the point is modified individually on its pixels' locations, the Local transform method which derives the output from the neighbourhood or the Global transform by which the whole image contributes to each output value.

2.5.1. Point Transforms

Manipulating individual pixel values such as Brightness adjustment, Contrast adjustment, Histogram manipulation, and Equalisation. Monadic point-to-point operator as in the figure 2.2 [10]



Figure 2.2 - Point to Point Operation

2.5.2. Example1 Brightness Adjustment

Add a constant to all values by using the formula $g' = g + k$ ($k = 50$) the output in the following figure 2.3 [10]

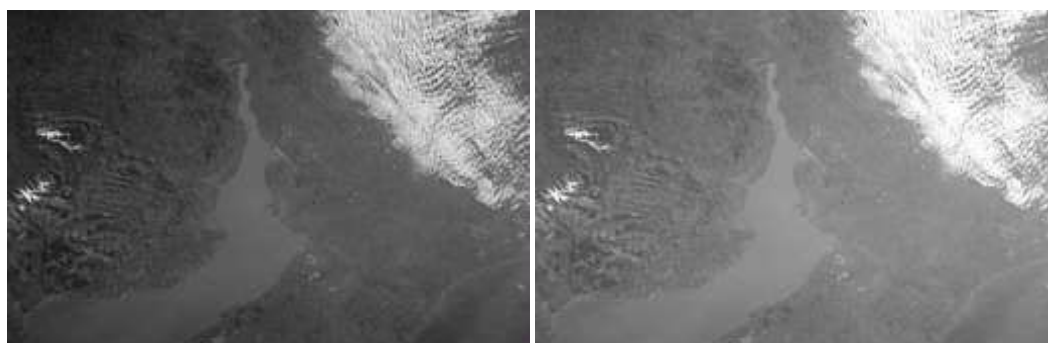


Figure 2.3 - Original Image is at the right, the Transformed Image is at the left.

2.5.3. Examples of Contrast Enhancement

Add, Subtract, Multiply, Divide, Maximum and Minimum.

2.6. Proposed Methods

In spatial domain techniques, we directly deal with the image pixels. The pixel values are manipulated to achieve the desired enhancement, the project focuses on spatial domain techniques for image enhancement, with particular reference to point and histogram processing methods.

2.6.1. Threshold

The simplest method to enhance [2, 4, 7, 8] grey-scale image and remove noise is to threshold it. This is an important function, which converts a grey scale image to a binary format. Unfortunately, it is often difficult, or even impossible to find satisfactory values for the user defined integer threshold value. The threshold equation is bellow in the formula 2.1.

$$C(x,y) \leq \begin{cases} W & a(x,y) \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Formula 2.1-Thersholding Equation

2.6.1.2. Threshold Pseudo Code

```
for (int y=0; y<height; y++)
    for(int x=0; x<width; x++)
        if (input.getxy(x,y) < threshold)
            output.setxy(x,y, BLACK);
        else
            output.setxy(x,y,WHITE);
```

2.6.1.3. Some Results of Thresholding Images

The MonaLisa image is in the figure 2.4 Upon Thresholding it, we obtain the Figure 2.5



Figure 2.4- Monalisa



Figure 2.5- Thresholding of figure 2.4

2.6.1.4. Advantage of Thresholding

If the input to a thresholding operation is a grayscale or color image, the output is a binary image. Black pixels correspond to background and white pixels correspond to foreground(or *vice versa*) and the segmentation is determined by a single parameter known as the *intensity threshold*. In a single pass, each pixel in the image is compared with this threshold. If the pixel's intensity is higher than the threshold, the pixel is set to, say, white in the output. If it is less than the threshold, it is set to black. Another common variant is to set to black all those pixels corresponding to background, but leave foreground pixels at their original color/intensity(as opposed to forcing them to white), so that that information is not lost. So one benefit that is thresholding may be applied to grey-level images, another benefit is the simple processing, the third benefit is that the algorithms are simple, and the fourth benefit is Low storage (no more than 1 bit/pixel), often this can be reduced as images are very amenable to compression(e.g. run-length coding).

2.6.1.5. Disadvantage of Thresholding

The First issue that should be notice is that the thresholding requires the Transform grey/colour image to a binary function, if $f(x, y) > T$ output = 1 Else output=0 so find it. It is difficult to construct the value of T. The second issue is the need to experiment the figures which were enhanced by threshold, so the image is quite readable. The third issue is that the best threshold value changes from an image to another. The larger the image, the harder it is to choose acceptable threshold value. So Enhancing the method is required to let the user choose the level of image to be completely binary(white and black and white) or to contain many levels Intensity Level method is used for this purpose.

2.6.2. Intensity Level

Converting a continuous image [5] function into a digital image requires sampling and quantization. This digitization process requires decisions to be made regarding the values for M(number of rows), N(number of columns), & L(number of discrete intensity levels). Due to storage and quantizing hardware considerations, the number of intensity levels is an integer power of 2, so $L=2^k$ such that k ranges from 1 to 7.

Humans can discriminate about 128 levels of Grey scale values(7 bit digitization needed).

Sometimes, the range of values spanned by the gray scale is referred to as dynamic range. Intensity resolution similarly refers to the smallest discernible change in intensity level .the intensity levels are reduced from 256 to 2 in the follow figure 2.6.

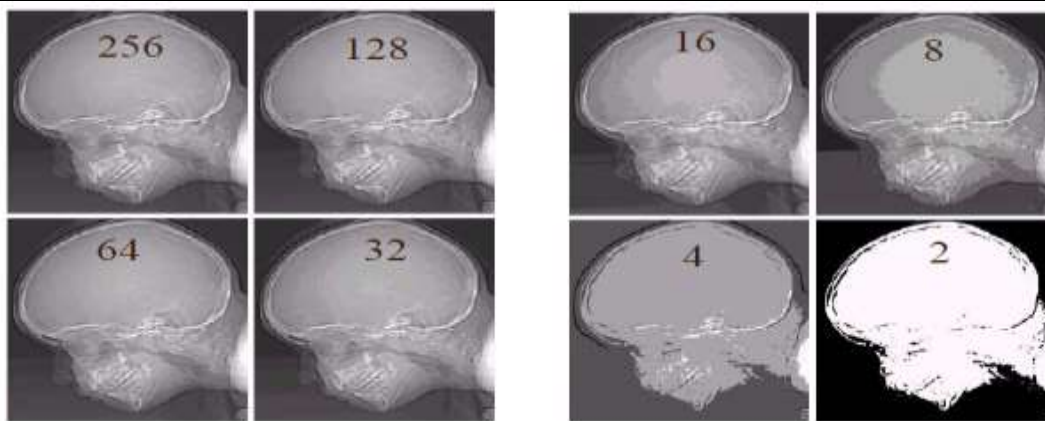


Figure 2.6 – Intensity Enhancements of an image [11]

26.2.1. The Advantages of Intensity Level Enhancement

Each color model breaks the light into one or more channels and then assigns an intensity level of each channel for each pixel in the image. The bitmap color mode contains only one channel with only two possible levels, 0 and 255. The input is a [grayscale](#) or [color image](#) the output is a reduced level gray scale image. In a single pass, each pixel in the image is same but with less contrast to produce output image between binary image level and the selected level. So one benefit of the method is that a bitmap is a black and white image without color and even without shades of gray, second benefit is that it may be applied to grey-level images, third benefit is simple processing, fourth benefit the algorithms are simple, fifth benefit is the lower storage than the original image.

2.6.2.2. The Disadvantages of Intensity Enhancement

One of the disadvantages is the manual choosing of the level. So The intensity method used to enhance image sometimes blurs out the result .In addition each time you make a correction to an image, change the levels, add a filter, and so on, you lose a little bit of the distinguishing details. Other disadvantages is that there is no separation for the object from its background, example of the failing thresholding segmentations are shown in fig2.7 and figure 2.8 of figure 2.9 so the solution is the histogram equalization method.



Figure 2.7- Failing Thresholding Figure 2.8- Failing Thresholding Figure 2.9 -Failing Thresholding

2.6.3. Equalization

In an image processing [5, 7, 8, 9, 10] context, the histogram of an image normally refers to a histogram of the pixel intensity values. This histogram is a graph showing the number of pixel in an image at each different intensity value found in that image. For an 8-bit grayscale image there are 256 different possible intensities, and so the histogram will graphically display 256 numbers showing the distribution of pixels

amongst those grayscale values. Histograms can be also made toward color images, either individual histograms of red, green and blue channels can be taken, or a 3-D histogram can be produced, with the three axes representing the red, blue and green channels, and the brightness at each point represents the pixel count. The exact output from the operation depends upon the implementation, it may simply be a picture of the required histogram in a suitable image format, or it may be a data file of some sort representing the histogram statistics.

2.6.3.1. How It Works

The operation is very simple. The image is scanned in a single pass and a running count of the number of pixels found at each intensity value is kept. This is then used to construct a suitable histogram.

2.6.3.2. Guidelines for Use

Histograms have many uses. One of the most common uses is determining what value of threshold to use when converting a grayscale to a binary one by thresholding. If the image is suitable for thresholding then the histogram will be *bi-modal* i.e. the pixel intensities will be clustered around two well-separated values. A suitable threshold for separating these two groups will be found somewhere in between the two peaks in the histogram. If the distribution is not like this then it is unlikely that a good segmentation can be produced by thresholding. The intensity histogram for the input image in fig2.10 is at the right of fig2.10 [\[10\]](#)



Fig 2.10 - Input Image at the left and its Histogram is at the right .

The object being viewed is dark in color and it is placed on a light background, and so the histogram exhibits a good bi-modal distribution. One peak represents the object pixels, and one represents the background. The histogram for input image in fig 2.11 [\[10\]](#)

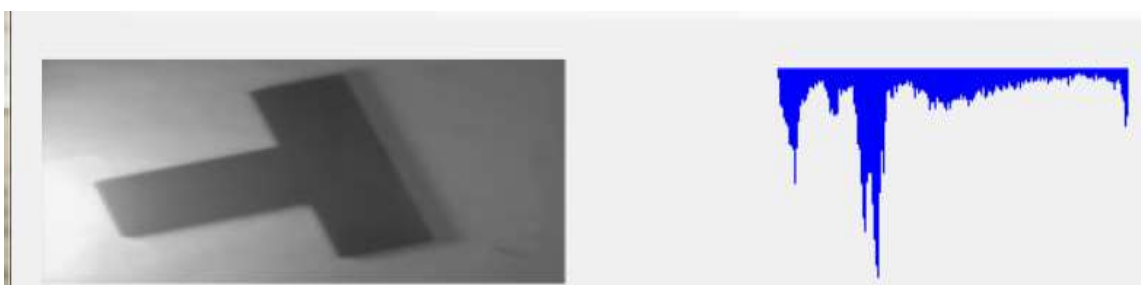


Fig 2.11 – Input Image at the left and its Histogram is at the right .

This time there is a significant incident illumination gradient across the image, and this blurs out the histogram. The bi-modal distribution has been destroyed and it is no longer possible to select a single global threshold that will neatly segment the object from its background.

The histogram is used and altered by many image enhancement operators. Two operators which are closely connected to the histogram are contrast stretching and histogram equalization. They are based on the assumption that an image has to use the full intensity range to display the maximum contrast. Contrast stretching takes an image in which the intensity values don't span the full intensity range and stretches its values linearly.

2.6.3.3. Brief Description

Histogram modeling techniques(*e.g.* histogram equalization) provide a sophisticated method for modifying the dynamic range and contrast of an image by altering that image such that its intensity histogram has a desired shape. Unlike contrast stretching, histogram modeling operators may employ *non-linear* and *non-monotonic* transfer functions to map between pixel intensity values in the input and output images. Histogram equalization employs a monotonic, non-linear mapping which re-assigns the intensity values of pixels in the input image such that the output image contains a uniform distribution of intensities.

2.6.3.4. The Step of Histogram

Histogram modeling is usually introduced using continuous, rather than discrete, process functions. Therefore, we suppose that the images of interest contain continuous intensity levels(in the interval $[0,1]$) and that the transformation function f which maps

an input image $A(x,y)$ onto an output image $B(x,y)$ is continuous within this interval. Further, it will be assumed that the transfer law(which may also be written in terms of intensity density levels, *e.g.* $D_B = f(D_A)$) is single-valued and monotonically increasing(as is the case in histogram equalization) so that it is possible to define the inverse law $D_A = f^{-1}(D_B)$. Consider the Fig 2.12 of the moon [\[10\]](#)



Fig 2.12 – Moon Figure

Which shows an 8-bit grayscale image of the surface of the moon. The histogram in the following figure 2.13. [\[10\]](#)

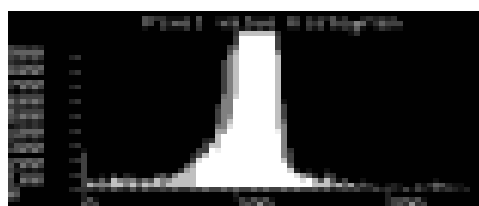


Fig 2.13 - Histogram of 2.12

Confirms what we can see by visual inspection: this image has poor dynamic range. In order to improve the contrast of this image, without affecting the structure(*i.e.* geometry) of the information contained therein, we can apply the histogram equalization operator. The resulting image is



Fig 2.14 - Equalization of 2.12.

and its histogram is shown in the follow figure 2.15

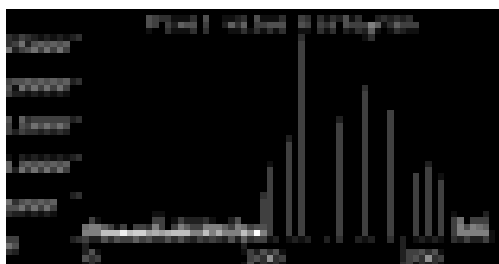


Fig 2.15 - Histogram of Equalized image 2.14.

Note that the histogram is not flat(as in the examples from the continuous case) but that the dynamic range and contrast have been enhanced. Note also that when equalizing images with narrow histograms and relatively few gray levels, increasing the dynamic range has the adverse effect of increasing visual grainyness. Compare this result with that produced by the linear contrast stretching operator which is shown in the figure 2.16 [\[10\]](#)



Fig 2.16 - Contrast Stretching of 2.12.

In order to further explore the transformation defined by the histogram equalization operator , Enhancing some images by using it. So Histogram processing & resulting transform curves in the following figure 2.17 [\[10\]](#)

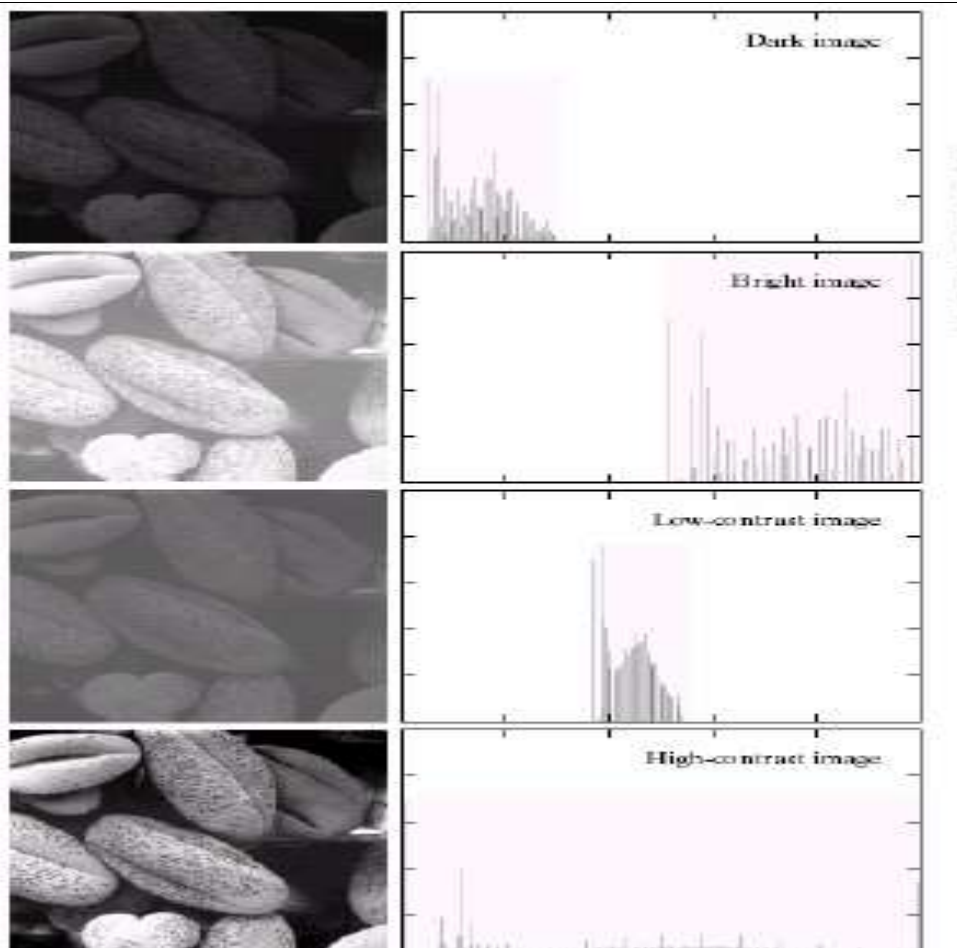


Fig 2.17 - Images and Histograms. [5]

2.6.3.5. Histogram and Histogram Equalization Formulas

The histogram is a discrete function: $h(r_k) = n_k$ where r_k is the k^{th} gray level and n_k is the number of pixels in the image having gray level r_k .

A normalized histogram is as $p(r_k) = \frac{n_k}{n}$ where $p(r_k)$ is an estimate of the probability of occurrence of gray level r_k .

The histogram of the image is a scaled representation of the PDF. The cumulative density function (CDF) is defined as follows in the formula 2.2

$$F_k = \sum_{i=0}^{i=k} p(r_k)$$

Formula 2.2- Summation of probability until k

The CDF is a non-decreasing function that goes from 0 to 1. The input-output relationship of HE is then in the follow formula formula 2.3

$$y_{out} = y_{min} + (y_{max} - y_{min}) F_k(y_m)$$

Formula 2.3- Equation of HE Function.

Where y_{min} (r_k minimum) and y_{max} (r_k maximum) are the minimum and maximum permissible luminance values. By itself, HE is typically not used for contrast enhancement since it over enhances the image and produces undesirable artifacts such as contouring, noise amplification and gray level crushing. The main

reason for this is that HE stretches the peaks of the histogram too much and consequently neighboring gray levels are pulled apart(much more than plain contrast stretching) and hence look discontinuous.

A Histogram Algorithm. We present an efficient algorithm here for computing the histogram of an image. The algorithm is described in a high level pseudo language that is easily translated into any language. The Algorithm for Computation of Histogram

```

For k=0 to 255 do
  C[k]=0 ; // c[k]=count of pixels at gray level k
For m=0 to M-1 do // For each Row and each column pixel
  For n=0 to N-1 do // Column in the image
    c[f[m,n]]=c[f[m,n]]+1; // Increment count at gray level count
  For k=0 to 255 do // Proportionalize each gray level count
    H[k]=c[k]/(M*N); // M*N =total pixel count , h[k] is prortion

```

B. The Equalizing Transformation

Let{ $f : k = 1, \dots, L-1$ } be the set of gray levels with histogram proportions{ $h_k : k = 0, \dots, L-1$ }. Then the gray level transformation is given by

$$g_k = \sum_{j=0}^k h_j = H_F(k) \text{ (For each gray level } k)$$

Formula 2.4 -Summation replaces the Integral of Equation (2.3).

where the summation replaces the integral of Equation(2.3). An algorithm for histogram equalization is given below for the case when the histogram has already been computed.

Algorithm For Histogram Equalization:

```

Sum=0.0;
For k=0 to 255 do // For each gray level
  Sum=Sum+h[k]; // sum histogram proportions
  H[k]=sum; // Collect cumulative values
For m=0 to M-1 do // For each Row and each column pixel
  For n=0 to N-1 do // position , compute new gray level g
    G[m,n]=255*H[f[m,n]]; // Compute and scale g , 0≤g≤1

```

The transformed gray levels $g[m,n]$ are approximately uniformly distributed across the grayscale(and are more uniform when the cdf is more one-to-one).

2.6.3.6. The Advantages of Histogram Equalization

The histogram mapping method is very good in enhancing contrast as shown in fig experiment. An advantage of the method is that it is a straightforward technique and an invertible operator(i.e., if the histogram equalization function is known, the original histogram can be recovered). Another advantage of histogram based techniques is simplicity of implementation of the algorithm. Also it should be mentioned that histogram based techniques are much less expensive compared to other methods. Another advantage of the histogram is that it shows the shape of the distribution for a large set of data; however the original data cannot be retrieved from a histogram.

2.6.3.7. The Disadvantages of Histogram Equalization

A disadvantage of the method is that it is indiscriminate. It may increase the contrast of background noise, while decreasing the usable signal. Sometimes we cannot read exact values of the data because data are grouped into categories, however, the more difficult it is to compare two data sets, so it can be used only with continuous data. Other disadvantage is that the actual data values cannot be determined from a histogram, only the number of values within intervals can be. So a method that groups the data into several categories like detect Edge method may be better than HE.

2.6.4. Enhance Image by Edge Detecting

The edge of an image [2, 5, 7, 8, 12] describes the boundary between an object and the background. It represents a sudden change in the value of the image intensity function. So an edge separates two regions of different intensities. However all the edges in an image are not due to the change in intensity values, where parameters like poor focus or refraction can result in edge in an image. The shape of edges in an image depends on different attributes like, lighting conditions, the noise level, type of material and the geometrical and optical properties of the object. Gradient operators of first derivative like Sobel, Prewitt, Roberts and second derivative like Laplacian are used to find the edge in an image. The efficient edge detection operator is evaluated subjectively by visually comparing the output images obtained with certain characteristics.

2.6.4.1. First & Second Derivative Filters

The gradient of an image $f(x, y)$ at the location (x, y) is given by the two dimensional column vector in the follow formula 2.5.

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Formula 2.5. - Second Derivative Equation .

The magnitude of the first derivative is used to detect the presence of an edge in the image. The magnitude of this vector is given by equation shown in the formula 2.6:

$$mag(\nabla f) = [G_x^2 + G_y^2]^{1/2}$$

Formula 2.6. - Magnitude of a Vector Equation.

Here $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ are the rates of change of two dimensional function $f(x, y)$ along x and y axis respectively. A pixel position is declared as an edge position if the value of the gradient exceeds some threshold value, because edge points will have higher pixel intensity values than those surrounding it.

A representation of the filter mask is as follow in the figure 2.18.

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

Figure 2.18- 3x3 Region of the Filter Mask.

Better Edge-gap in Grayscale Image Using Gaussian Method and multiply the vector w by the array w1 by (x-1,y-1) and w2 by (x-1,y)w5 by (x,y) w9 by (x+1,y+1) .

2.6.4.2. Laplace Operator

The sign of the second derivative [5, 8, 12] is used to decide whether the edge pixel lies on the dark side or light side of an edge. The second derivative at any point in an image is obtained by using the laplacian operator. The Laplacian for an image function $f(x, y)$ of two variables is defined as in the following formula 2.7:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Formula 2.7 - Equation Laplacian to Detect the Edge.

The Laplacian operator is given by the equation in the following formula 2.8:

$$\nabla^2 f = (W_2 + W_4 + W_6 + W_8) - 4W_5$$

Formula 2.8- Laplacian Operator Equation.

2.6.4.3. Methods

The following proposed algorithm is to get the edge map of the input gray scale image. The algorithm describes the general procedure used to find the edge map for all gradient operators.

2.6.4.4. ALGORITHM.

Edge detection by derivative operators.

Begin

- (1) Select a gray scale input image.
- (2) Store all the pixel values of the image along x and y coordinates in matrix form.
- (3) Generate the convolution mask for different gradient operators and store it in different matrices.
- (4) The sum of all the coefficients of each mask must be zero.
- (5) Each mask along the horizontal and vertical direction is convolved with the input image.
- (6) The magnitude of the gradient vector is obtained.
- (7) Finally, the gradient vector is normalized and threshold to a particular level for display of edge map information.

End

2.6.4.5. Advantages of Edge detection

In Edge detection algorithms large sections of these images are black [2, 5, 13] because the Laplacian contains both positive and negative values, and all negative value are clipped at zeroes as shown in edge detection experiments, see that the using of second mask is more suitable and the edges are shown in last experiments. In addition The enhanced figures which are obtained by using Laplacian, show that the detail in them unmistakably clearer and sharper than in the original image. Adding the original image to the Laplacian, the net result is an image in which small details were enhanced and background tonality was reasonably preserved. The edge detection extracts the edges and stores in the logical variable. Sometimes the edges are not closed due to lack of contrast in the image. Hence, the filtering finds all the blank points at least 2 edge neighbours, fills them and closes all the blank edges as if in the original edges. In some figures, we find that the image has some black points and edges are not closed enough, this problem is due to lighting and noising. However, in output Figure, after being filtered and sharpened, the object in the image is identified by a complete closed edge made of a sufficient number of pixels.

2.6.4.6. Disadvantages of Edge detection

Note that extracting edges may result in negative values. Thus, we must normalize the edge image to display the result.

The proposed methods include histogram equalization , thresholding , edge-detection algorithms, median filtering, and averaging are methods to be used as human tools, and are controllable by them .

In summary the existing methods are powerful grayscale image enhancement techniques leading to high contrast enhancement. But Image processing can be a time consuming task and, leads to conversion to a parallel algorithm.

Parallel Computing

There is a continuous demand for greater computational power from computer systems than is currently possible. One of the problems is the need for huge quantities of repetitive calculations on large amounts of data in order to give valid results.

Computations must be completed within a reasonable time period.

3.1. Parallel Computing

The past decade has seen tremendous advances [14, 15, 16, 17, 18, 19] in microprocessor technology, such as clock rates of processors have increased from about 40 MHz to over 2.0 GHz. At the same time, processors are now capable of executing multiple instructions in the same cycle, and the average number of cycles per instruction(CPI) of high end processors has improved over the past 10 years. In addition all this translates to an increase in the peak floating point operation execution rate(floating point operations per second, or FLOPS) of several orders of magnitude. Moreover faster computation by using of n computers, which operate simultaneously to achieve the result, so the time consumed is n times faster than computer operating serially. Other benefits include fault tolerance, larger amount of memory available.

The traditional logical view of a sequential computer consists of a memory connected to a processor via a data-path, and all three components processor, memory, and data-path which present bottlenecks to the overall processing rate of a computer system. A number of architectural innovations over the years have addressed these bottlenecks, and one of the most important innovations is multiplicity in processing units, data-paths, and memory units.

3.2. Trends in Microprocessor Architectures

While microprocessor technology [14, 15, 16, 17] has delivered significant improvements in clock speeds over the past decade, it has also exposed a variety of other performance bottlenecks, microprocessor designers have explored alternate routes to cost-effective performance gains, also the increments in clock speed are severely diluted by the limitations of memory technology, moreover at the same time, higher levels of device integration have also resulted in a very large transistor count, however techniques that enable execution of multiple instructions in a single clock cycle have become popular, Some examples of current generation of microprocessors Itanium, Sparc Ultra, MIPS, and Power4.

3.3. Pipelining and Superscalar Execution

Processors have long relied on pipelines [14, 15, 16, 17, 18, 19]for improving execution rates, by overlapping various stages in instruction execution(fetch, schedule, decode, operand fetch, execute, store, among others). Furthermore, in typical instruction traces, every fifth to sixth instruction is a branch instruction, and Long instruction pipelines therefore need effective techniques for predicting branch destinations so that pipelines can be speculatively filled .In short, a way to improve instruction execution rate is to use multiple pipelines, during each clock cycle, multiple instructions are piped into the processor in

parallel, and these instructions are executed on multiple functional units. For example, the Pentium 4, which operates at 2.0 GHz, has a 20 stage pipeline

First issue notice that data may be in True data dependency which means that the results of an instruction may be required for subsequent instructions; second issue is that the instructions maybe in *Resource dependency* which means two instructions compete for a single processor resource. How to resolve the dependency between instructions results from the finite resources share by using various pipelines, the idea is the co-scheduling of two floating point operations on a dual issue machine with a single floating point unit, third issue is that the procedure maybe in branch dependencies or procedural dependencies which means execution of a conditional branch instruction, and the branch destination is known only at the point of execution, and scheduling instructions a priori across branches may lead to errors. A proposed solution is to schedule the branches and rolling back in case of errors. One consequence is that the ability of a processor to detect and schedule concurrent instructions is critical to superscalar performance.

The processor needs the ability to issue instructions out-of-order to accomplish desired reordering. Most current microprocessors are capable of out-of-order issue and completion. This processor uses a window of instructions from which it selects instructions for simultaneous issue. This window corresponds to the look-ahead of the scheduler. And the performance of superscalar architectures is limited by the available instruction level parallelism.

3.4. Limitations of Memory System Performance

The effective performance of a program [14, 15, 16, 17, 18, 19] on a computer relies not just on the speed of the processor but also on the ability of the memory system to feed data to the processor. At the logical level, a memory system, possibly consisting of multiple levels of caches, takes in a request for a memory word and returns a block of data of size b containing the requested word after l nanoseconds. Here, l is referred to as the latency of the memory.

The rate at which data can be pumped from the memory to the processor determines the bandwidth of the memory system.

It is very important to understand the difference between latency and bandwidth since different, often competing, techniques are required for addressing these. As an analogy, if water comes out of the end of a fire hose 2 seconds after a hydrant is turned on, then the latency of the system is 2 seconds. Once the flow starts, if the hose pumps water at 1 gallon/second then the 'bandwidth' of the hose is 1 gallon/second. If we need to put out a fire immediately, we might desire a lower latency. This would typically require higher water pressure from the hydrant. On the other hand, if we wish to fight bigger fires, we might desire a higher flow rate, necessitating a wider hose and hydrant.

3.5. Improving Effective Memory Latency Using Caches

One such innovation addresses the speed mismatch [15, 16, 17, 18, 19] by placing a smaller and faster memory between the processor and the DRAM, called Cache Memory which acts as a low-latency and high-

bandwidth storage, the data needed by the processor is first fetched into the cache, and all subsequent accesses to data items residing in the cache are serviced by the cache. In principle, if a piece of data is repeatedly used, the effective latency of this memory system can be reduced by the cache. The fraction of data references satisfied by the cache is called the cache hit ratio of the computation on the system.

3.6. Impact of Memory Bandwidth

The Memory bandwidth refers to the rate at which data [14, 15, 16, 17] can be moved between the processor and memory, moreover, it is determined by the bandwidth of the memory bus as well as the memory units, so one commonly used technique to improve memory bandwidth is to increase the size of the memory blocks, If the size of the memory block that a single memory request returns a contiguous block of four words is referred to as a cache line. The Conventional computers fetch two to eight words together into the cache. Let's see the Example 3.1 which introduce the effect of block size dot-product of two vectors. If the processor can fetch a four-word cache line every 100 cycles, assuming that the vectors are laid out linearly in memory, a single memory access fetches four consecutive words in the vector, two accesses can fetch four elements of each of the vectors, and eight FLOPs(four multiply-adds) can be performed in 200 cycles, so This corresponds to a FLOP every 25 ns, for a peak speed of 40 MFLOPS.

Note that increasing the block size from one to four words did not change the latency of the memory system. However, it increased the bandwidth four-fold. In this case, the increased bandwidth of the memory system enabled us to accelerate the dot-product algorithm which has no data reuse at all. Physically, the scenario illustrated in Example 3.1 corresponds to a wide data bus(4 words or 128 bits) connected to multiple memory banks. In practice, such wide buses are expensive to construct. In a more practical system, consecutive words are sent on the memory bus on subsequent bus cycles after the first word is retrieved.

3.7. Alternate Approaches for Hiding Memory Latency

Imagine sitting at your computer browsing [14, 15] the web during peak network traffic hours, the lack of response from your browser can be alleviated using one of three simple approaches are below

First issue we anticipate which pages we are going to browse ahead of time and issue requests for them in advance; second issue we open multiple browsers and access different pages in each browser, thus while we are waiting for one page to load, we could be reading others, or the third issues we access a whole bunch of pages in one go amortizing the latency across various accesses. The first approach is called prefetching, the second multithreading, and the third one corresponds to spatial locality in accessing memory words.

3.8. Speedup Factor

The rule to calculate the increasing speed by using multi processor is follow in the following Rule

$$S(p) = \frac{\text{Execution time using one processor (best sequential algorithm)}}{\text{Execution time using a multiprocessor with } p \text{ processors}} \frac{t_s}{t_p}$$

Rule 3.1- Increasing Speed Rule .

Where t_s is execution time on a single processor and t_p is execution time on a multiprocessor, $S(p)$ gives increase in speed by using multiprocessor. So Use best sequential algorithm with single processor system. The underlying algorithm for parallel implementation might be(and is usually) different.

3.9. Types of Parallel Computers

There are two principal types, [14, 16, 17] the first is shared memory multiprocessor and the second is distributed memory multicomputer.

Conventional Computer Consists of a processor executing a program stored in a (Main) memory . Each main memory location located by its address, the addresses start at 0 and extends to $2^b - 1$ when there are b bits(binary digits) in address. Natural way to extend single processor model is to have multiple processors connected to multiple memory modules, such that each processor can access any memory module as in the following figure 3.1

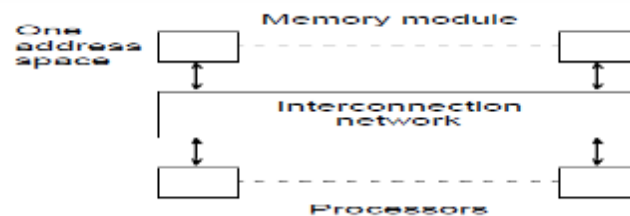


Figure 3.1 -Memory with One Address and Multi Processors.

Simplistic view of a small shared memory multiprocessor as in the following figure 3.2



Figure 3.2 - Shared Memory and Multiprocessors Representation

Some Examples as Dual Pentiums, Quad Pentiums, Multi-Core systems (ie Intel Core Duo) connect two CPUs together to the same die on the motherboard; a dual-core processor with two cores at 2GHz may perform very nearly as fast as a single core of 4GHz, and Intel core i7. Quad Pentium Shared Memory Multiprocessors as in the figure 3.3

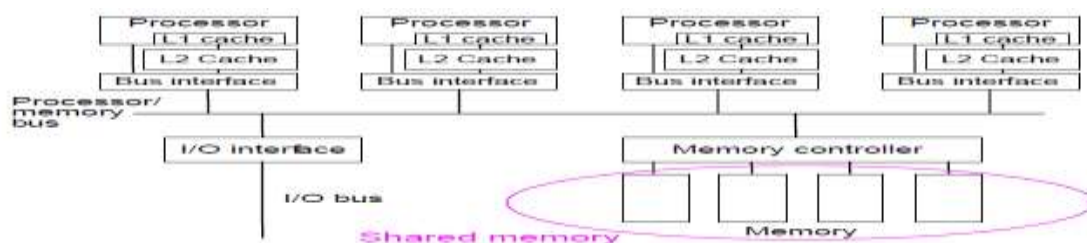


Figure 3.3 - QUAD Pentium Shared Memory Multiprocessors

3.10. Some Techniques to Implement Parallel Program

First technique is Threads programmer decomposes program into individual parallel sequences(threads), each being able to access variables declared outside the threads. For example Pthreads. Second technique is Sequential programming language with preprocessor compiler directives to declare shared variables and

specify parallelism. For example OpenMP. The third Sequential programming is language with added syntax to declare shared variables and specify parallelism.

3.11. Message-Passing Multicomputer

Complete computers connected [14, 15, 18, 19] through an interconnection network so the message passing is as follow in the figure 3.4

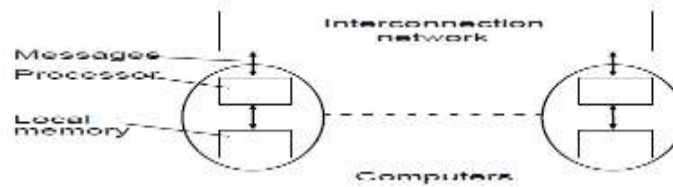


Figure 3.4 - Message Passing Representation.

3.12. Interconnection Networks

Limited and exhaustive interconnections, 2- and 3-dimensional meshes, Hypercube, using Switches as Crossbar and Trees, and Multistage interconnection networks

3.13. Distributed Shared Memory

Making the main memory of a group of interconnected computers [15] look as though a single memory with single address space. This can use shared memory programming techniques as shown in the figure 3.5

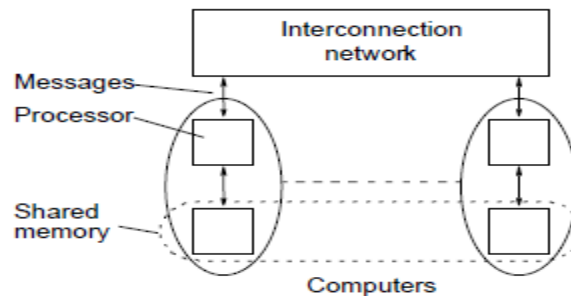


Figure 3.5 - Interconnected Network and Shared Memory Representation.

3.14. Flynn's Classifications

Flynn (1966) created a classification [14, 16, 17] for computers based upon instruction streams and data streams:

3.14.1. Single Instruction Stream

Single instruction stream-single data stream(SISD) computer Single processor computer which is Instructions operate upon a single stream of data items.

3.14.2. Multiple Instruction Stream

Multiple Data Stream (MIMD) [15] Computer operate on multiprocessor system and has General-purpose each processor has a separate program and one instruction stream is generated from each program for each processor, each instruction operates upon different data.

3.14.3. Single Instruction Stream-Multiple Data Stream (SIMD)

It is a specially designed computer, a single instruction stream from a single program, but multiple data streams exist. Instructions from program broadcast to more than one processor. Each processor executes the same instruction in synchronism, but using different data. It is developed because a number of important applications that mostly operate upon arrays of data were used.

3.14.4. Multiple Program Multiple Data (MPMD) Structure

Within the MIMD classification, each processor will have its own program to execute as in the follow figure



Figure 3.6 - MIMD Representation.

3.14.5. Single Program Multiple Data (SPMD) Structure

A single source program is written and each processor executes its personal copy of this program, although independently and not in synchronism. Source program can be constructed so that parts of the program are executed by certain computers and not others depending upon the identity of the computer.

3.15. Networked Computers as a Computing Platform

A network of computers became a very attractive alternative to expensive supercomputers and parallel computer systems for high-performance computing in early 1990's. Very high performance workstations and PCs are readily available and at low cost, the latest processors can be easily be incorporated into the system as they became available and The existing software can be easily used or modified.

3.16. Software Tools for Clusters

Based upon Message Passing Parallel Programming

First Parallel Virtual Machine (PVM) is developed in the late 1980's, and became very popular, and then Message-Passing Interface (MPI) is standard defined in the 1990s. Both provide a set of user-level libraries for message passing. They are used with regular programming languages like C and C++. As hardware the using of a group of interconnected "commodity" computers achieving high performance with low cost. Typically using commodity interconnects with high speed Ethernet, and Linux OS.

3.17. Message-Passing Computing

Two primary mechanisms are needed:

1. The method of creating separate processes for execution on different computers
2. The method of sending and receiving messages

3.17.1. Multiple Program and Multiple Data (MPMD)

MPM model is shown in figure 3.7

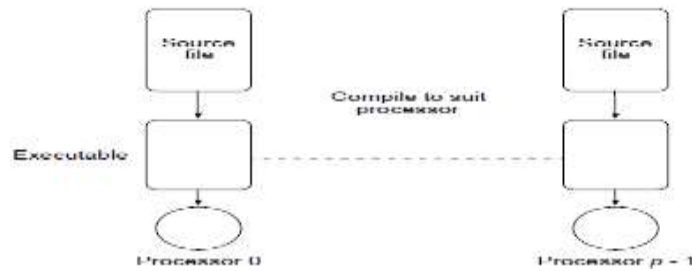


Figure 3.7 - MPMD Representation.

3.17.2. Single Program Multiple Data (SPMD) Model

Different processes are merged into one program. Control statements select different parts for each processor to execute. All executables started together- *static process* creation as in the figure 3.8

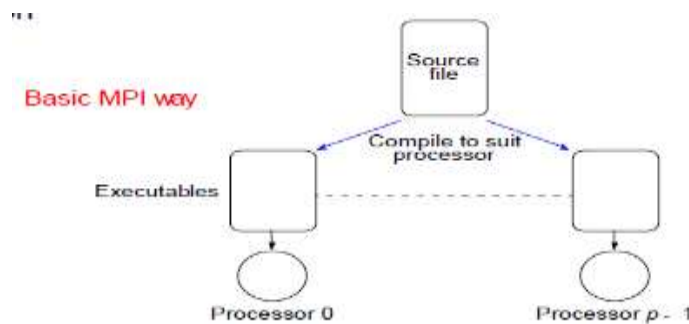


Figure 3.8 - SPMD Representation

3.17.3. Multiple Program Multiple Data (MPMD) Model

Separate programs for each processor. One processor executes master process. Other processes started from within the master process(process1 is a master process and process2 is a slave process) *dynamic process* creation as shown in the following figure 3.9

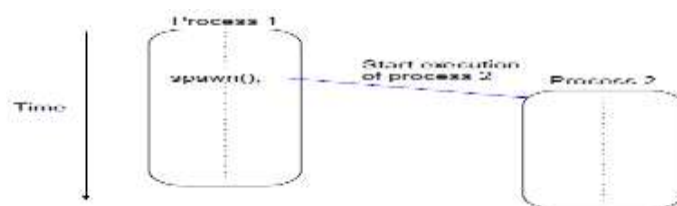


Figure 3.9 - MPMD Model .

3.17.4. Basic “Point-to-Point” Send and Receive Routines

Passing a message between processes using send() and recv() library calls as shown in the figure 3.10

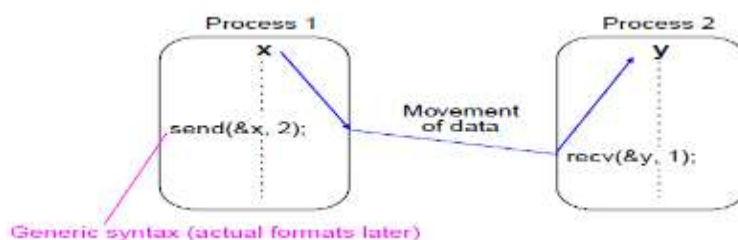


Figure 3.10 - Message Passing between Processes .

3.17.5. Synchronous Message Passing Routines

The Synchronous Message Passing Routines [14, 15, 16, 17] are *Synchronous send routine* which Waits until complete message can be accepted by the receiving process before sending the message, and *Synchronous receive routine* which Waits until the message it is expecting arrives. However Synchronous routines basically perform two actions: They transfer data and they synchronize processes.

3.17.6. MPI Definitions of Blocking and Non-Blocking

The MPI Definitions of Blocking and Non-Blocking are **Blocking** which returns after their local actions are complete, though the message transfer may not have been completed, and **Non-blocking** which returns immediately. It assumes that data storage used for transfer is not modified by subsequent statements prior to being used for transfer, and it is left to the programmer to ensure this. These terms may have different interpretations in other systems.

3.18. Embarrassingly Parallel Computations

A computation can be divided into a number of completely independent parts, each of which can be executed by a separate process as shown in the figure 3.11

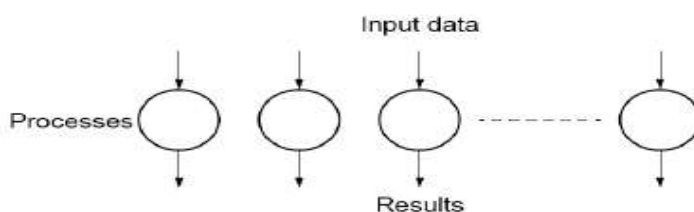


Figure 3.11- Parallel Computation within Independent Parts

No communication or very little communication exists between the processes. Each process can do its tasks without any interaction with other processes.

Practical embarrassingly parallel computation with static process creation and master-slave approach and starting together as shown in the following figure 3.12

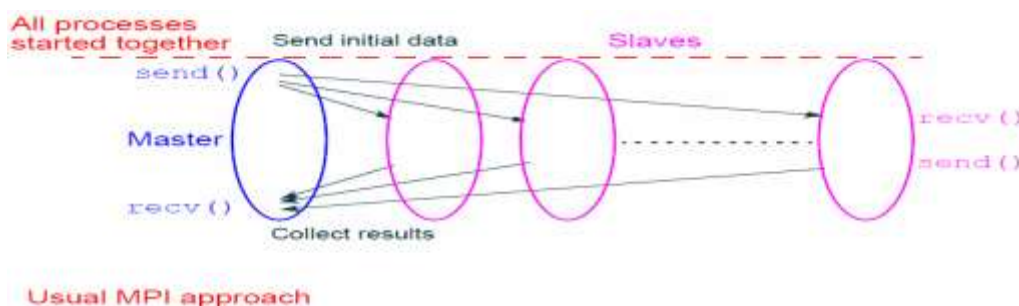


Figure 3.12- Parallel Computation Master Slave Approach .

Practical embarrassingly parallel computation with dynamic process creation and master-slave approach and Master starts initially as shown in the following figure 3.13

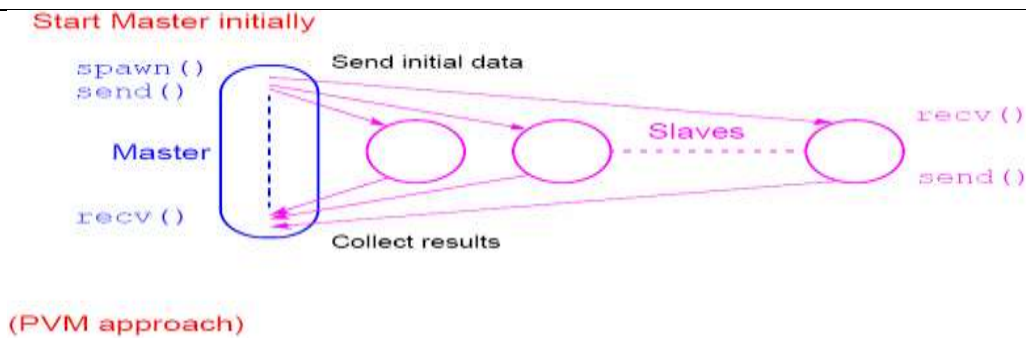


Figure 3.13 - Parallel Computation Master Slave Approach.

3.19. Parallel Programming in the .NET Framework 4

Many personal computers and workstations have two or four cores (CPUs) that enable multiple threads to be executed simultaneously. Computers in the near future are expected to have significantly more cores. To take advantage of the hardware of today and tomorrow, you can parallelize your code to distribute work across multiple processors. In the past, parallelization required low-level manipulation of threads and locks. Visual Studio 2010 and the .NET Framework 4 enhance the support for parallel programming by providing a new runtime, new class library types, and new diagnostic tools. These features simplify parallel development so that you can write efficient, fine-grained, and scalable parallel code in a natural idiom without having to work directly with threads or the thread pool. The following illustration provides a high-level overview of the parallel programming architecture in the .NET Framework 4 and is shown in the figure 3.14

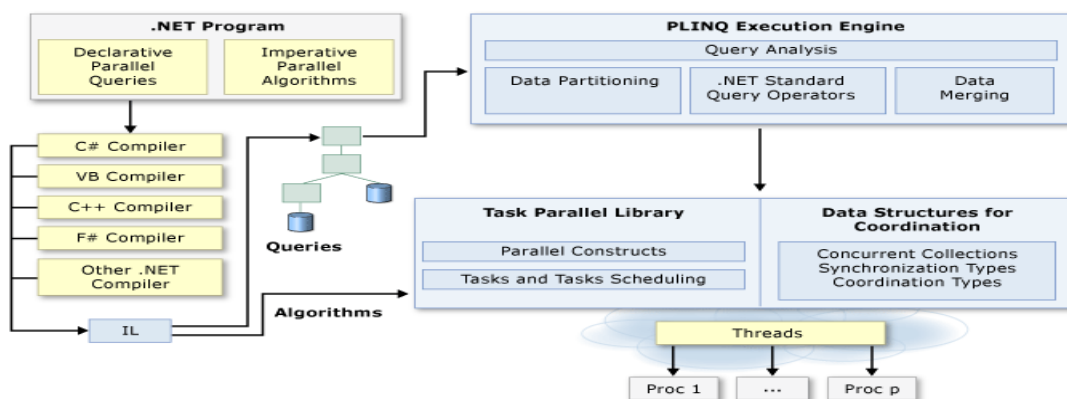


Figure 3.14 - Architecture of .NET Framework 4.

3.20. Thread

Visual basic supports parallel execution [20, 21] of code through multithreading. A thread is an independent execution path, able to run simultaneously with other threads.

A visual basic 2010 *client* program (Console, WPF, or Windows Forms) starts in a single thread created automatically by the CLR and operating system (the “main” thread), and is made multithreaded by creating additional threads. Here’s a simple example and its output in the figure 3.17:

3.20.1. Pseudo Code Thread1

Program Thread

```

Sub main()
Var t : Thread(AddressOf WriteY) ' Kick off a new thread ' running WriteY()
    Start(the tread t)
For i=0 to 1000
Write( "x")
End for
End sub

Sub WriteY()
    For i As Integer = 0 To 1000
        Write("y")
    End for
End sub

End Program

```

The main thread creates a new thread t on which it runs a method that repeatedly prints the character “y”. Simultaneously, the main thread repeatedly prints the character “x” as shown in the figure 3.15



Figure 3.15 - Execution of the Source Code Thread1.

Once started, a thread's `IsAlive` property returns true, until the point where the thread ends. A thread ends when the delegate passed to the Thread's constructor finishes executing. Once ended, a thread cannot restart. The CLR assigns each thread its own memory stack so that local variables are kept separate. In the next example, we define a method with a local variable, then call the method simultaneously on the main thread and a newly created thread

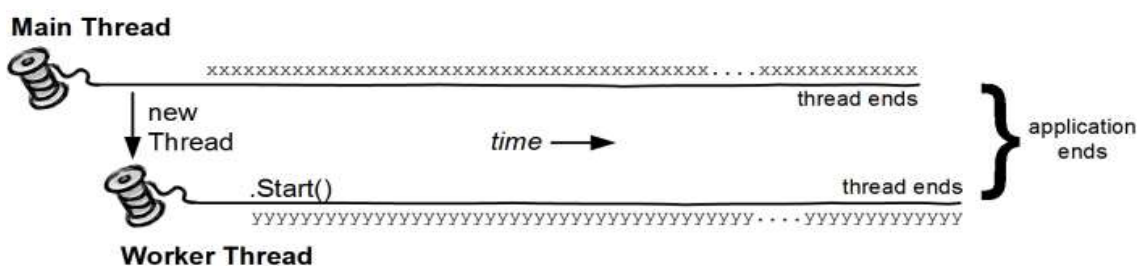


Figure 3.16 - Main Thread and Worker Thread Representation.

3.20.2. The Join and Sleep (Source Code2)

You can wait for another thread to end by calling its Join method. For example

```
Program Thread2
```

```
Sub Thread()
```

```
Var t : Thread= New Thread(AddressOf go)
```

```
Start(the tread t)
```

```
t.join()
```

```
WriteLine("Thread t has ended! program done by wafaa bazzi")
```

```
    call go1()
```

```
End Sub
```

```
Sub go()
```

```
    For i As Integer = 0 To 10
```

```
        WriteLine("done")
```

```
    End for
```

```
End Sub
```

```
Sub go1()
```

```
    For i As Integer = 0 To 10
```

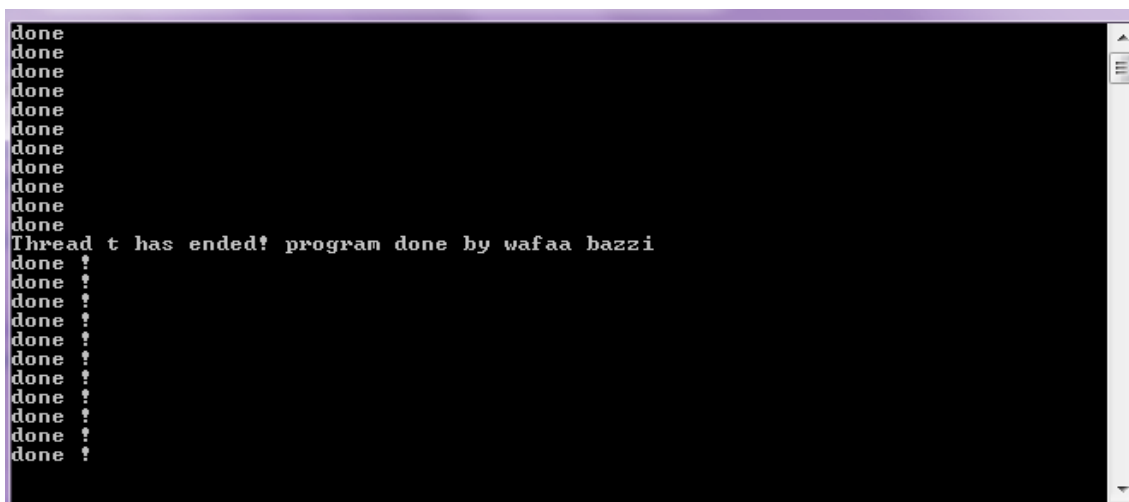
```
        WriteLine("done !")
```

```
    End for
```

```
End SubProgram
```

```
End Program
```

This prints “done” 10 times, followed by “Thread t has ended!”, after that call the another thread and prints "done !" 10 times. You can include a timeout when calling Join, either in milliseconds or as a TimeSpan. It then returns true if the thread ended or false if it timed out the result is shown in the figure 3.17



```
done
done
done
done
done
done
done
done
done
done
done
Thread t has ended! program done by wafaa bazzi
done !
done !
done !
done !
done !
done !
done !
done !
done !
done !
```

Figure 3.17 - Execution of the Second Thread Code

Another thread which shows how the two threads work together is shown in the following figure 3.18



Figure 3.18 - Execution of the Third Thread Code

3.20.3. Pseudo Code 3

Program Thread3

Sub main()

Var t : Thread(AddressOf WriteY) ' Kick off a new thread ' running WriteY()

Start(the tread t) ' Simultaneously, do something on the main thread.

For i=0 to 1000

Write("x wafa ")

End for

End sub

Sub WriteY()

For i As Integer = 0 To 1000

Write("y bazzi ")

End for

End sub

End Program

3.21. Threading Strategy Works

Multithreading is managed [20, 21] internally by a thread scheduler, a function the CLR typically delegates to the operating system. A thread scheduler ensures that all active threads are allocated the appropriate execution time, and that threads that are waiting or blocked(for instance, on an exclusive lock or on user input) do not consume CPU time.

On a single-processor computer, a thread scheduler performs *time-slicing*— rapidly switching execution between each of the active threads. Under Windows, a time-slice is typically in the tens-of-milliseconds region— much larger than the CPU overhead in actually switching context between one thread and another.

On a multi-processor computer, multithreading is implemented with a mixture of time-slicing and genuine

concurrency, where different threads run code simultaneously on different CPUs.— .A thread is said to be *preempted* when its execution is interrupted due to an external factor such as time-slicing. In most situations, a thread has no control over when and where it's preempted.

3.22. Threads vs Processes

A thread is analogous to the operating system process [20, 21] in which your application runs. Just as the processes run in parallel on a computer, threads run in parallel *within a single process*. Processes are fully isolated from each other; threads have just a limited degree of isolation. In particular, threads share(heap) memory with other threads running in the same application. This, in part, is why threading is useful: one thread can fetch data in the background, for instance, while another thread can display the data as it arrives.

3.23. Threading's Uses and Misuses

Multithreading has many uses; here are the most common:

3.23.1. Maintaining a Responsive User Interface

By running time-consuming tasks on a parallel “worker” thread, the main UI thread is free to continue processing keyboard and mouse events.

3.23.2. Making Efficient Use of an Otherwise Blocked CPU

Multithreading is useful when a thread is awaiting a response from another computer or piece of hardware. While one thread is blocked while performing the task, other threads can take advantage of the otherwise unburdened computer.

3.23.3. Parallel Programming

A code that performs intensive calculations can execute faster on multicore or multiprocessor computers if the workload is shared among multiple threads in a “divide-and-conquer” strategy.

3.23.4. Speculative Execution

On multicore machines, you can sometimes improve performance by predicting something that might need to be done, and then doing it ahead of time. [LINQPad](#) uses this technique to speed up the creation of new queries. A variation is to run a number of different algorithms in parallel that all solve the same task.

3.23.5. Allowing Requests to be Processed Simultaneously

On a server, client requests can arrive concurrently and so need to be handled in parallel(the .NET Framework creates threads for this automatically if you use Web Services, or Remoting). This can also be useful on a client(e.g., handling peer-to-peer networking — or even multiple requests from the user).

The biggest problem is multithreading which can increase complexity. Having lots of threads does not in and of itself create much complexity; it's the interaction between threads(typically via shared data) that does. This applies whether or not the interaction is intentional, and can cause long development cycles and nonreproducible bugs. For this reason, it pays to keep interaction to a minimum, and to stick to simple and proven designs wherever possible.

3.24. Parallel.for and Parallel.ForEach Using Visual Basic 2010

A [ForEach](#) loop works [\[22\]](#) like a [For](#) loop. The source collection is partitioned and the work is scheduled on multiple threads based on the system environment. The more processors on the system, the faster the parallel method runs. For some source collections, a sequential loop may be faster, depending on the size of the source, and the kind of work being performed. The first program example that I designed is about convert several images 180 degrees in parallel way at the same time

3.25. Pseudo Code Parallel.ForEach

' Design a Simple Parallel.ForEach Loop , Aust Analyst and Programmer Wafaa Bazzi

Program Main()

```
Dim files As String() = System.IO.Directory.GetFiles("Drectory", "*.jpg")
```

```
Dim newDir As String = "newDirectory"
```

```
System.IO.Directory.CreateDirectory(newDir)
```

```
Parallel.ForEach(files, Sub(currentFile)
```

```
'The greater the speedup compared to a sequential foreach loop.
```

```
Dim filename As String = System.IO.Path.GetFileName(currentFile)
```

```
Dim bitmap As New System.Drawing.Bitmap(currentFile)
```

```
bitmap.RotateFlip(System.Drawing.RotateFlipType.Rotate180FlipNone)
```

```
bitmap.Save(System.IO.Path.Combine(newDir, filename))
```

```
' Peek behind the scenes to see how work is parallelized.
```

```
Console.WriteLine("Processing {0} on thread {1}", filename, Thread.CurrentThread.ManagedThreadId)
```

```
End Sub)
```

```
End program
```

The original images are in the figure 3.19



Figure 3.19 - Multiple-Image

When you run the parallel program , run by using multithread each thread is for converting one image 180 degree in short time , the program converts all of them

```
file:///C:/thesis/program/parllel.for/ForEachDemo/ForEachDemo/bin/Debug/ForEachDemo.EXE
Processing Jellyfish.jpg on thread 12
Processing Hydrangeas.jpg on thread 11
Processing Lighthouse.jpg on thread 14
Processing Penguins.jpg on thread 15
Processing Koala.jpg on thread 13
Processing Desert.jpg on thread 10
Processing Tulips.jpg on thread 16
Processing Chrysanthemum.jpg on thread 9
AUST Programmer Wafaa Bazzi,Parallel Programming , 20011
Processing complete. Press any key to exit.
```

Figure 3.20 - FOR.Each Execution Schedule.

The result of converting are in the following figure 3.21



Figure 3.21 – Invert Result Image..

3.26. Parallel.For

The second program uses `parallel.for` to multiply 2 matrices, the first uses sequential method and the second uses `parallel.for`, compare the time consumed upon using sequential method and with that upon using parallel method. Remark, the efficiency of using `parallel.for` when having big-length matrix. The *Pseudo code is bellow*

```
Sub MultiplyMatricesSequential( matA: Double(,),matB: Double(,),result:Double(,))
```

```
    For i As Integer = 0 To matA.GetLength() - 1
        For j As Integer = 0 To matBCols(GetLength)- 1
            For k As Integer = 0 To matACols(GetLength) - 1
                result(i, j) += matA(i, k) * matB(k, j)
            endloops
        endloops
    endloops
```

3.27. The Parallel Algorithm

```
Sub MultiplyMatricesParallel(matA: Double(,),matB: Double(,), result: Double(,))
```

```
    ' Parallelize the outer loop to partition the source array by rows.
```

```
    Parallel.For(0, matA.GetLength(), Sub(i)
        For j As Integer = 0 To matBCols(GetLength) - 1
```

```
        ' Use a temporary to improve parallel performance.
```

```
            var temp: Double = 0
```

```
            For k As Integer = 0 To matA.GetLength() - 1
```

```
                temp += matA(i, k) * matB(k, j)
```

```
            Next
```

```
            result(i, j) += temp
```

```
        Next
```

```
    End Sub)
```

```
End Sub
```

```
Sub Main(ByVal args As String())
```

```
    enter the column dimension of first array colCount
```

```
    enter the row dimension of first array rowCount
```

```
    enter the column dimension of second array colCount2
```

```
    Generate-Random-Matrix m1 (rowCount, colCount)
```

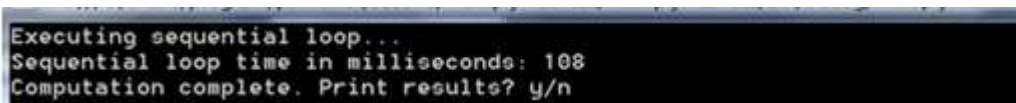


```

Generate-Random-Matrix m2 (colCount, colCount2)
Var result (rowCount - 1, colCount2 - 1)
' First do the sequential version.
Var stopwatch As New Stopwatch()
stopwatch.Start()
MultiplyMatricesSequential(m1, m2, result)
stopwatch.[Stop]()
WriteLine("Sequential loop time in milliseconds:", stopwatch. Milliseconds)
Print(rowCount, colCount2, result)
' Reset timer and results matrix.
stopwatch.Reset()
stopwatch.Start()
MultiplyMatricesParallel(m1, m2, result)
stopwatch.[Stop]()
WriteLine("Parallel loop time in milliseconds: ", stopwatch.Milliseconds)
Print(rowCount, colCount2, result)
End Sub

```

The Result of The first image(Figure 3.22) command to Put column dimension and row dimension of the first array, the program puts 2 randomized arrays from your dimension. You can put big dimension or try to put medium sizes to see the multiplied array result the third image and after calculating the result, show the sequential time in milliseconds 108



```

Executing sequential loop...
Sequential loop time in milliseconds: 108
Computation complete. Print results? y/n

```

Figure 3.22 – Result Screen.

The second image shows the matrix result, and the parallel time in this case is 50 approximately the half time.

A remark is that this example shows how to use the simplest overload of the Parallel.For method to compute the product of two matrices. It also shows how to use the System.Diagnostics.Stopwatch class to compare the performance of a parallel loop with a non-parallel loop. Then the third image shows the matrix result, and the matrix result verifies that the output is the same as in sequential method the results are shown below Figure 3.23 and Figure 3.24

```

ROW 97:
221912 226813 230497 219722 206272 213465 206750 222636 226651 217898 210771 227657 209279 206482 213794 228091 215698 218566 205076 23
0619 211964 242535 245345 224436 232997 217168 216699 221937 209630 215328 222103 202254 211547 231444 235872 207534 222356 239346 2163
56 219227 243820 244401 224036 211703 214962 233324 196314 238117 202659 194471 208811 209933 218173 219083 248717 195194 217263 227833
210000 237823 233713 225123 228811 224012 224795 220815 221047 216448 203838 230699 221010 216502 216281 214106 215862 214709 207860 2
30490 216913 233690 207352 218048 228772 235633 197785 223469 216585 222360 214656 237435 231102 239276 198411 216279 223152 239831 206
954 220559 223741 209221
ROW 98:
267818 281446 279321 272927 274443 247514 254980 269222 269117 262309 252893 280835 234226 230708 250678 267933 245316 246662 245753 28
2680 255066 303392 307829 282619 287468 259918 267460 266526 251211 265488 259546 235717 258488 296703 263926 253622 270561 297186 2609
32 274352 285728 285053 268217 265037 268732 261470 238635 294399 232826 238591 236202 252800 278445 264989 278821 235726 251496 284251
242107 269483 290460 286058 256896 269693 277784 259525 268082 265247 247527 276885 263582 256673 275212 243970 255616 272533 247546 2
88836 263998 272065 246903 274247 273531 266795 252651 267471 251473 264804 267381 273089 275434 271911 252303 266294 264233 292221 233
333 252204 272088 251116
ROW 99:
253457 267220 285182 254558 265095 263703 239486 267280 246089 236545 266836 269097 237617 246192 231671 263715 255647 257447 252342 27
0029 241735 291433 283948 265748 265121 274991 253321 254276 246458 256732 260454 220962 235599 287627 267050 239571 280691 294086 2602
84 258551 274360 284514 237552 277875 251730 254656 248983 269542 231472 232782 240293 247322 253965 273397 273302 221832 271668 283805
231806 261661 296582 284975 268683 269191 261089 253793 250574 246235 234634 277156 259868 236499 258354 252604 253352 262079 255370 2
82218 268845 248491 268067 276263 252492 249905 233416 263116 249218 257463 289592 273884 253489 265933 230844 282780 260571 286470 244
837 259626 270731 228838
Executing parallel loop...
Parallel loop time in milliseconds: 50
Computation complete. Print results? y/n

```

Figure 3.23 - Parallel.For Screen Result.

```

53824 239503 251844 222650 262062 227506 234781 224598 229051 234499 244078 239609 242796 239695 239774 219071 232963 236877 259184 214
004 227484 244114 216988
ROW 97:
221912 226813 230497 219722 206272 213465 206750 222636 226651 217898 210771 227657 209279 206482 213794 228091 215698 218566 205076 23
0619 211964 242535 245345 224436 232997 217168 216699 221937 209630 215328 222103 202254 211547 231444 235872 207534 222356 239346 2163
56 219227 243820 244401 224036 211703 214962 233324 196314 238117 202659 194471 208811 209933 218173 219083 248717 195194 217263 227833
210000 237823 233713 225123 228811 224012 224795 220815 221047 216448 203838 230699 221010 216502 216281 214106 215862 214709 207860 2
30490 216913 233690 207352 218048 228772 235633 197785 223469 216585 222360 214656 237435 231102 239276 198411 216279 223152 239831 206
954 220559 223741 209221
ROW 98:
267818 281446 279321 272927 274443 247514 254980 269222 269117 262309 252893 280835 234226 230708 250678 267933 245316 246662 245753 28
2680 255066 303392 307829 282619 287468 259918 267460 266526 251211 265488 259546 235717 258488 296703 263926 253622 270561 297186 2609
32 274352 285728 285053 268217 265037 268732 261470 238635 294399 232826 238591 236202 252800 278445 264989 278821 235726 251496 284251
242107 269483 290460 286058 256896 269693 277784 259525 268082 265247 247527 276885 263582 256673 275212 243970 255616 272533 247546 2
88836 263998 272065 246903 274247 273531 266795 252651 267471 251473 264804 267381 273089 275434 271911 252303 266294 264233 292221 233
333 252204 272088 251116
ROW 99:
253457 267220 285182 254558 265095 263703 239486 267280 246089 236545 266836 269097 237617 246192 231671 263715 255647 257447 252342 27
0029 241735 291433 283948 265748 265121 274991 253321 254276 246458 256732 260454 220962 235599 287627 267050 239571 280691 294086 2602
84 258551 274360 284514 237552 277875 251730 254656 248983 269542 231472 232782 240293 247322 253965 273397 273302 221832 271668 283805
231806 261661 296582 284975 268683 269191 261089 253793 250574 246235 234634 277156 259868 236499 258354 252604 253352 262079 255370 2
82218 268845 248491 268067 276263 252492 249905 233416 263116 249218 257463 289592 273884 253489 265933 230844 282780 260571 286470 244
837 259626 270731 228838
Press any key to exit.

```

Figure 3.24 - Parallel.For Final Result.

In short Parallel [23] computing is a simple idea. The human brain is well versed in performing tasks in parallel; however, 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 matter of minutes that would normally take hours or days on a single processor. The most basic concept behind parallel computing is the distribution of the workload between the individual processors that are

working together to perform computations. For example, consider the problem of matrix addition, which is an embarrassingly parallel application. Embarrassingly parallel problems are those that require dividing up the work and having each processor operate on their portion as though it were a sequential algorithm. Suppose we have a parallel machine with p processors, and we want to add two matrices with p elements each. This problem only requires that we distribute the two corresponding elements from each matrix to their own processor, let the processor add the elements, and then collect them into a solution matrix, as illustrated in Figure 3.25

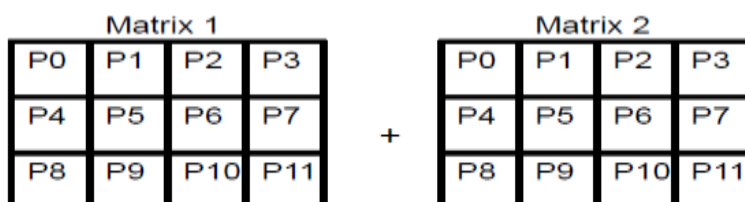


Figure 3.25 - Parallel Matrix Addition Screen.

Each processor, P_n , gets an element from each matrix. The ideal theoretical parallel computer consists of p processors with an infinite amount of memory. With such resources available, we can divide the workload between p processors to the point where each processor works with the smallest possible piece of data. In practice however, there does not exist such a computer. Efficient and proper use of a parallel computer involves careful workload distribution between processors. There are two basic architectures used with parallel computers: SIMD and MIMD. Single Instruction Multiple Data(SIMD) architecture is defined by a simultaneous execution of a single instruction on multiple pieces of data. For example, referring back to the matrix addition case, every processor has access to each element in both matrices and each processor executes the same instructions, adding the corresponding elements based on their processor ids. In Multiple Instruction Multiple Data(MIMD) architecture, each processor runs independently using its own set of instructions. Inter-processor communications allow for the transmission of data between processors and sending of signals for reporting on their status. In many cases this allows for the synchronization of the individual processors among their group, preventing some from moving forward before the others are ready. Often, synchronization is crucial for reliable execution of applications due to sharing of memory space. A race condition may occur if two processors require read/write access to the same data register. When this happens you cannot predict which processor gains access to the data first and therefore you can't guarantee the integrity of the data. Race conditions can be avoided through inter-processor communications. However how the processors communicate usually depends on the type of processor relationship being implemented. Washington and Lee's Beowulf cluster, The Inferno, is a 64 processor cluster that uses the MPI(Message Passing Interface) protocol for communication between processors. In clusters such as this one, there is no central shared memory repository for data; instead data is often distributed by a central processor and stored in each processor's local memory. Early parallel computers made use of a central shared memory, but as time has progressed it has become "difficult and expensive" to make larger machines with this form of

memory. Furthermore MPI is used in order to distribute data between processors and allow for inter-processor communication. With the message-passing model, we are able to work with very large sets of data without being restricted by the size of a global shared memory. The type of relationship between processors working in parallel determines the structure for the program itself. The master/slave relationship is a common paradigm for working in parallel. This consists of a sole “master” processor that presides over a group of “slave” processors. The master is responsible for organizing and distributing the data while the slaves operate on the data. Typically, the slaves, upon starting, signal the master that they are ready and waiting. As these signals are received, the master processor distributes the data among the slave processors. The master is only responsible for managing the data. In some cases, the master processor may clean up loose ends, but generally the slave processors do the majority of the computing, as illustrated in Figure 3.26. Once they have finished, the master collects the resulting data set from the slaves. The work-pool method for processor communication, as illustrated in Figure 3.27, is similar to the master/slave method, except that the slave processors make requests for smaller chunks of data from a “pool” managed and distributed by the master processor.

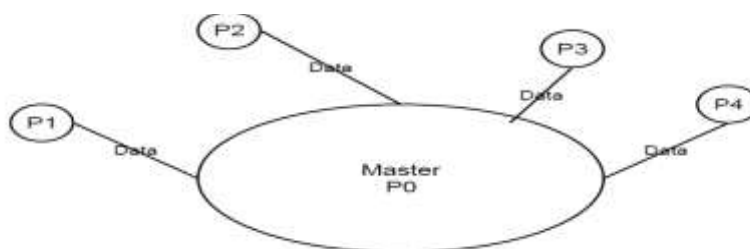


Figure 3.26 - Master/Slave Workload Distribution.

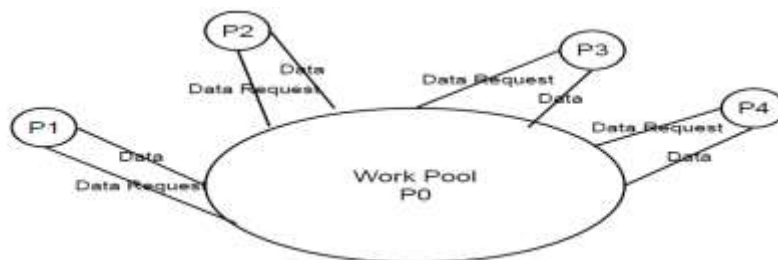


Figure 3.27 - Work Pool the Workload Distribution.

The key techniques for making computers compute “bigger problems faster” is to use multiple computers at once, this is called parallelism. For example if it takes 1000 hours for this program to run, if I use 100 computers it may take only 10 hours(Extend the CPU). If The computer can only handle a dataset that’s 2GB, so maybe if I use 100 computers I can deal with a 200GB dataset (Extend the memory).

CHAPTER 4

When processing images for a human observer, it is important to consider how images are converted into information by the viewer. Understanding visual perception helps during algorithm development. This chapter is firstly concerned with looking at the image and processing the image with self algorithms based on information explained in the chapter 2. Generating sequential methods to enhance image by using VB2010 (intensity, histogram, equalization spatial filter, edge detecting , Guassian Filtering), and making judgments based on visual results, memory complexity and time efficiency .

In relation to this thesis, Image processing can be a time consuming task and, leads to conversion to a parallel algorithm.

The fact that the Gray scale image enhancements are fundamental modules in image processing, optimizing the existing algorithms will result in an improvement of many methods and procedures. Gray scale Image processing applications are computationally and data intensive problems. With the recent advancement in multi-core architectures, these problems are gaining insight from a whole new perspective. The second issue to focus on in this chapter is to parallelize the sequential algorithms on a multi-core architecture in order to study the speedup of the parallel algorithm. We use Parallel.for to parallelize the algorithm. We also apply a similar method mentioned in the second chapter to enhance the gray-scale images(Sequential Method to Enhance Image) and we change it to be parallel method to enhance the image.

4.1 My Own Experimental Results by Using Intensity Method

The intensity of a pixel is expressed [5, 8, 7, 24] within a given range between a minimum and a maximum, inclusive. This range is represented in an abstract way as a range from 0 (total absence, black) and 1(total presence, white), with any fractional values in between. Given that the number of intensity levels is an integer power of 2 like $L=2^k$ such that k ranges from 1 to 7.

$L=256=2^8$ (true color), $L=2^7=128$,.....and $L=2^0$ (white color). Humans can discriminate about 128 levels(7 bit digitization needed)

4.1.1. Intensity Enhancement Pseudo Code

Input the intensity Level I

Select Case I

Case 1

```
for (int y=0; y<height; y++)
```

```
    for(int x=0; x<width; x++)
```

```
        c= input.getxy(x,y)
```

```
        Set the level r to be one of 2 levels (0 ,255)
```

```
        Set the bitmap output image to the new level
```

```
        End loops (x,y)
```

Case 2

```

for (int y=0; y<height; y++)
for(int x=0; x<width; x++)
  c1= input.getxy(x,y)
  r = the level of c1
  Set the level r to be one of 4 levels (0,64,128,255)
  Set the bitmap output image to the new level
End loops(x,y)

```

Case 3,4,5 or 6

```

for (int y=0; y<height; y++)
for(int x=0; x<width; x++)
  c1= input.getxy(x,y)
  r = the level of c1
  Set the level r to be one of (2 power I) levels ( between 0 and 255)
  Set the bitmap output image to the new level
End loops (x,y)
End Select

```

4.1.2. Intensity Results

Reduce intensity to level 3, so the first result is shown in the figure 4.1

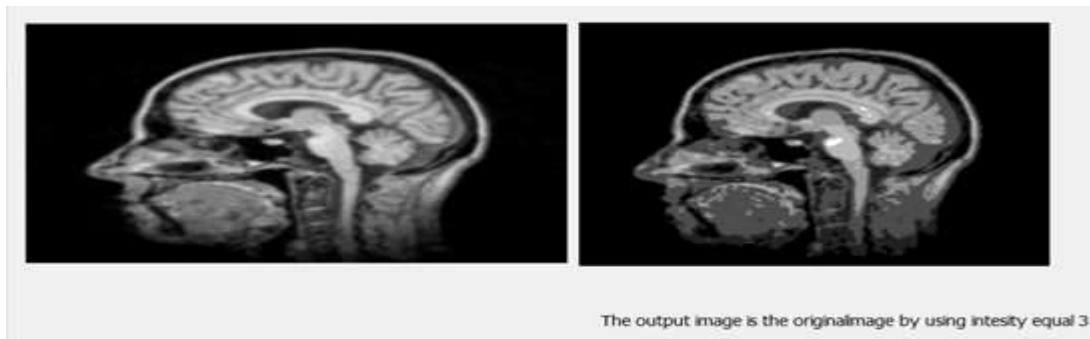


Figure 4.1- Intensity Level 3 Enhancement.

Reduce Intensity to level 1 and the result is shown in the figure 3.2

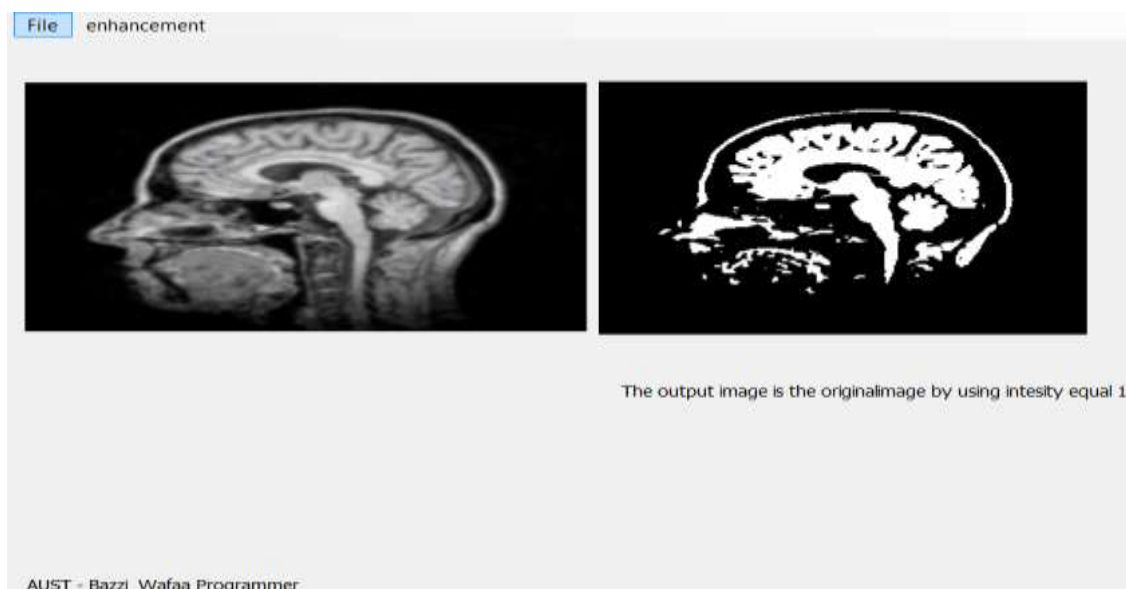


Figure 4.2 - Intensity Level1 Enhancement.

Another result with intensity 1 is shown in the figure 4.3

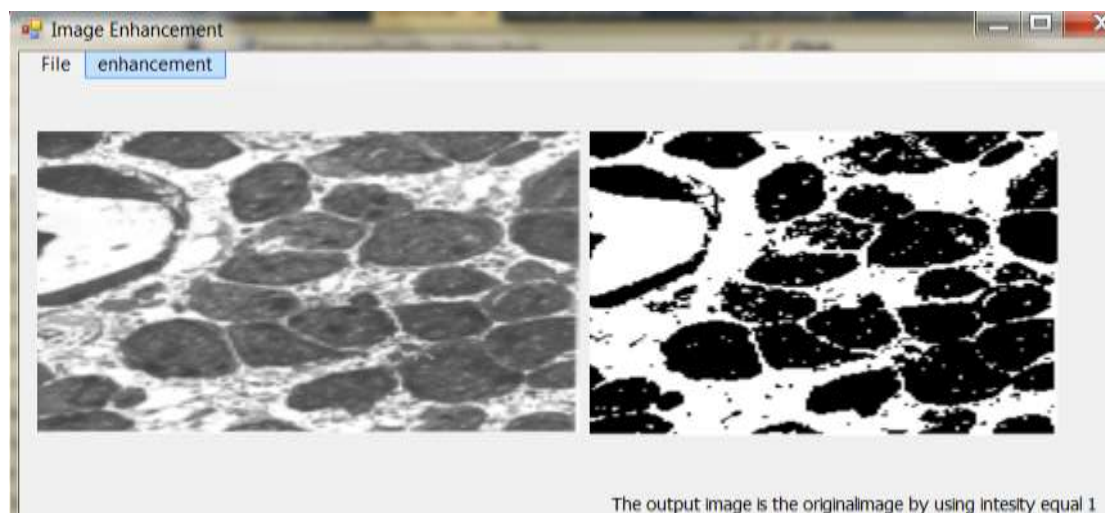


Figure 4.3 - Intensity level1 Enhancement.

4.2. My Own Histogram Solution

Often images have a limited range of colors, or [2, 5, 10] are lacking contrast. Enhancing the image can allow improving details. A histogram simply plots the frequency at which each grey level occurs from 0 (black) to 255 (white). So given this data, we can enhance the image automatically to expand the colors within the image to fill the entire 0-255 spectrum. This is done by histogram equalization. To do this, we need to calculate the cumulative frequencies within the image. It will be extremely easy to redistribute the colors across the entire spectrum.

4.2.1. My Own Histogram Results

Below in the figure 4.4 an image and its histogram and in the figure 4.5 equalized case of figure 4.4 and its histogram



Figure 4.4 - Histogram of Input Image



Figure 4.5 - Equalized Image of Figure 4.4 and its Histogram

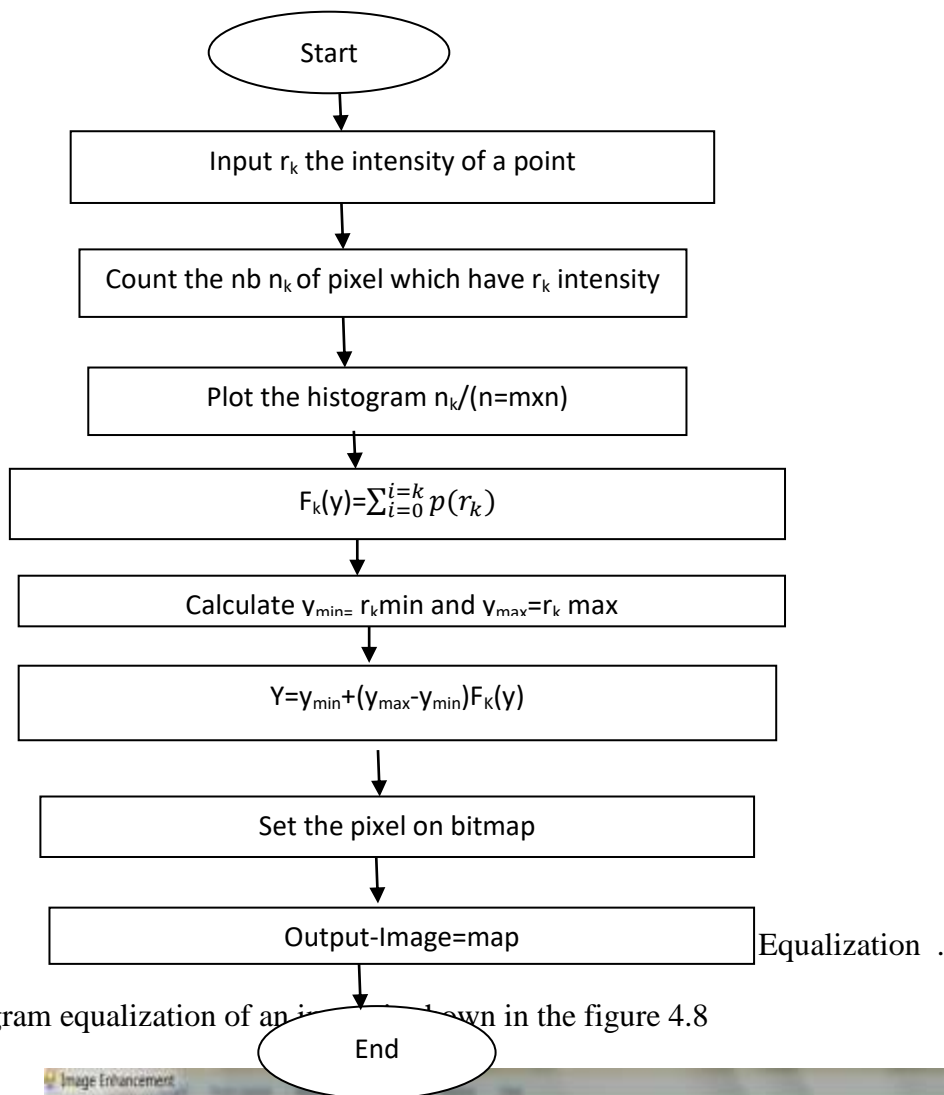
Some effects of Histogram Equalization are in the image on the top left, it has a histogram with a large peak. The histogram is shown on the right of it. The image on the bottom is the output of HE. Note how the peak has been stretched. Another result by using my program histogram of the image is shown in the figure 4.6



Figure 4.6 - Histogram of an Input Image

4.2.2. My histogram Flowchart

Process of Enhancement of an image by histogram equalization is shown by a flowchart in figure 4.7



Histogram equalization of an image is shown in the figure 4.8

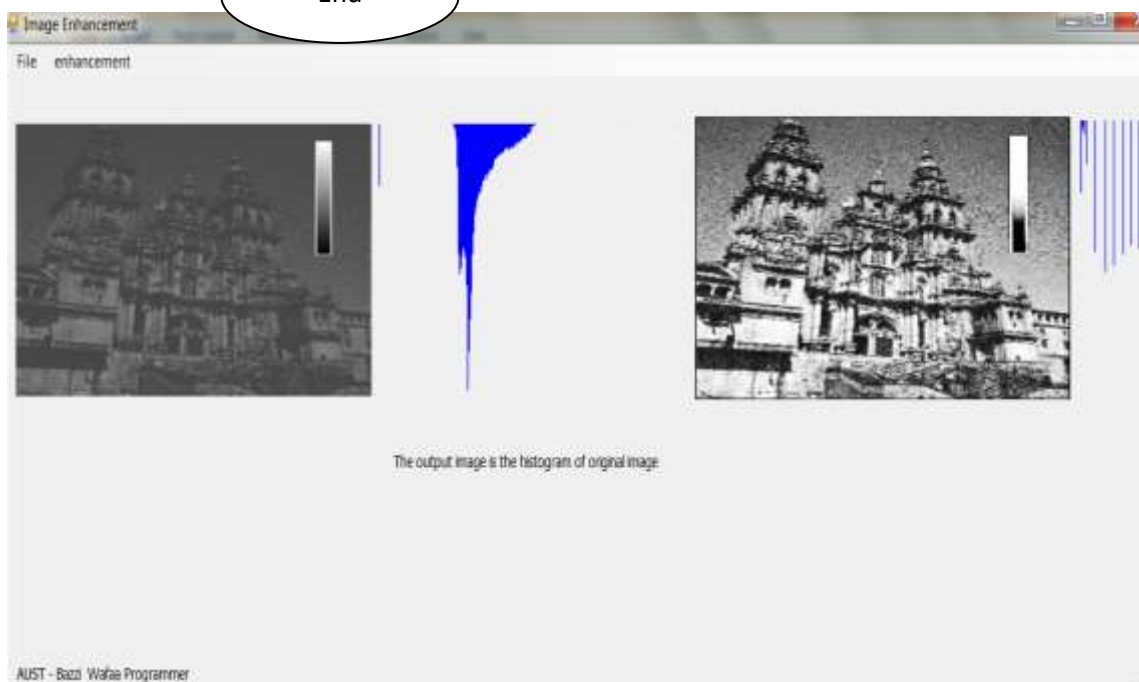


Figure 4.8 - Enhancement of an image by Equalization method and the two Histograms.

4.2.3. My Pseudo Code of Histogram Equalization

```

1. calculate histogram
loop over i ROWS of input image
loop over j COLS of input image
k = input_image[i][j]
hist[k] = hist[k] + 1
end loops( over j and i)
2. calculate the sum of hist
loop over i gray levels
sum = sum + hist[i]
sum_of_hist[i] = sum
end loop over i
3. transform input image to output image
area = area of image (ROWS x COLS)
Dm = number of gray levels in output image
loop over i ROWS
loop over j COLS
k = input_image[i][j]
out_image[i][j] = (Dm/area) x sum_of_hist[k]
end loops ( over j and i)

```

4.3 .Filtering

Previous methods of histogram equalizations is global. why we do not use local enhancement, most pixels in images look very similar to their neighbours, but the noise look very different from their neighbors, this method is called spatial spatial filter , Image noise results in pixels that look very different from their neighbors Generally, we support on the idea that most pixels in images look very similar to their neighbours this is true even at an edge; but, across the edge, the pixels are not, so the solution is not perfect, hence we try to enhance the smoothing, by underlining the edge ,and forcing pixels different to their neighbours (noise pixels) to look more like their neighbours this method and called Guassian Filtering.

4.3.1. Spatial filtering

We need to write a program to perform spatial filtering [2, 5] of an image by using spatial mask at 3x3 size with variables coefficients. To implement that , we use a neighborhood (typically small rectangle)and predefined operation that is performed on the image pixels encompassed by the neighborhood. Filtering creates a new pixel with coordinates equal to the coordinates of the center of the neighborhood, and whose value is the result of the filtering operation. Processed(filtered) images are generated as the center of the filter visits each pixel in the input image . The operation is called a linear spatial filter. At any point(x,y), the response, $g(x,y)$, of the filter coefficients and the image pixels encompassed by filter is shown in the figure 4.1 :

$$G(x, y) = w(-1, -1)f(x - 1, y - 1) + w(-1, 0)f(x - 1, y) + \dots + w(0, 0)f(x, y) + \dots + w(1, 1)f(x + 1, y + 1)$$

Equation 4.1- the Equation to Calculate the Output Image by Spatial Filter.

By example we use the linear spatial filtering in order to obtain output response of a smoothing linear spatial filter, which is simply the average of the pixels contained in the the neighborhood of the filter mask . The filters sometimes are called averaging filters.

The equation shown in figure 4.1 changed to the equation shown in the figure 4.2

$$G(x,y) = (w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \dots + w(0,0)f(x,y) + \dots + w(1,1)f(x+1,y+1)) / \sum \sum w(s, t)$$

Equation 4.2 - Equation of Average Filtering

4.3.2. Result of Enhancement of Image by Spatial Filter

The resultant image by applying averaging filters and using mask

1	1	1
1	1	1
1	1	1

is shown in the figure 4.9

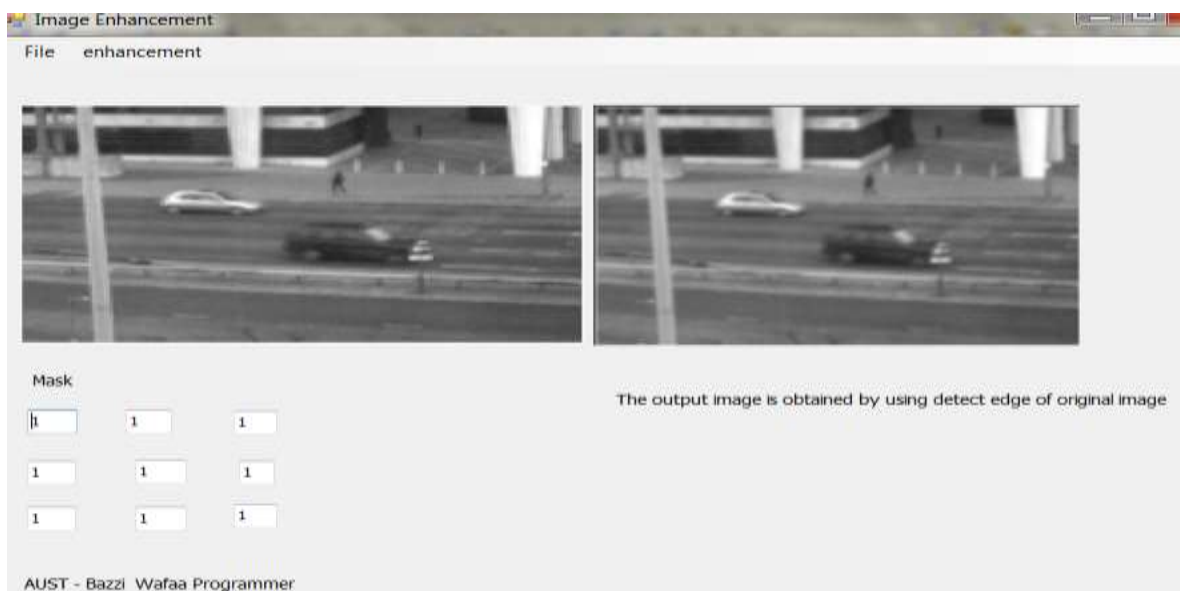


Figure 4.9 - Spatial Filter Enhancement Result

4.3.3. Edge Detecting Equation and Flowchart

The laplacian equation is the follow in the Equation 4.3:

$$\Delta^2(x, y) = w(0,0)f(x - 1, y - 1) + w(0,1)f(x - 1, y) + \dots + w(2,2)f(x + 1, y + 1)$$

Equation 4.3- Edge Detecting Equation.

The Flowchart [12] to detect the edge of an image is shown in the figure 4.13

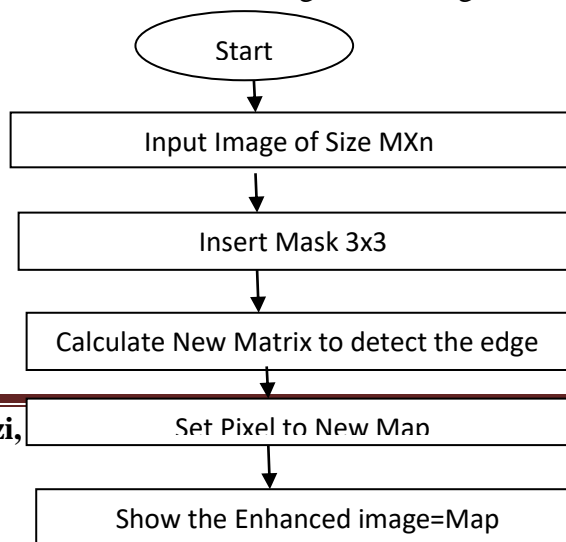


Figure 4.10 - Flowchart of Edge Detection

4.3.4. Edge detecting Results

If we apply Laplacian filter without scaling of the original image and the mask is

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

-1 We obtain the following result which is shown in figure 4.11

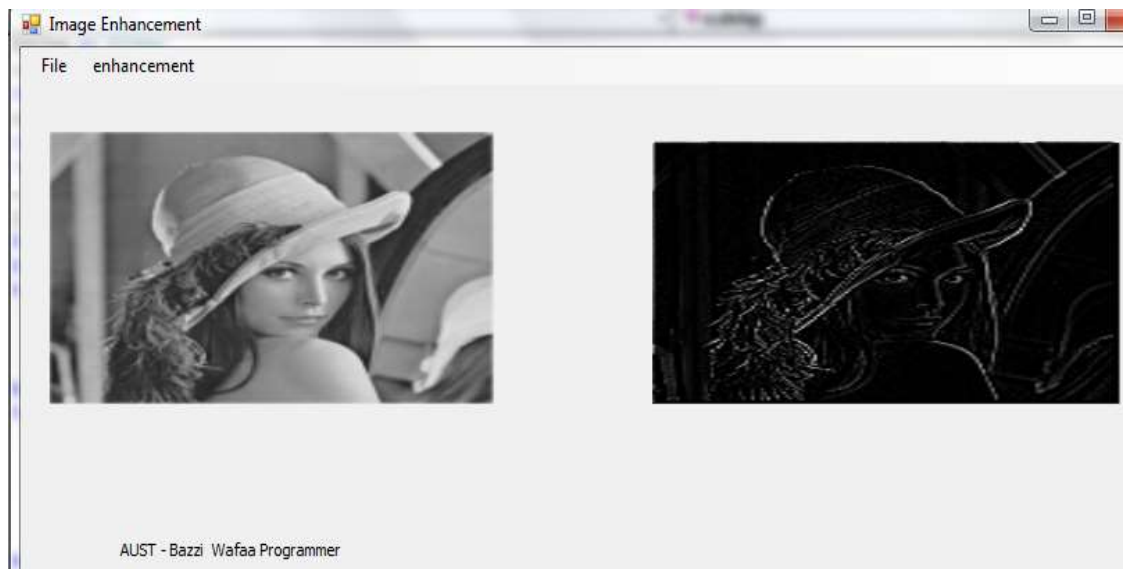


Figure 4.11 -Edge Detection of Lena Image.

Then if we apply Laplacian to detect edge by using the mask

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

So The resultant figure is shown in the figure 4.12

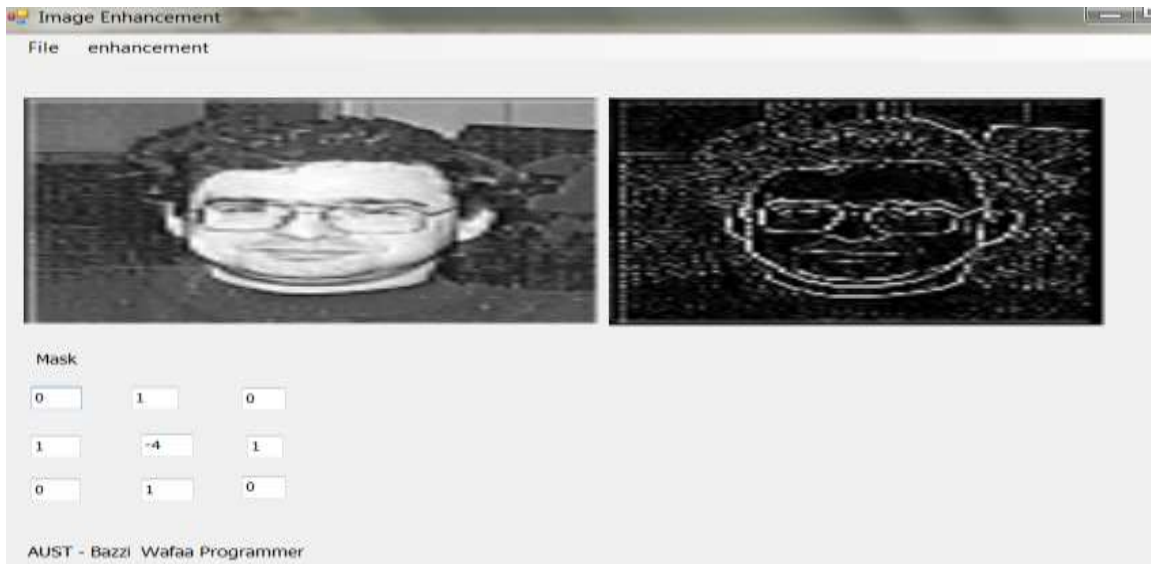


Figure 4.12 - Edge Detection Result.

Apply same mask on the earth image to obtain the result shown in the figure 4.13

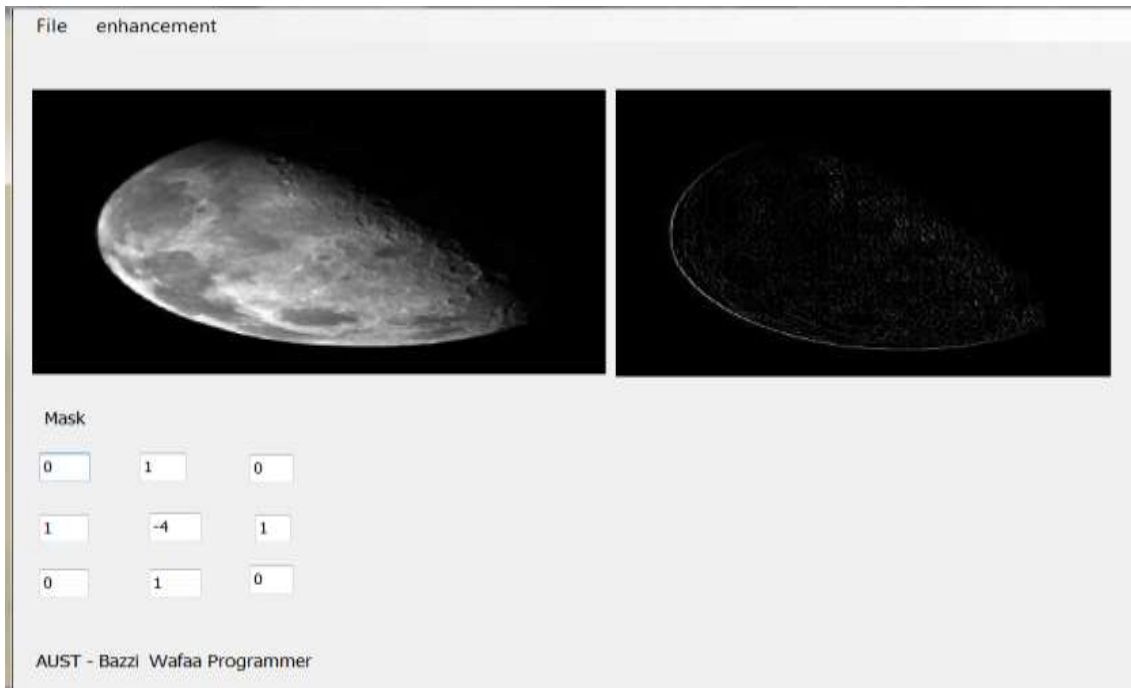


Figure 4.13 -Edge Detection of Earth Image.

If we apply the mask :

-1	-1	-1
-1	8	-1
-1	-1	-1

We obtain sharp edge as follow in the figure 4.14

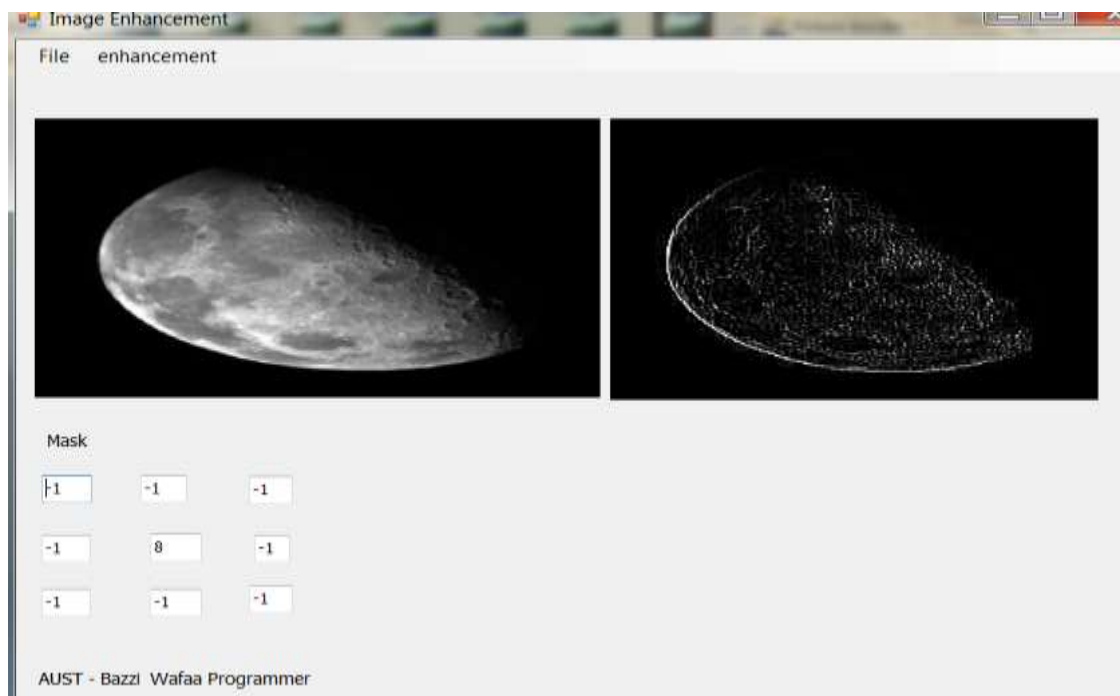


Figure 4.14 - Sharp Edge Detection of the Earth Image.

Large sections of these images are black because the laplacian contains both positive and negative values, and all negative value are clipped at 0 by the display .

4.3.5. Pseudo-Codes for Sobel Edge Etection Method

Input: A Sample Image

Output: Detected Edges

Step 1: Accept the input image

Step 2: Apply mask Gx,Gy to the input image

Step 3: Apply Sobel edge detection algorithm and the gradient

Step 4: Masks manipulation of Gx,Gy separately on the input image

Step 5: Results combined to find the absolute magnitude of the gradient

Step 6: the absolute magnitude is the output edges

4.4. My Experimental Results by Applying Gaussian Filter

Because the laplacian [5, 8, 12] is a derivative operator, its use highlights intensity discontinuities in an image and deemphasizes regions with slowly varying intensity levels. This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be recovered while still preserving the sharpening effect of the Laplacian simply by adding, the laplacian image to the original, the equation will be as the following in the Equation 4.4:

$$G(x, y) = f(x, y) + \Delta^2 f(x, y)$$

Equation- 4.4 the Equation of laplacian filtering

4.4.1 Flowchart Laplacian filter

The flowchart of enhancing image by laplacian filter is in the following figure 4.15

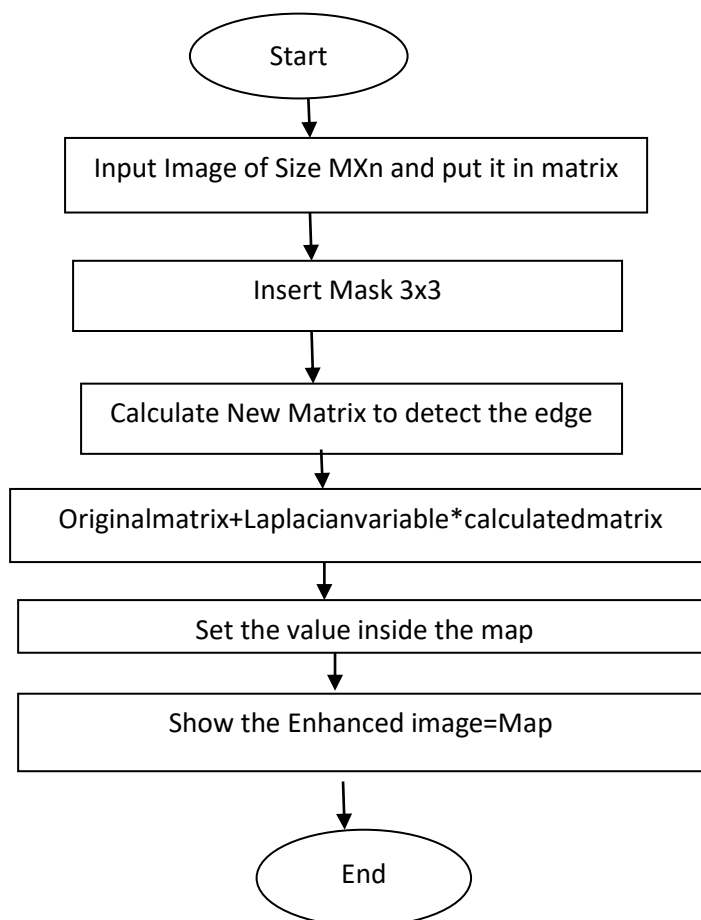


Figure 4.15 - Gaussian Filter Flowchart

4.4.2. My Own Results by Applying Gaussian Filter

Where $f(x,y)$ and $g(x,y)$ are the input and the sharpened images respectively, the constant is $c=-1$ if the mask of laplacian filter is

$$\begin{bmatrix} 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and $c=1$ if the mask of laplacian filter is

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

Then it is sharper than the first result as the result in the following figure 4.16

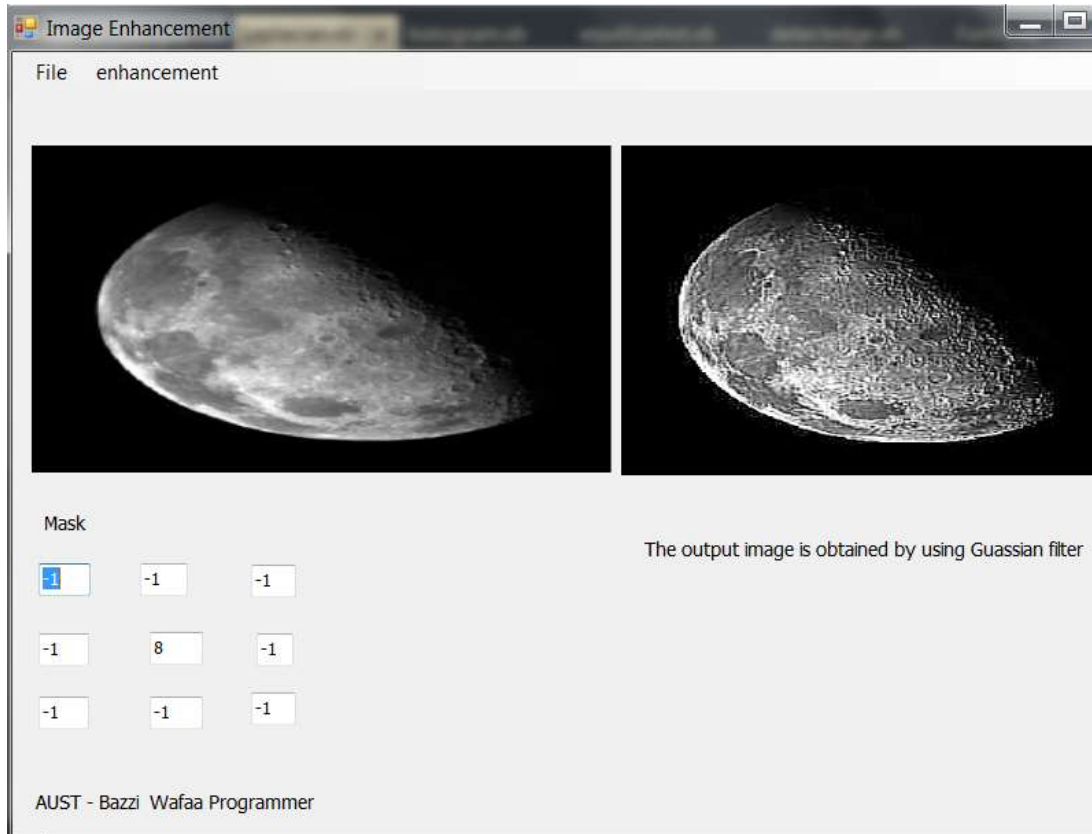


Figure 4.16 - Sharp Image Enhancement by Gaussian Filter

If I apply the first mask on the same picture , I obtain the follow result in the figure 4.17

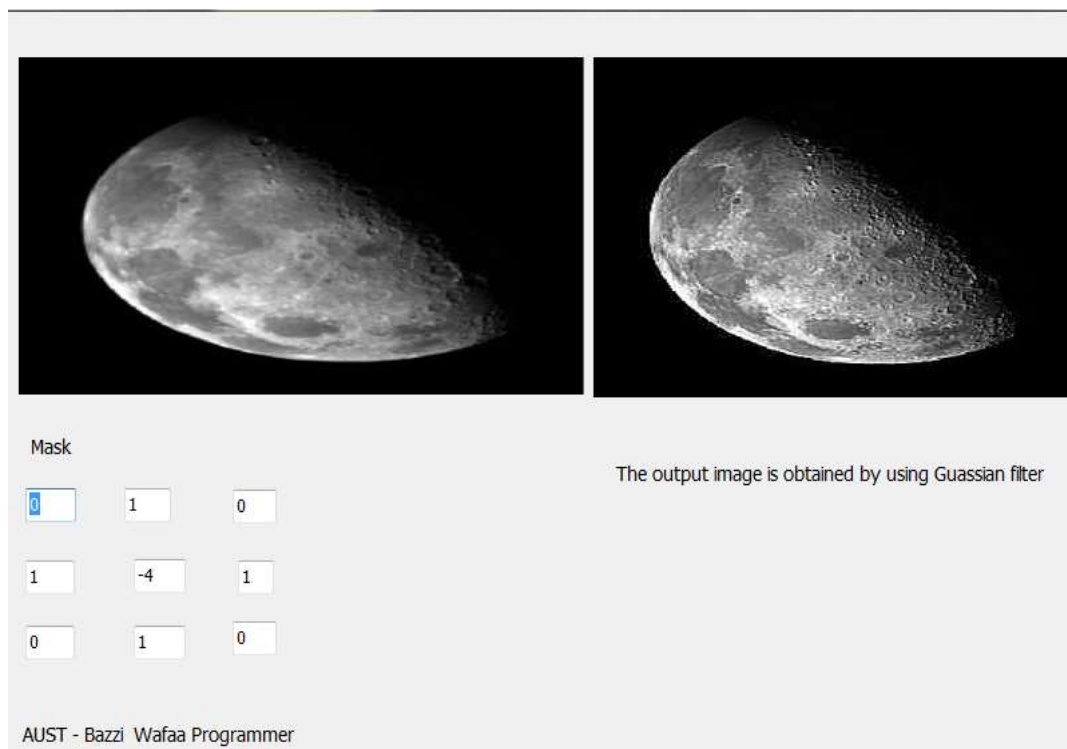
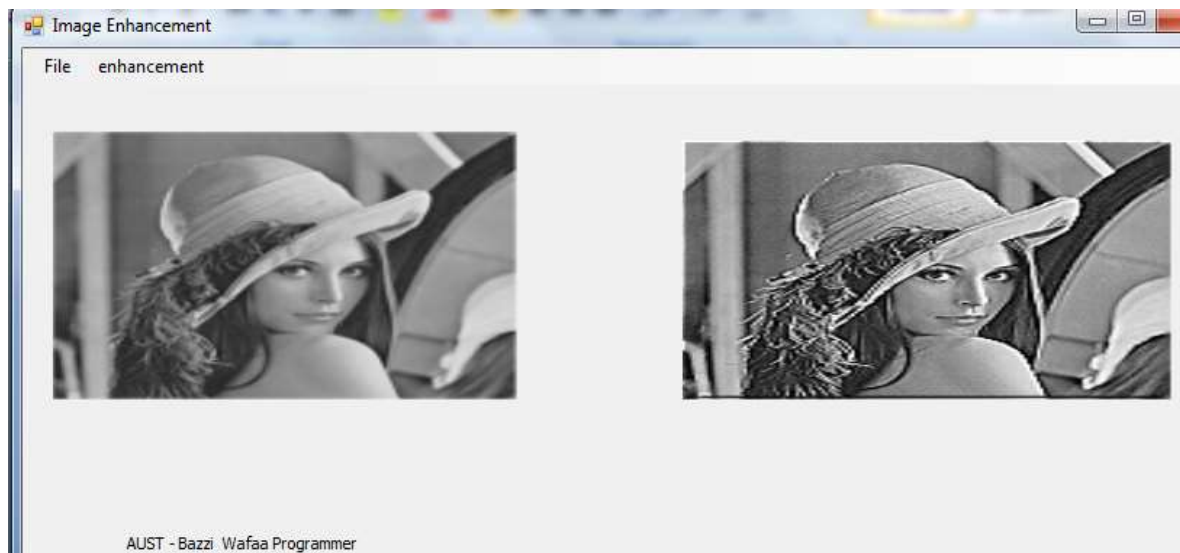


Figure 4.17 - Gaussian Filtering Result.

The image shows that the detail in it is unmistakably clearer and sharper than in the original image by using laplacian. Adding the original image to the laplacian restored the overall intensity variations in the image, as laplacian increases the contrast at the locations of intensity discontinuities. The net result is an image in which small details were enhanced and background tonality was reasonably preserved. Gaussian filter to enhance the dame has the follow result in the figure 4.18



Figer 4.18 - Gaussian Filtering Result

4.5. Working with Lightweight Concurrency

Unfortunately, neither Amdahl's Law nor Gustafson's Law [13, 14, 25, 26] takes into account the overhead introduced by parallelism. Consider the existence of patterns that allow the transformation of sequential

parts into new algorithms that can take advantage of parallelism. It is very important to reduce the sequential code that has to run in applications to improve the usage of the parallel execution units.

In previous .NET Framework versions, if you wanted to run code in parallel in a VB application you had to create and manage multiple threads (software threads). Therefore, you had to write complex *multithreaded* code. Splitting algorithms into multiple threads, coordinating the different units of code, sharing information between them, and collecting the results are indeed complex programming jobs. As the number of logical cores increases, it becomes even more complex, because you need more threads to achieve better scalability. The multithreading model wasn't designed to help developers tackle the multi-core revolution. In fact, creating a new thread requires a lot of processor instructions and can introduce a lot of overhead for each algorithm that has to be split into parallelized threads. Many of the most useful structures and classes were not designed to be accessed by different threads, and, therefore, a lot of code had to be added to make this possible. This additional code distracts the developer from the main goal: achieving a performance improvement through parallel execution.

Because this multithreading model is too complex to handle the multi-core revolution, it is known as *heavyweight concurrency*. It adds an important overhead. It requires adding too many lines of code to handle potential problems because of its lack of support of multithreaded access at the framework level, and it makes the code complex to understand.

The aforementioned problems associated with the multithreading model offered by previous .NET Framework versions and the increasing number of logical cores offered in modern processors motivated the creation of new models to allow creating parallelized sections of code. The new model is known as *lightweight concurrency*, because it reduces the overall overhead needed to create and execute code in different logical cores. **It doesn't mean that it eliminates the overhead introduced by parallelism**, but the model is prepared to work with modern multi-core microprocessors. The heavyweight concurrency model was born in the multiprocessor, when a computer could have many physical processors with one physical core in each. The lightweight concurrency model takes into account the new micro architectures in which many logical cores are supported by some physical cores. The lightweight concurrency model is not just about scheduling work in different logical cores. It also adds support of multithreaded access at the framework level, and it makes the code much simpler to understand. Most modern programming languages are moving to the lightweight concurrency model. Luckily, .NET Framework 4 is part of this transition.

4.6. Creating Successful Task-Based Designs

To optimize an existing solution [\[25\]](#) and to take advantage of parallelism, you have to understand an existing sequential design, and then you have to re-factor it to achieve a performance improvement without generating different results. You can take a small part and create a *task-based design*, and then you can introduce parallelism. The same technique can be applied when you have to design a new solution. You can create successful task-based designs by following these steps:

1. Split each problem into many sub-problems and forget about sequential execution.
2. Think about each sub-problem as any of the following
 1. Data that can be processed in parallel — Decompose data to achieve parallelism.
 2. Data flows that require many tasks and that could be processed with some kind of complex parallelism — Decompose data and tasks to achieve parallelism.
 3. Tasks that can run in parallel — decompose tasks to achieve parallelism.
 4. Organize your design to express parallelism.
 5. Determine the need for tasks to chain the different sub-problems. Try to avoid dependencies as much as possible (minimizes locks).
 6. Design with concurrency and potential parallelism in mind.
 7. Analyze the execution plan for the parallelized problem considering current multi-core microprocessors and future architectures. Prepare your design for higher scalability.
 8. Minimize critical sections as much as possible.
 9. Implement parallelism using task-based programming whenever possible.
 10. Tune and iterate.

The abovementioned steps don't mean that all the sub-problems are going to be parallelized tasks running in different threads. The design has to consider the possibility of parallelism and then, when it is time to code, you can decide the best option according to the performance and scalability goals. It is very important to think in parallel and split the work to be done into tasks. This way, you will be able to parallelize your code as needed.

4.7. Designing With Concurrency in Mind

When you design code [15, 25] to take advantage of multiple cores, it is very important to stop thinking that the code inside a VB2010 application is running alone. VB2010 is prepared for concurrent code, meaning that many pieces of code can run inside the same process simultaneously. The same class method can be executed in concurrent code. If this method saves a state in a static variable and then uses this saved state later, many concurrent executions could yield unexpected and unpredictable results.

As previously explained, parallel programming for multi-core microprocessors works with the shared-memory model. The data resides in the same shared memory, which could lead to unexpected results if the design doesn't consider concurrency. It is a good practice to prepare each class and method to be able to run concurrently, without *side effects*. If you have classes, methods, or components that weren't designed with concurrency in mind, you would have to test their designs before using them in parallelized code.

Each sub-problem detected in the design process should be capable of running while the other sub-problems are being executed concurrently. Once you begin working with parallelized code, it is very easy to incorporate other existing classes, methods, and components that create undesired side effects because they weren't designed for concurrent execution.

4.8. Project Description or Flowchart of My Application

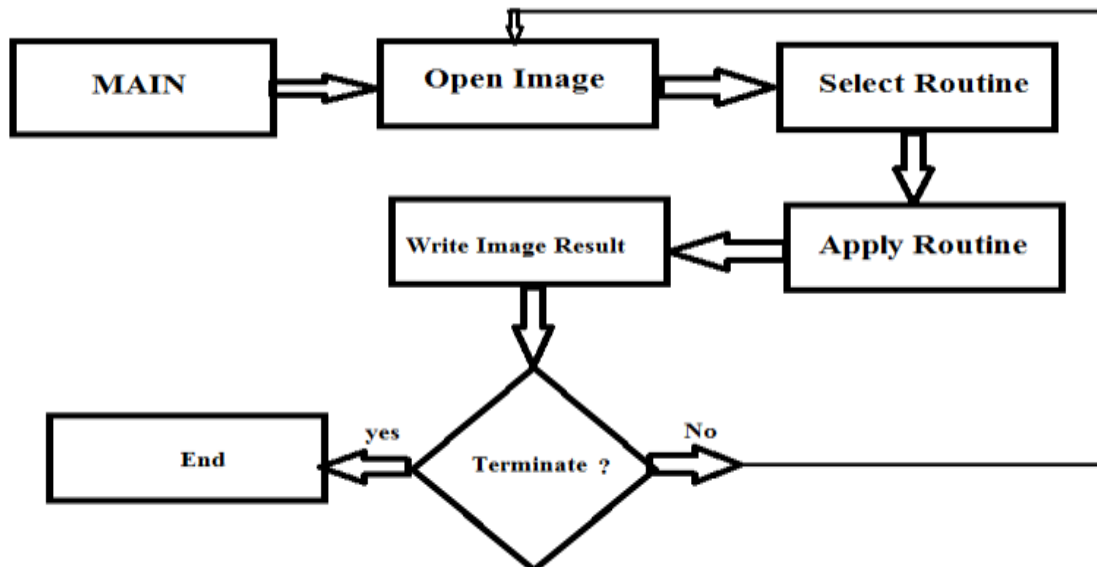


Figure 4.19 - Flowchart of My Application

Low image contrast limits the amount of information conveyed to the user. With the propagation of digital imagery and computer interface between man-and-machine, it is now feasible to consider enhancing the image digitally before presenting it to the user. This chapter explores the effects of the Intensity, histogram equalization, spatial filter and Guassian Filtering when it is applied on gray scale digital images so the result is an improved image quality.

4.9. My Own Experimental Results

The result of the Sequential implementation of detect edge technique is in the following figure 4.20

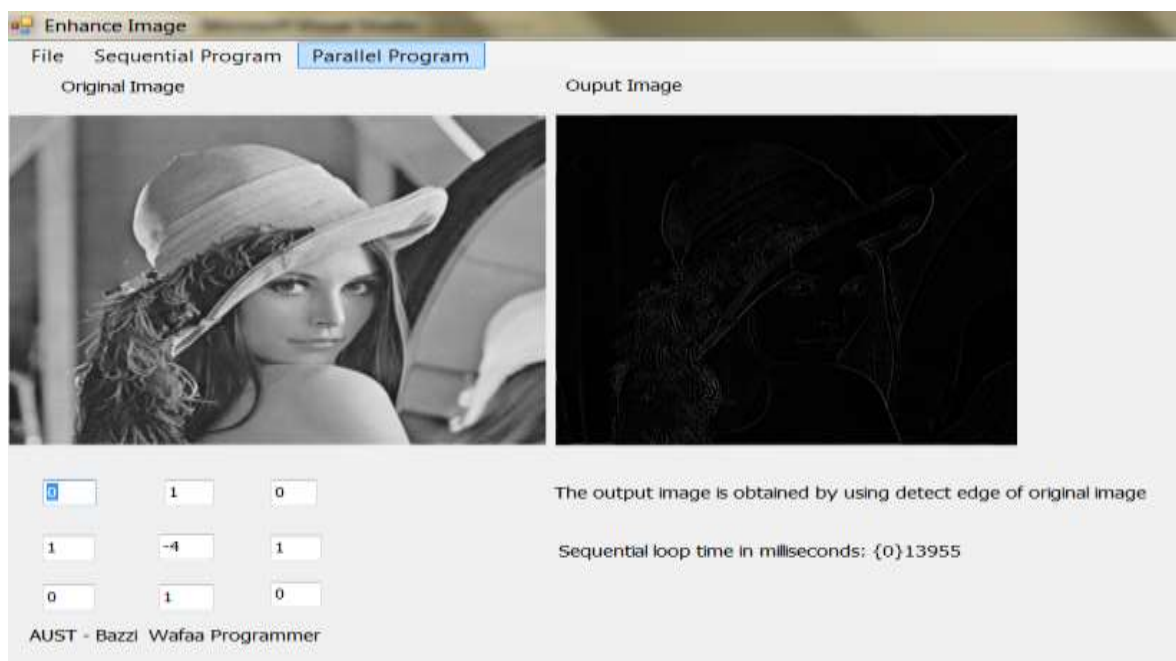


Figure 4.20 - Edge Detection of a Selected Image

Notice that the sequential time process requires 13955 ms. Hence if we run the program to detect Edge by parallel enhancement method, then the result is the following picture

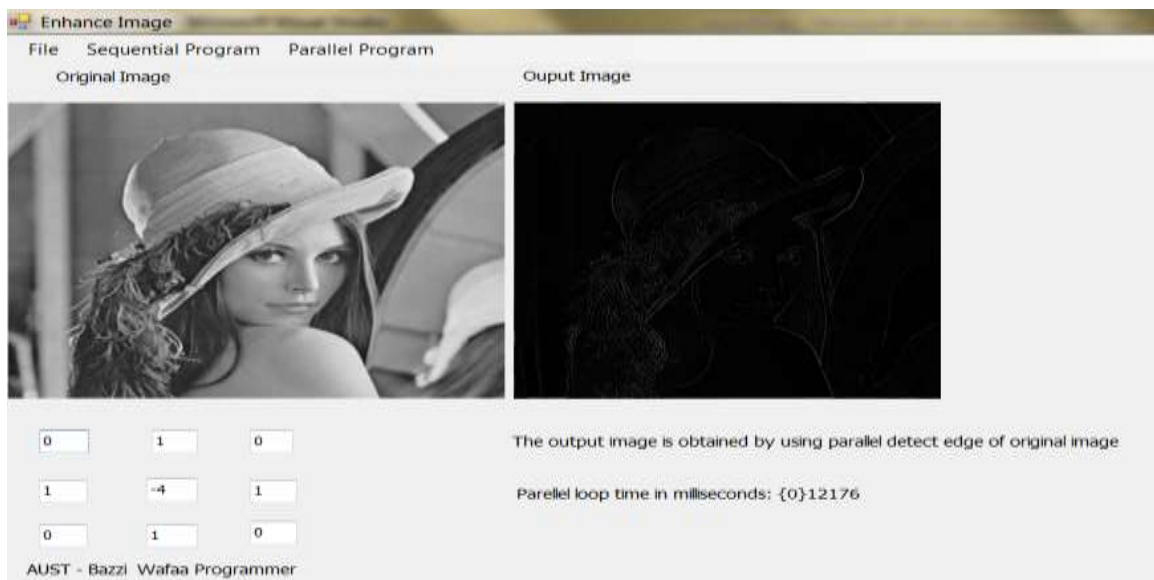


Figure 4.21 - Parallel Edge Detection

Notice that the parallel time process requires 12176 which means that the parallel program reduces the time approximately by $13955 - 12176 = 1779$ Milliseconds. Then if the user chooses sharp image and sequential method then the result is the following picture 4.22

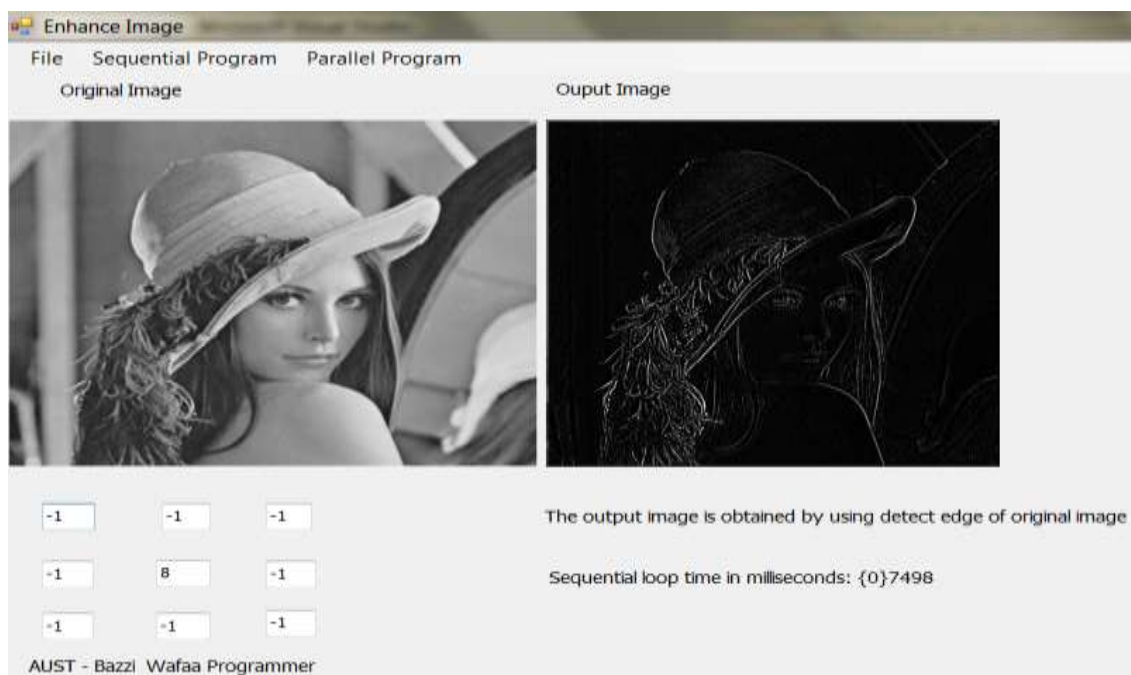


Figure 4.22 - Sequential Edge Detection

Notice that the sequential time process requires 7498 ms. So if we run the program to detect Edge by parallel enhancement method and to sharp image then the result is as follows in picture 4.23

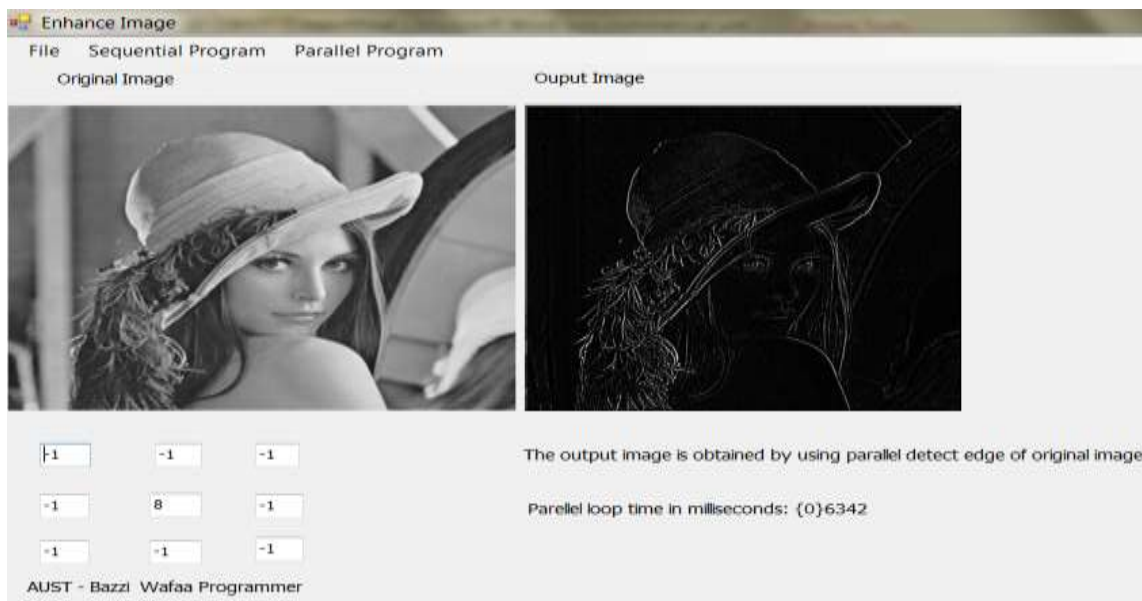


Figure 4.23 -Sharp Edge Detection

Notice that the parallel time process requires 6342 ms which means that the parallel program reduces the time approximately by $7498 - 6342 = 1156$ Milliseconds.

A remark is that the reading of the mask takes time and that's why sometime it is preferable to let the application to run it. So if we run the program to calculate the histogram and enhance image by equalization then the result obtained is in the following picture 4.24

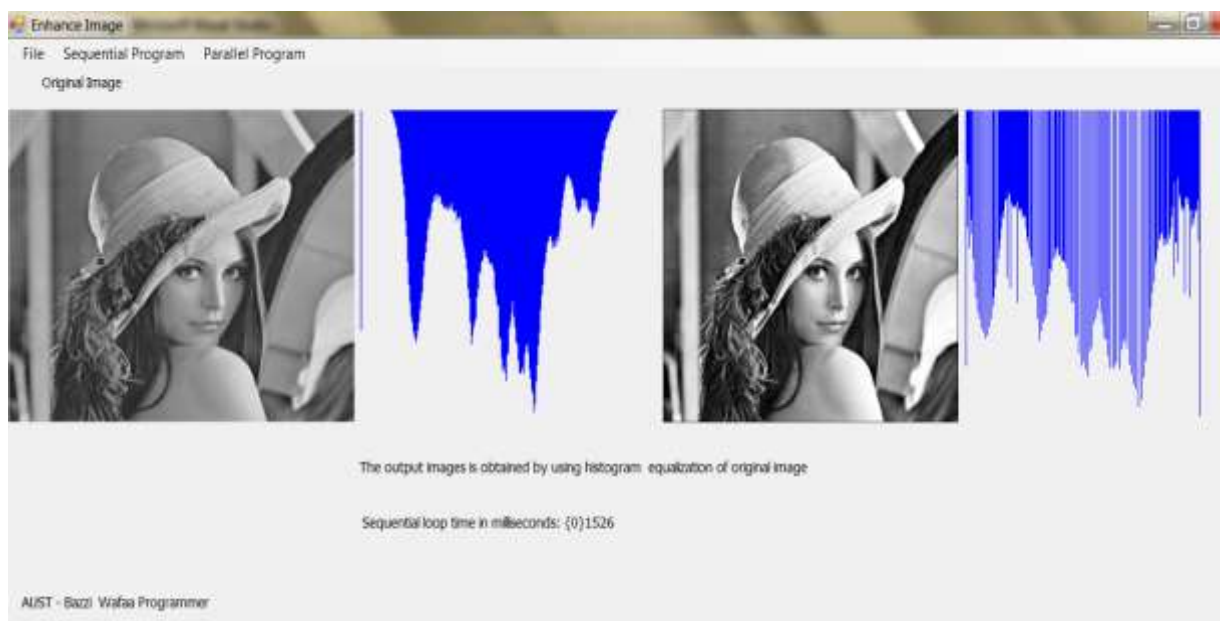


Figure 4.24 – Sequential Histogram and Histogram Equalization

Notice that the sequential time process requires 1526 ms.. So if we run the program to calculate the histogram and enhance the image by equalization in parallel method then the results are obtained in the following picture 4.25



Figure 4.25 - Parallel Histogram and Histogram Equalization

Notice that the parallel time process requires 1265 ms which means that the parallel program reduces the time approximately by $1526-1265=261$ Milliseconds. Then try to run the program to enhance the image by scale-Laplacian or Gaussian filter in sequential method to obtain the following picture 4.26

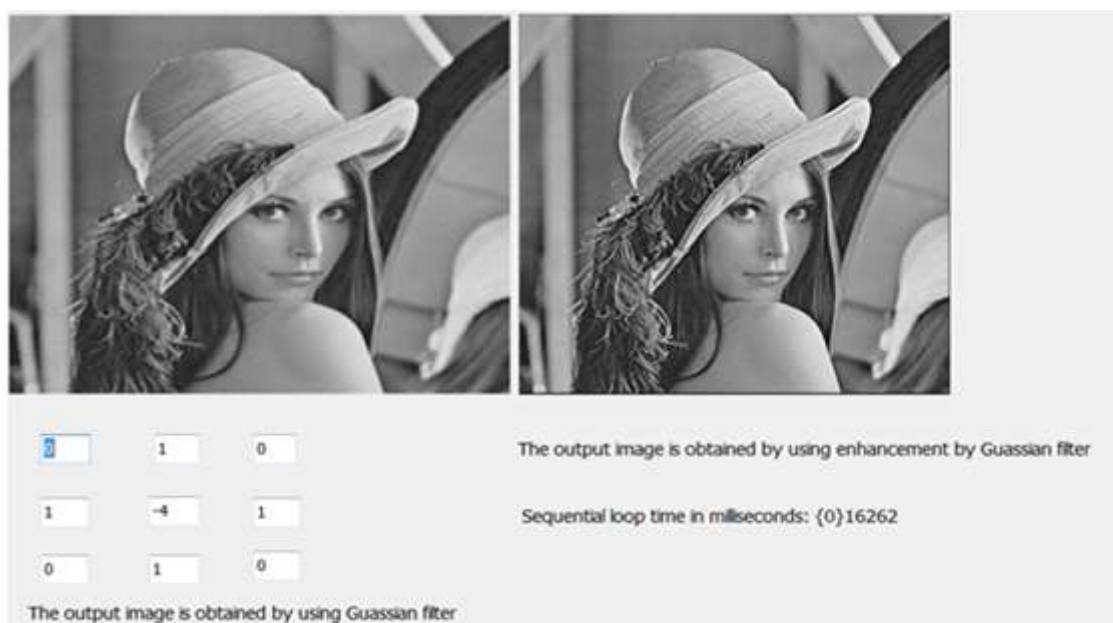


Figure 4.26 - Sequential Gaussian Filtering

Notice that the sequential time process requires 16262 ms. . So if we run the program to enhance the image by Gaussian Filter in parallel method then the results are obtained the following picture 4.27

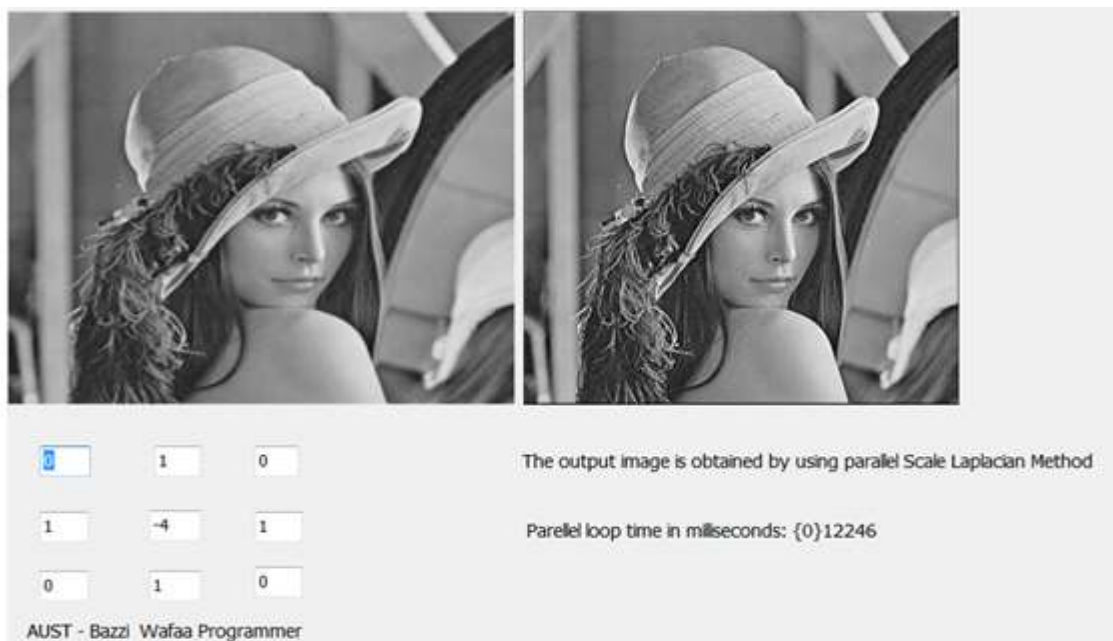


Figure 4.27 – Parallel Gaussian Filtering

Notice that the parallel time process requires 12246 ms which means that the parallel program reduces the time approximately by $16262 - 12246 = 4016$ Milliseconds. Run the application to enhance by Guassian filter by choosing that the application put the mask and unsharp the image, the result is as follows in the figure 4.28

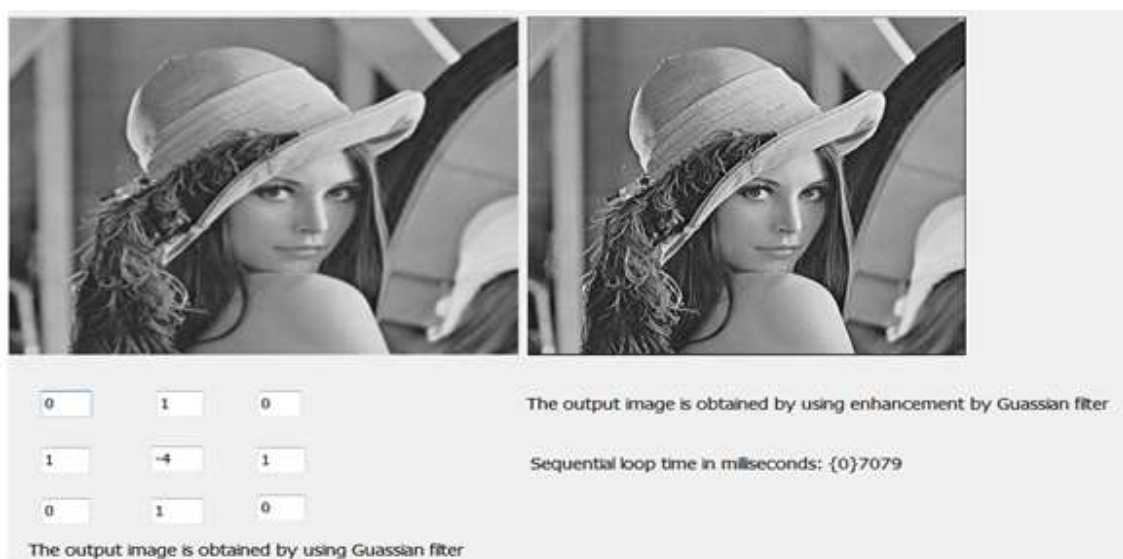


Figure 4.28 – Sequential Unsharp Gaussian Filtering

Notice that the sequential time process requires 7079 ms. If we run the program to enhance the image by Gaussian Filter in parallel method and let the applicatrion put the mask and unsharp the image, then the result is the following picture 4.29

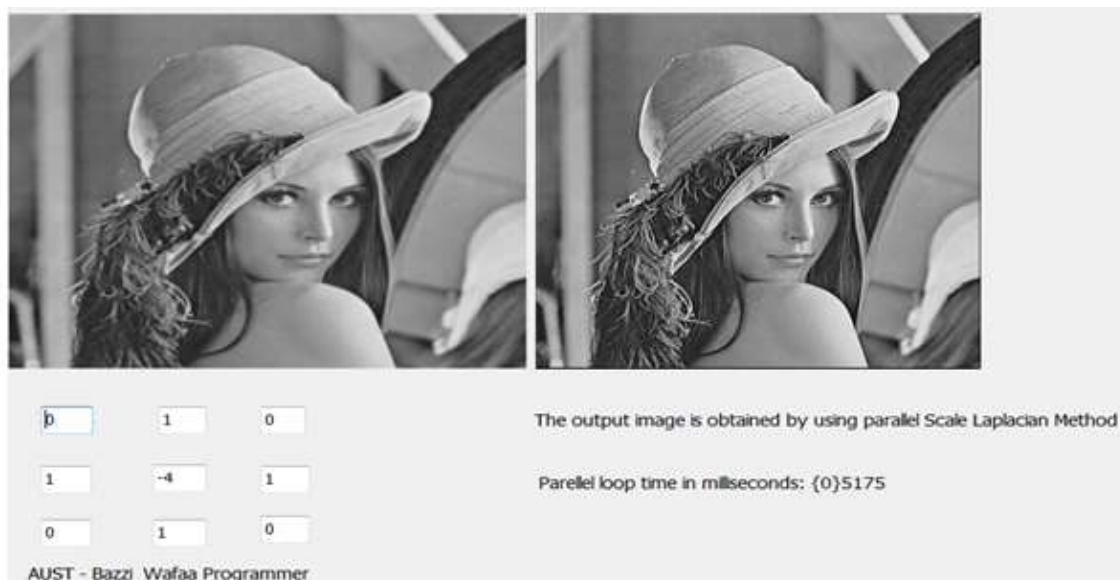


Figure 4.29 – Parallel Unsharp Gaussian Filtering

Notice that the parallel time process requires 5175 ms which means that the parallel program reduces the time approximately by $7079 - 5175 = 1904$ Milliseconds. If we run the program to enhance the image by Gaussian Filter in parallel method and let the application put the mask and sharp the image, then the results are obtained in the following picture 4.30

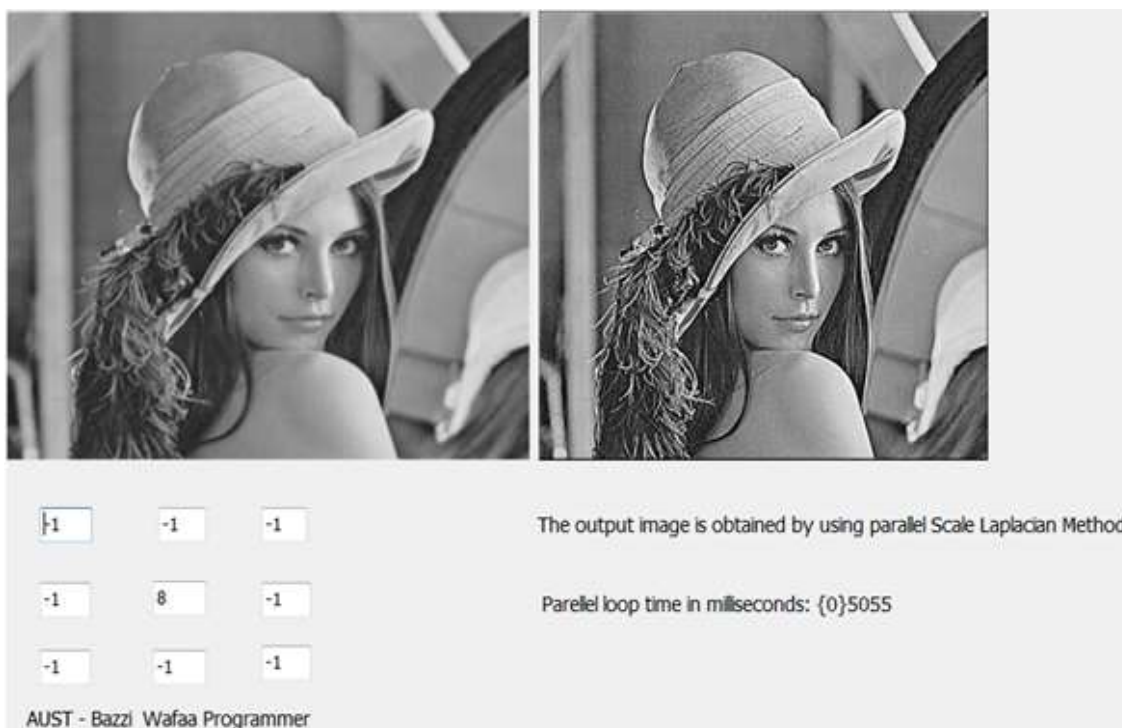


Figure 4.30 - Parallel Sharp Gaussian Filtering

Notice that the parallel time process requires 5055 ms and the image is sharper than before. Then run the spatial method filter method to enhance the same image in sequential way to obtain the following result in figure 4.31

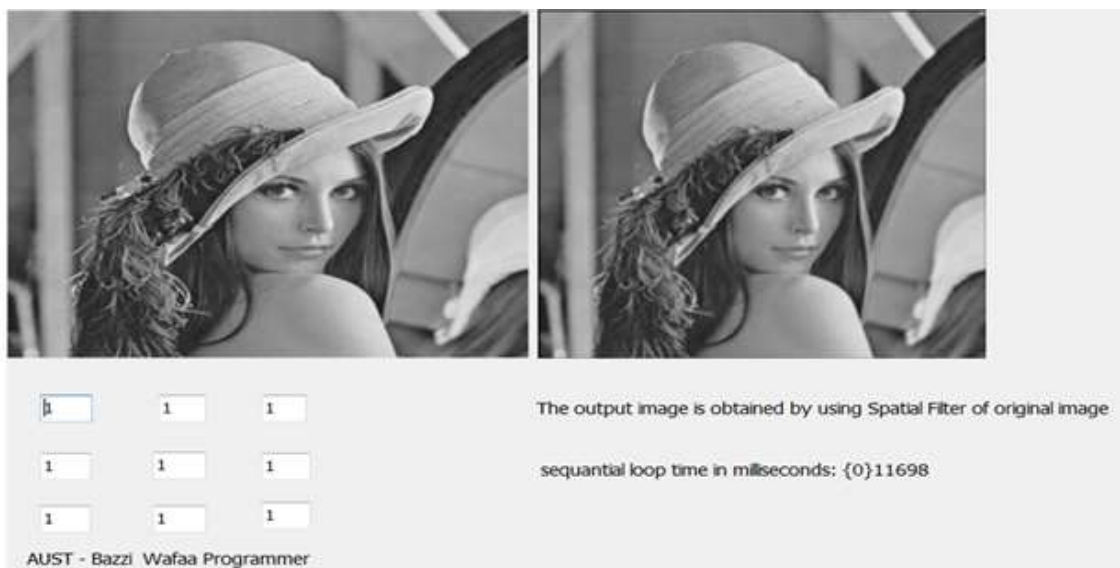


Figure 4.31 - Sequential Spatial Filtering

Notice that the sequential time process requires 11698 ms. If we run the program to enhance the image by Spatial Filter in parallel method, the results are in the following picture 4.32



Figure 4.32 - Parallel Spatial Filtering

Notice that the parallel time process requires 8195 ms which means that the parallel program reduces the time approximately by $11698 - 8195 = 7673$ Milliseconds.

4.10. Pseudo Code of Parallel Histogram

(*parahistogram*)

(mxn image, 256 gray *)

Parallel.For(0, 255, Sub(i)

 hist(i) = 0

End Sub)

Parallel.For(0, nrx, Sub(i)

 For j As Integer = 0 To nry

$$r = m11(i, j)$$

$$\text{hist}(r) = \text{hist}(r) + 1$$

Next

End Sub)

'plot the histogram

4.11. The appropriate choice of Image Enhancement Techniques

Digital image enhancement techniques provide a multitude of choices [4, 27, 28] for improving the visual quality of images. Appropriate choice of such techniques is greatly influenced by the imaging modality, task at hand and viewing conditions.

4.11.1. Thresholding transformations are particularly useful for segmentation in which we want to isolate the object of interest from a background as shown in the figure below Figure 4.33

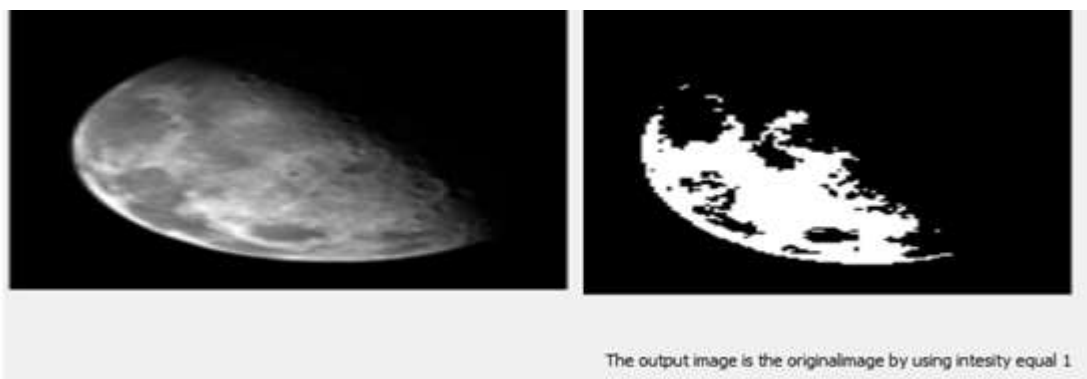


Figure 4.33 –Intensity Level Enhancement of the Moon

4.11.2. Histogram equalization is a common technique for enhancing the appearance of images. Suppose we have an image which is largely dark. Then its histogram would be skewed towards the lower end of the grey scale and all the image details are compressed into the dark end of the histogram. If we could 'stretch out' the grey levels at the dark end to produce a more uniformly distributed histogram then the image would become much clearer as shown in the figure 4.34

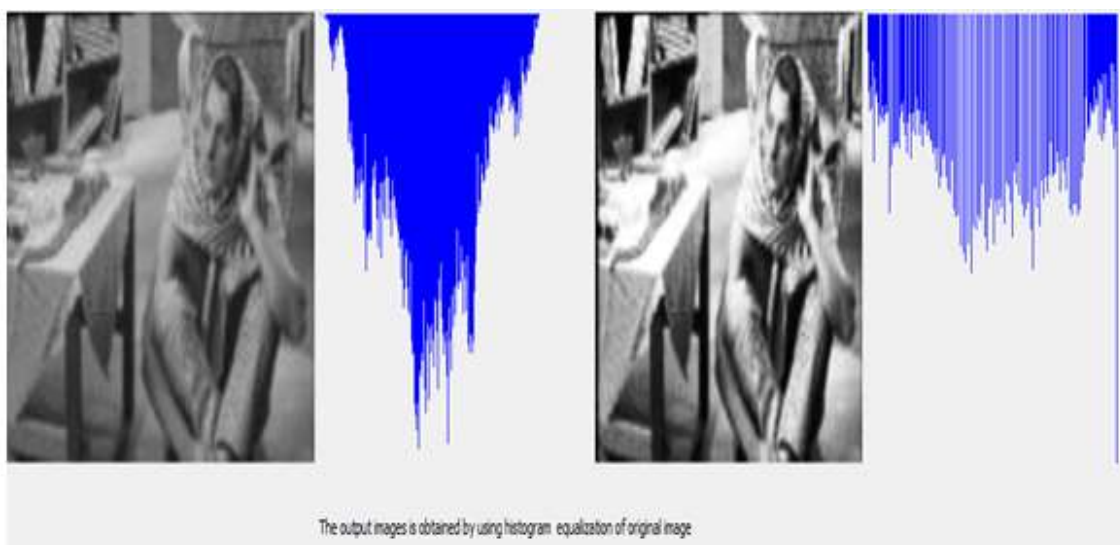


Figure 4.34 – Histogram Equalization of the Girl Image



Figure 4.35 –Histogram Equalization of an Image

The original image and its histogram, and the equalized versions. Both images are quantized to 64 grey levels. Histogram equalization is a transformation that stretches the contrast by redistributing the gray-level values uniformly.

4.11.3. Previous methods of histogram equalizations are global. So, local enhancement is used. Define square or rectangular neighborhood (mask) and then move the center from pixel to pixel. For each neighborhood, calculate histogram of the points in the neighborhood. Map gray level of pixels centered in neighborhood as shown in the figure 4.36

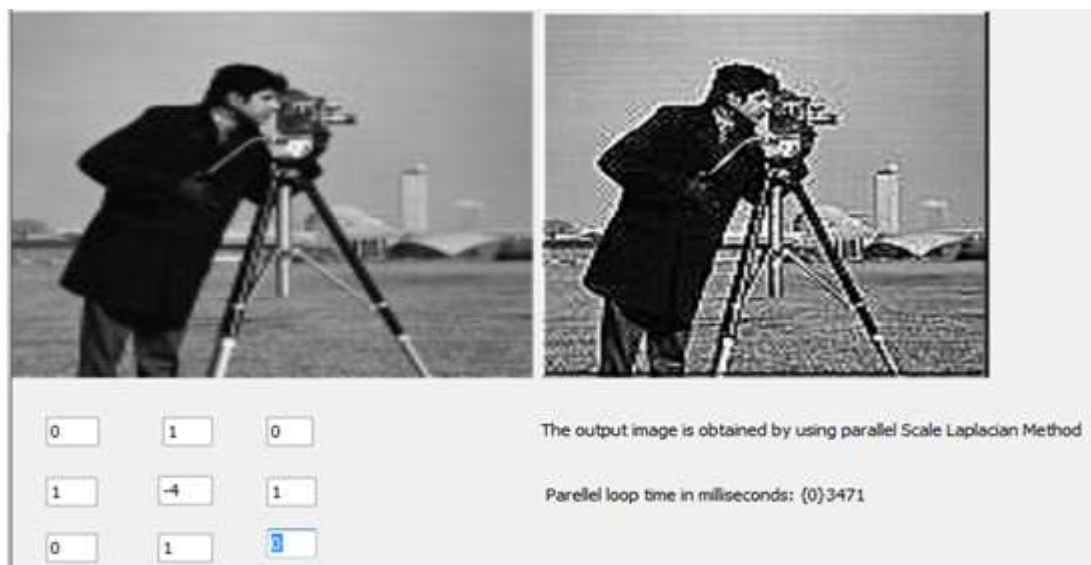


Figure 4.36 –Spatial Filter result of an Image

4.11.4. Most pixels in the image look very similar to their neighbours, but Image noise results in pixels that look very different from their neighbors. The else, the pixels across the edge are not. What is to be done? smoothing the image should help, by forcing the noise pixels to look more like their neighbours, and underline the edge . Edge Detect method produces images that have grayish edge lines and featureless backgrounds. Adding the laplacian image to the original will result in preserving the Edge as shown in both two figures 4.37 and 4.38 .

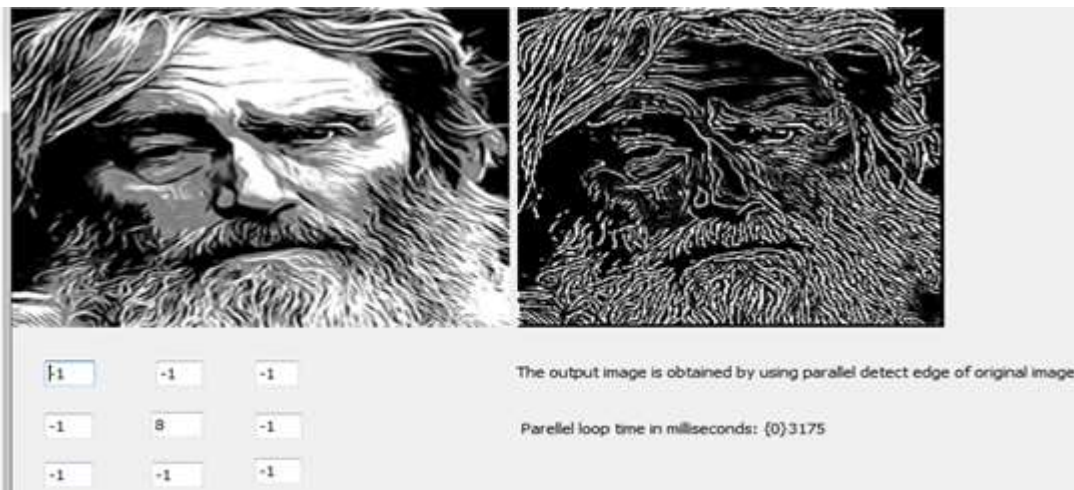


Figure 4.37 – Edge Detecting of a Selected Image.

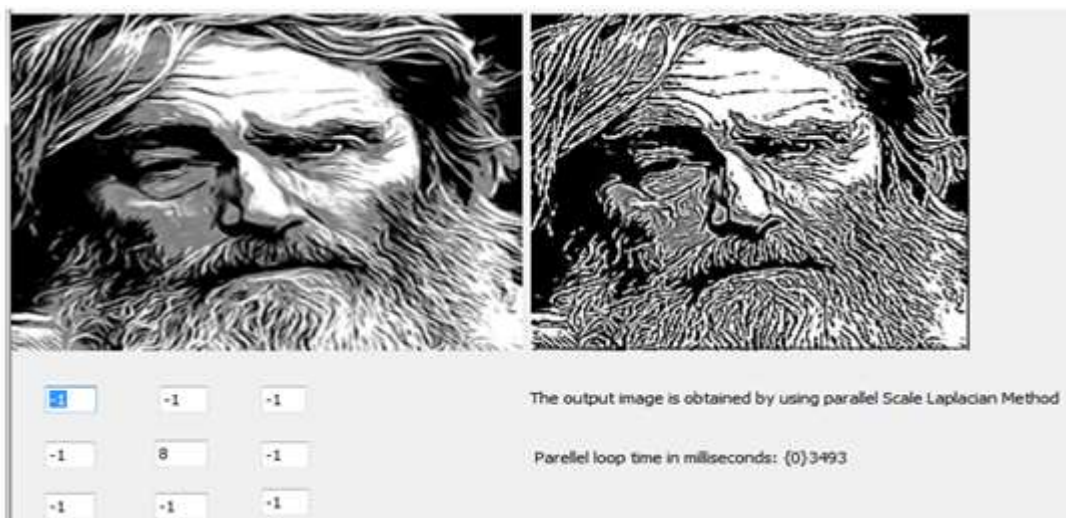


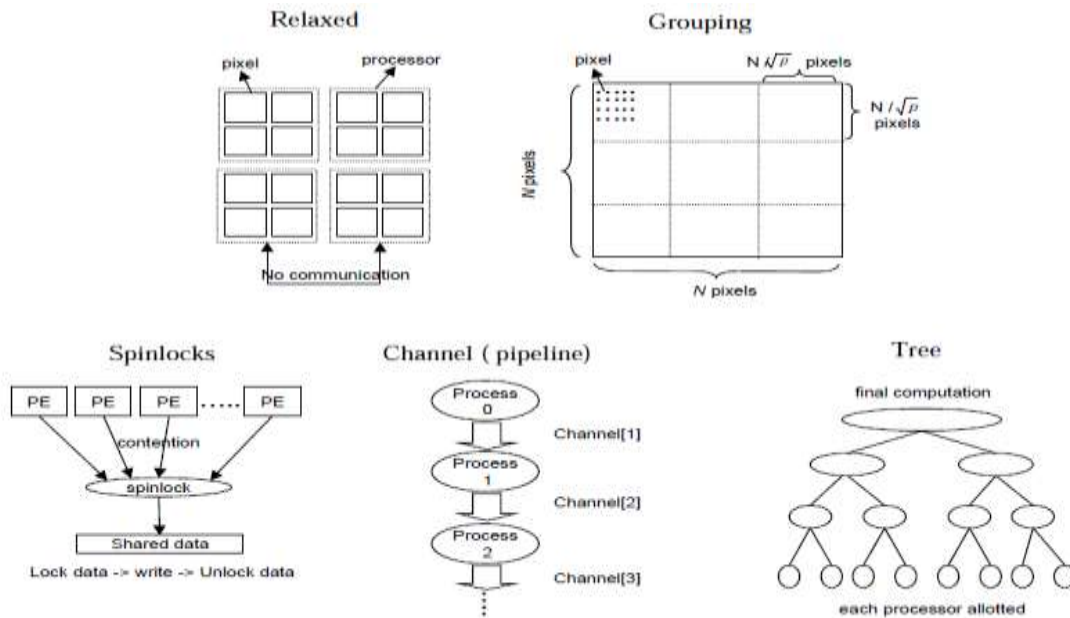
Figure 4.38 - Guassian Filter of an Image .

Nowadays, the performance of the image enhancement algorithms in the Image processing have grand challenge. Gray scale images were used in the tests. The results indicated that the new algorithms are 8 times more accurate than the enhancement image algorithms of sequential method which is embedded in my application and 80 times more accurate than the preceding ones. Some of the algorithms are able to enhance faster than the others because some of the other methods contain more dependent processes, and the parallelism is better when the processes are independent.

PARALLEL PERFORMANCE

5.1. Parallel Algorithm and Task Scheduling

Some aspects of parallel algorithms [14, 15, 26, 29, 30] are illustrated through some representative tasks (Histogram , Histogram Equalization, Spatial Filter , Edge Detection , Gaussian Filter). A brief description of the scheduling is also provided below



Scheduling Scheme 1.

5.2. Scheduling Schemes

These schemes are depicted [14] in Figure 5.1 and Figure 5.2. Two scheduling schemes, local and global. In local scheduling, each operator can be optimized through appropriate schemes for best performance. Also, a global job consisting of several tasks is arranged by task graph. At run time, the workload is distributed dynamically among the processors. Dynamic load balancing can be classified as centralized, fully distributed, or semi distributed. In centralized load balancing global information can allow the algorithm to do a good job balancing the work among the processors, but this approach does not scale well. Fully distributed load balancing allow processors to exchange information with neighboring processors. It has the advantage of lower scheduling overload, but the workload may not be balanced as well as it would be by centralized algorithms. Semi-distributed approach is hybrid of the above two schemes. Efficient load balancing and intelligent task management must be included for better performance. Each operator requires a specific scheme in accordance with its computational characteristics. It is hard to build parallel algorithms for data-dependant operations.

1. Data-independent

- Local operators (Pixel) : Relaxed , Grouping.
- Local operators (Mask): Partially Relaxed , Grouping.
- Global operators: Spinlock, Channel, Grouping, Tree.

2. Data-dependent

- Partitioning , Grouping, tree.
- AI-approach (quite difficult).

Process communication , data sharing, synchronization, data partitioning, and message passing should be combined and coordinated to carry out each operation .

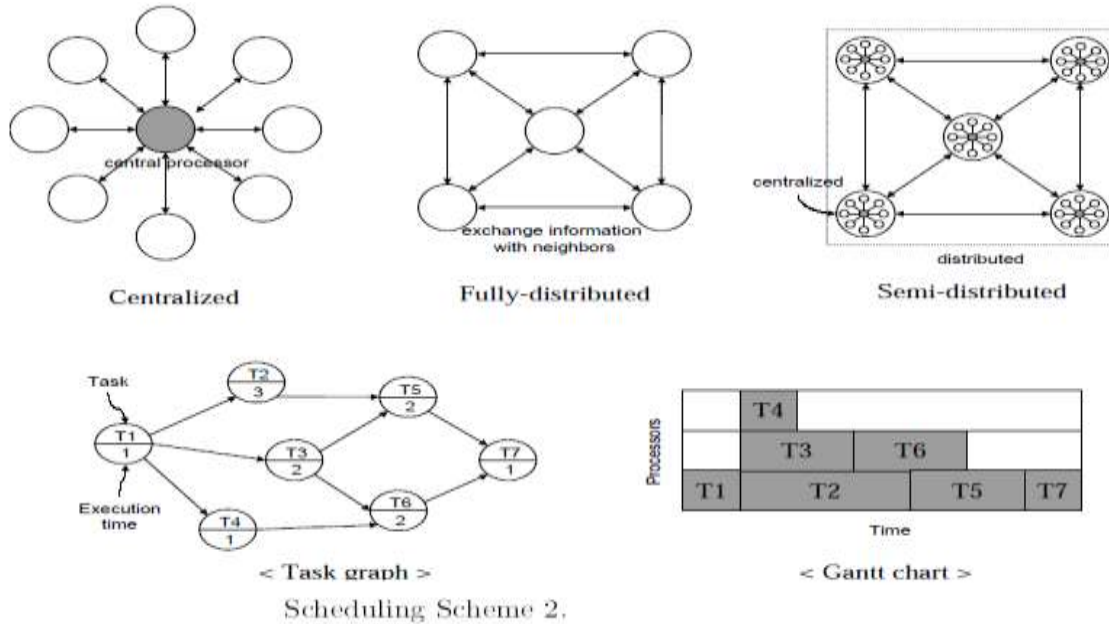


Figure 5.2 - Global Schedule Scheme

The table below contains classification of overall tasks.

Table 5.1- Classification of Overall Tasks

	Local operators		Global operators
Data-independent	Pixel	mask	
	Arithmetic	2D	Histogram
	Multiple operand		Histogram Equalization
	Single operand		
	Complex arithmetic		
	Non linear functions		
	Comparison, logic		
	Data conversion		
	Format conversion		
	Classification		
Data dependent	Edge Detecting		

	Guassain filter
	Spatial filter

Figure 5.1 - Local Schedule Scheme

5.3. The Performance (Calculation Rule)

It's useful to estimate the CPU or memory resources [14] of an algorithm. This complexity analysis tries to put estimation of resource usage (time or space) with a simple formula approximation. Many programmers have had ugly surprises when they moved from small test data to large data sets. This analysis will make you aware of potential problem.

5.4. Implementation Complexity

Analyzing algorithms like the RAM model imply that each simple operation (+,-,=,if ,call) takes exactly one step. Furthermore Loops and subroutine calls are not simple operations, but depend upon the size of the data and the contents of the subroutine. So each memory access takes exactly one step. And the *worst case complexity* of the algorithm is the function defined by the maximum number of steps taken on any instance of size n. The *best case complexity* of the algorithm is the function defined by the minimum number of steps taken on any instance of size n. The *average case complexity* of the algorithm is the function defined by the average number of steps taken on any instance of size n.

5.5. The Sequential Method Efficiency

The space used to store the image is mxn , and to store the mask (used to enhance image) is $3 \times 3 = 9$. The usual filter of mask to the image has time efficiency $10mn + \epsilon$. which belong to (mn) .To detect the edge , the time efficiency is about $9mn + \epsilon$ belong to (mn) .Time efficiency Scale for Laplacian to enhance image is about $10mn + \epsilon$ belong to (mn) . In the following table 5.2 are some of the measures for time performance.

Table 5.2 - Measure of time efficiency.

			Best Case	Worst case	Average Case
Process/Operatio n	Rea d arra y	Read mask	Operations Numbers	Operations Numbers	Operations Numbers
Pre-filtering (mxn)	mxn	3×3	$mxn + 9$	$mxn + 9$	2
Filtering	mxn	3×3	$9mn + mn$	$9mn + mn$	$p/mn = 9$ appro x
Edge detection	9		$9mn$	$9mn$	$p/mn = 9$ appro x

Laplacian with scale			$9mn+mn$	$9mn+mn$	$p/mn=9$ appro x
----------------------	--	--	----------	----------	---------------------

5.6. Difference between the Sequential and the Parallel Efficiency

A histogram of an image is used to display the distribution of gray-values in the image. An 8-bit image has 256 gray-level ranging from absolute black to absolute white. Computing the histogram is a very common technique used in image processing as a part of some larger process b, such as identifying in the image or equalizing the image.

5.6.1. Histogram sequential algorithm1

(*SeqH istogram*)

(*mxn image, 256 gray levels*)

Begin

For i=0 to image-width-1

For j=0 to image-height-1

Get pixels

R=c.r

Hist(r)=hist(r)+1

Next

Next

Plot the histogram

This histogram algorithm can be parallelized by changing the outer FOR loop into parallel.for statement(Parallel.for means that whole statement in the Parallel.for performed in parallel grouped into specified number of partitions). This will create one parallel process for each row of the image. This is shown in algorithm2

5.5.2. Histogram Parallel Algorithm2

(*parahistogram*)

(mxn image, 256 gray ,L is spinlock*)

Parallel.For(0, 255, Sub(i)

hist(i) = 0

End Sub)

Parallel.For(0, nrx, Sub(i)

For j As Integer = 0 To nry

r = m11(i, j)

hist(r) = hist(r) + 1

Next

End Sub)

'plot the histogram

The following notations are used in the analysis of these algorithms, N is the size of problem, P is the Number of processes, $T(n,p)$ is the Elapsed time (parallel time complexity), $T(n,1)$ is the Time complexity of the best known serial algorithm for solving the same problem, $S(P) = \frac{T(n,1)}{T(n,p)}$:Speedup, and $E(n,p) = \frac{S(p)}{p}$ is the Efficiency.

In this histogram algorithm case, $T(n,1)$, the serial time complexity is $O(n^2)$ and $T(n,n)$, the parallel time complexity is $O(n)$. (Algorithm 2). Hence, the speedup is $O(n^2)/O(n) = O(n)$ and efficiency is $O(n)/n = O(1)$, which may not be bad. This is the same case as when I use The loop Parallel. For to enhance the time efficiency of other algorithms. Some of the m has dependent tasks and I try to divide them as much as possible in several subroutines in order to use the parallelism and private variables to lock the variables from changing by other tasks.

Conclusions & Future Work

There are many different ways to increase the contrast and other features of an image. These concepts are important for the recognition of the objects and details in an image. Histogram equalization is a simple way to increase the contrast of an image. By stretching the pixels 'around' the mean intensity, an image emphasizing the light and dark areas of an image can be realized.

Images always contain some amount of noise. It is often useful to model or simulate different types of noise in order to design filters, to remove it or to study its effects on other image transformation (eg. edge detection algorithms). There are filters that can be applied to an image in order to remove the noise. These filters can be average filters or median filters. Average filters are often used to remove Gaussian noise as they help 'smooth' the noise out over the image.

Edge detection is commonly used for object tracking, recognition, etc. Noise can affect the results of these edge detection algorithms. In environments where edge detection is often used, motion blur is often a problem. However, these results appeared not to be very useful in real-world applications. In motion blurry environments, Gaussian Filter seemed to be the most effective of the four filters.

6.1 Problems Encountered and Solved

We classify the parallel spatial methods for enhancing the image into 4 different categories:

- A) Histogram and histogram equalization
- B) Spatial Filtering .
- C) Guassian Filter.
- D) Detect Edge.

In addition, we decompose the algorithms into independent procedures that use parallel method and increase the time efficiency.

Furthermore, we choose the image enhancing algorithm to parallelize because of its data input and processes which require time complexity. There are several sequential algorithms to enhance the images; however there is no parallel version for these algorithms. Therefore, we decide to investigate improving the performance of the parallel version of the algorithm.

6.2. Conclusion

Spatial Operations and Transformation are fundamental modules in the image processing. We modify the sequential algorithms to remove the noise of gray-scale images. We also provide a parallel versions of the algorithms in order to speed up the processes of enhancement. Due to data dependencies in the algorithm, we use a method similar to the pipeline method in order to exploit parallelism. The parallel algorithm achieved the best speedup while processing medium sized images.

Our application allows you to quickly and easily reduce or remove the depth of details within images, while conserving important structural details and enhancing edges. Such that you can achieve the variety of smoothing, detail construction and edge enhancement. With just few clicks, you can see the appearance and intensity of the details or the elimination of all the noise in your photographs .

6.3. Suggestions for future extensions of the project

As a future work, we can investigate [\[23\]](#) for a method to provide a parallel enhancement algorithm which divides the image between several threads and each thread utilizes the algorithm enhancement ideas proposed in this paper for its local data. Then we integrate the local results to provide a consistent output for the whole image.

GPUs are intended for working with data streams. Future work could involve investigating the possibility of streaming multiple images into the GPU for image processing. Creating data streams would enable copying of data to the device as the prior image is being processed. Working with multiple images over the course of a single execution would likely make working with CUDA even more efficient than it already is.

GPU programming has the potential to be more effective at performing fast computations as traditional parallel computing. With this technology it is feasible that medical care could be improved in areas without access to expensive hospitals or medical experts.

**Approved by the Computer Science Department and the
Supervisory Committee
Program Authorized to Offer the Degree of Master of Science in
Computer Science**

We certify that we have read and tested this thesis and that, in our opinion, it is satisfactory in scope and quality as a Master Thesis for the degree of Master of Science in Computer Science.

Project Committee

Aziz M. Barbar, Ph.D, Adel Chit, Ph.D. , Elie Nasr, M.S.

Acknowledgments

This project is the most significant accomplishment in my life and it would be impossible without people who supported me and believed in me.

I would like to extend my gratitude and my sincere thanks to my honorable, esteemed supervisor Prof. A. Barbar, Department of Computer Science. He is not only a great lecturer with deep vision but also and most importantly a kind person. I sincerely thank for his exemplary guidance and encouragement. His trust and support inspired me in the most important moments of making right decisions and I am glad to work with him.

I want to thank all my teachers Prof. H. Mokbel, Prof. A.Tarhini, Prof. and C. Sarraf and for providing a solid background for my studies and research thereafter. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank all my friends and especially my classmates for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious. I've enjoyed their companionship so much during my studies.

Last but not least I would like to thank all my family , who taught me the value of hard work by their own example. They rendered me enormous support during the three years studies at AUST.

References

- [1] - Axel S. Trajano and Joseph Anthony C. Hermocilla, "Implementation of Selected Image Processing Routines for Single and Distributed Processors", Bachelor's thesis, Faculty of the Institute of Computer Science, University of the Philippines Los Banos, 2010.
- [2] - Luong Chi Mai, "Introduction to Image Processing and Computer Vision" , Department of Pattern Recognition and Knowledge Engineering, Institute of Information Technology, Hanoi, Vietnam, **2005**.
- [3] - Bhabatosh Chanda and Dwijest Dutta Majumder, "Digital Image Processing and Analysis", 2002.
- [4] - Raman Maini and Himanshu Aggarwal, "A Comprehensive Review of Image Enhancement Techniques", Journal of Computing , Volume 2, Issue 3, 2010 .
- [5] - E. Rafael and C. Gonzalez Richard, "Digital Image Processing" , Addison-Wesley; 3rd edition., 2003 .
- [6] - Scott T. Acton, "Image Enhancement", Oklahoma State University, 1999.
- [7] - C. Gonzalez and R. E. Woods, "Digital Image Processing" , Prentice-Hall, 2nd Edition, 2002.
- [8] - Chek Koon Teo, "Thesis-Digital Enhancement Of Night Vision And Thermal Images", MASTER's thesis, Naval Postgraduate School, 2002.
- [9] - John Hulskamp , "Computer Vision & Image processing", Thermal Imaging Fusion Enhanced Night Vision .
- [10] - **Linda G. Shapiro, George C. Stockman**, "Computer Vision", Prentice Hall, 2001.
- [11] - Gonzalez, Woods, "Digital Image Processing using MATLAB", Prentice Hall, 2004.
- [12] - EE465, "Introduction to Digital Image Processing", Copyright Xin Li, 2009.
- [13] - Dr. Monica Trifas, "Medical Image Enhancement" , International Conference on Computer Vision , Las Vegas, Nevada, USA, 2005.
- [14] - B. Wilkinson & M. Allen, "Parallel Programming Techniques & Applications Using, Networked Workstations & Parallel Computers" , 2nd Edition, Pearson Education Inc, 2004.
- [15] - Ananth Grama, "Introduction to Parallel Computing" , 2nd Edition., Addison Wesley, 2003.
- [16] - Calvin Lin and Lawrence Snyder, "Principles of Parallel Programming", Addison-Wesley, 2009.
- [17] - Mathew Hong, Quinn Lewis, and Udit Patidar , "Enhancing the Enhancement: Gray-Scale Image Enhancement as an Automatic Process Driven by Evolution" .
- [18] - Mehdi Niknam, Parimala Thulasiraman, Sergio Camorlinga, "A Parallel Algorithm for Connected Component Labelling of Gray-scale Images on Homogeneous Multicore Architectures", The open access *Journal of Physics: Conference Series (JPCS)* , 2010.
- [19] - Bertan Karahoda and Gulden Kaktur, "Gall Bladder Ultrasonic Image Analysis by Using Discrete Wavelet Transform" , Dokuz Eylul University Mechanical Engineering Department, Izmir, Turkey, 2009.
- [20] - [Wayne Wood](#), "A Brief Test on the Efficiency of a .NET 4.0, Parallel Code Example", 2010.
- [21] - Joseph Albahari, "Threading in C#", O'Reilly Media, 2012 .

-
- [22] - MSDN Library from Microsoft , "How to: Write a Simple Parallel.ForEach Loop" .
- [23] - Alexander Lee Jackson, " A Parallel Algorithm For fast Edge Detection on the Graphics Processing unit", Master's thesis, Faculty of the Department of Computer Science, Washington and Lee University, 2009.
- [24] - Luong Chi Mai, " Introduction to Image Processing and Computer Vision ", Department of Pattern Recognition and Knowledge Engineering, Institute of Information Technology, Hanoi, Vietnam, **2005**.
- [25] - Ali Tarhini, Bachelor's thesis, "On software development and algorithms-Parallel Depth-First Sudoku Solver Algorithm", 2011.
- [26] - E. Dokladalova, "Architecture And Parallel Computing For Imagery And Virtual Reality", Submitted journal article, 2011.
- [27] - Bhabatosh Chanda and Dwijest Dutta Majumder, "Digital Image Processing and Analysis", 2002.
- [28] - R.W.Jr. Weeks, "Fundamental of Electronic Image Processing" . Bellingham: SPIE Press, 1996.
- [29] - N. Kim, T. S. Choi and R.S. Ramakrishna, "A New Parallel Vision Environment in Heterogeneous Networked Computing", Society of Photo-Optical Instrumentation Engineers, 1998.
- [30] - Sylvain Paris, "Samuel W. Hasinoff And Jan Kautz, Local Laplacian Filters"(Edge-aware Image Processing with a Laplacian Pyramid) , Journal ACM Transactions on Graphics (TOG) , 2011.