Parallel Processing of Large Satellite Images Via Threads On a Local Area Network

By
Osama Omar Hasan Quzmar

Supervisor
Dr. Ahmad sharieh

**Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in
Computer Science**

**Faculty of Graduate Studies
University of Jordan**

July 2003

This thesis was successfully defended and approved
On: July 27, 2003

Examination Committee	**Signature**
Dr. Ahmad Sharieh, Chairman **Assoc.Prof. of Parallel Processing**	**……**
Dr. Ahmed Aljaber, member **Assoc.Prof. of Algorithms**	**……**
Dr. Moh'd Belal Al-Zoubi **Assis. Prof. of Graphics and Pattern Recognition**	**……**
Dr. Mahmoud A. Hassan **Assoc. Prof. of Radiation Detectors**	**……**
Dr. Ismail Moh'd Ababneh **Assis. Prof. of Parallel Computing**	**……**

Dedication

To

My Parent

My Brothers

My Sisters

And My Wife

Acknowledgement

We acknowledge the exceptional efforts of the people at developing this study, specially Ahmad sharieh for his precious directions.

We would like to thank the discussion committee: Dr. Ahmed Aljaber, Dr. Mohammed Al-Zoubi, Dr. Mahmoud Hassan, and Dr. Ismail Ababneh.

List of Contents

List of Tables

List of Figures

Appendices

Parallel Processing of Large Satellite Images via Threads On a Local Area Network

By
Osama Omar Quzmar

Supervisor
Dr. Ahmad sharieh

Abstract

With the Rapid development of remote sensing and observation technology, such as satellite images data needs large-scale processing and storage systems that provide a high performance at low cost. Parallel processing and distributed computing is rapidly becoming the standard base for high performance and large-scale computation. Machines connected via a Local Area Network (LAN) can be used for centers that do not have multi-processors machines.

This thesis proposes a solution that decomposes an image into sub-images; the image processing is then applied to each sub-image. Creating a number of threads on the server and three threads on each client around the LAN. The multithreaded to read, process, and write the sub-image can run concurrently.

This solution reduces both: time, which can be the main challenge for processing large images, and the memory requirement. The measurements and results of

experiments indicate that multithreads on a LAN increase the efficiently of distributed LAN. The tested samples of data show that nine threads are the best for cost effective application on the proposed system. The efficiency (E) for threads: three to nine threads is higher than 80% for images of size 3120x2990.Although, the highest value of E = 0.95 is achieved when the ratio of image size to number of threads was 3,109,600 pixel/thread, where number of threads was three. These indicate that, the ratio of image size to number of threads about six to seven million pixel/thread will produce a high performance.

1. INTRODUCTION

1.1 Introduction

This chapter gives an introduction to the existing digital image processing techniques, parallel processing techniques, the research problem, the study objectives, and investigation of procedures described in this thesis.

A literature survey was undertaken to study the existing parallel image processing. Thread was selected for this study. Indeed, a fewer techniques of parallel processing that completely depend on threads, specially, in the case of parallel processing for satellite images, are used.

The field of parallel processing is concerned with architectural and algorithmic methods for enhancing the performance or other attributes (e.g. cost effectiveness, reliability) of digital computers through various forms of concurrency. Even though concurrent computational has been around since the early days of digital computers. Recently, it has been applied in a manner, and on scale to better performance as super computers (Behrooz Parhami, 1999).

Threads are known as a lightweight process and may run concurrently to perform a task. A thread-driven architecture reduces over head. It reduces the amount of information regarding the state of execution of a process that has to be saved and loaded in to memory. They share sections of the same memory areas
(Yunqing P. Zhang, 1999).

The Threads are very small compared with processes. Thread creation is relatively cheap in terms of CPU costs; threads are like memory frugal, they increase the performance in a uniprocessors environment. When application performs operation on file or socket I/O (Thuan and Pankaj, 1999).

A literature review of existing parallel processing for digital images standards, and techniques to improve parallel parameters such as high-performance are discussed in Section 1.3. The thesis objectives are explained in Section 1.4. And the thesis outlines are given in Section 1.5.

1.2 Problem Statement

Since remote sensing image data (satellite images) process two dimensional image data, it needs very high load, very high space (memory requirement) in computations. The processing load has been increased year after year by the following factors (Fokuda et al, 2000):

1- Increment of processing requirement: Demand for data has been increasing, because use of remote sensing image data has become available in various fields.

2- High-resolution of sensor: Size of image data becomes much bigger than before. Because resolution of the surface of the earth was enhanced by technological advanced and much information quantity data could acquire.

3- On-board recording and data relay: Data Acquired from satellites have been increasing, because we can observe the surface of the earth globally through the data recording by On-broad and data-relay satellite.

4- Complexity of processing algorithm: Growing number of processing has been performing the same quality data.

5- Requirement to processing software and environment of hardware/software had been changed.

In addition to these situation, some countries like Jordan do not have supercomputers, but they have LAN.

1.3 Literature Review

The areas covered in the literature survey are Parallel implementation, and performance analysis.

Chaver et al. (2002) discussed in their several issues relevant to the parallel implementation of a 2-D Discrete Wavelet Transform (DWT) on general purpose their interest in this transform is motivated by its usage in an image fusion application which has to manage large image sizes, making parallel computing highly advisable. They have also paid much attention to memory hierarchy exploitation, since it has a tremendous impact on performance due to the lack of spatial locality when the DWT processes image columns.

Kasahara et al. (2001) presented the implementation scheme of the automatic coarse grain task parallel processing using OpenMP API as an example of realization and its performance on an off the shelf SMP machine. OSCAR compiler generates coarse grain parallelized code, which forks threads only once at the beginning of a program and joins only once at the end. To minimize the overhead though hierarchical, coarse grain task parallelism are automatically exploited. In the performance evaluation, the OSCAR compiler with XL Fortran compiler gave us scalable speed up for application programs in perfect and significant speed-up compared with native XL Fortran.

Hawick et al.(2002) described the distributed computing approaches they have used to integrate bulk-data and meta-data sources. The grid computing techniques they have used embed parallel services in an operational infrastructure. They described some of the parallel techniques for data assimilation; for image and map data processing; for data cluster analysis; and for data mining. They also discussed issues related to emerging standards for data exchange and design issues for integrating together data in a distributed ownership system. They include a historical review of their work in this area over the last decade which leads us to believe parallel computing will continue to play an important role in Geographical Information System (GIS). They speculate on algorithmic and systems issues for the future. They get a good high-performance result.

Hawick and James (1999) describe distributed algorithm for processing remotely sensed data such as geostationary satellite imagery. They built a distributed data repository based around the client-server computing model across wide-area ATM networks, with embedded parallel and high-performance processing modules. They focus on algorithm for classification, analysis of the data. They consider characteristics of image data collected from the Japanese GMS5 geostationary meteorological satellite. They have measured good speed-up performance for the parallel and distributed components of their system.

Yang and hung (2000) show that, scalable computing clusters, ranging from a cluster of PCs or workstations to symmetric Multiprocessors (SMPs), are rapidly becoming the standard platform for high-performance and large-scale computing. To utilize the parallelism of cluster of SMPs, they present the basic programming techniques by using

PVM to implement a message-passing program. The experimental results show that their Linux/PVM cluster can achieve high speed-up for application.

Giess et al. (1999) presented the design and implementation of a parallel system for interactive segmentation and visualization of three-dimensional medical images which is distributed on a heterogeneous cluster of work stations or personal computers. All image-processing functions are multithreaded to use the advantages of symmetric multiprocessors. Its platform-independence is achieved using standardized libraries like message passing interface (MPI) and POSIXthreads.

The use of MPI and POSIXthreads allows interoperability in heterogeneous environment. With this system, they achieved the integration of several image processing and visualization algorithm into one parallelization scheme avoiding repeated redistribution of large volume data.

Peppers (Taylor et al., 1999) used a distributed parallel computing, tile-based pipelined approach to process remotely sensed image data products, which are characterized by their very large image size (often more then 500MB). The method made use of the inherently localized nature of most images processing algorithms to formulate the division of the effort or labor, both spatially across the image and temporally along the pipeline of transformations. This formulation can be used to split the load among all available processing units, either using multiple threads of execution on a multi-CPU computer, or using a message passing technique (MPT) to coordinate multiple processes distributed across a network cluster of computer.

CASA CALCRUST application was develop to interactive 3D-visualization program for geophysical application whose input data includes seismic, landsat, and topographic databases distributed over a wide-area network along with a heterogeneous collection of supercomputer server (Bergman et al., 1998). The output was a high resolution, full-color view of the earth's crust that can be rotated, zoomed, sliced, and "flown over" by a scientist sitting at a workstation. With conventional technology, each image requires weeks of computation on super-minicomputers, but it was the goal of this effort to distribute the application among the supercomputer on the CASA network to allow renderings to be produced in under one second.

Pathfinder inputs sensor data reading obtained from satellite traversing polar orbits and output a 12 band, 8km-resolution image of the world. In this study, the goal is to map Pathfinder to a multiprocessor (Saltz et al., 1999). It partitions the output image horizontally (by latitude) and to have each processor sample the input to determine the map between the input data and the output image. The output image is too large to be stored (used for application), so the process of carrying out successive update to the output image is formulated as an out-of-core computation. Once updates are completed, rewriting the partitioned image from local disks and local memories into persistent storage produces the final results.

1.4 Thesis Objectives

This thesis concentrates on using the advantage of thread in parallel computing for processing large satellite images. The primary objective is to simulate and implement a parallel processing of large satellite images via threads on a Local Area Network

(LAN). It aims to reduce the amount of computation for loading large satellite images, and reduce the amount of space (memory requirement).

1.5 Thesis Outline

This section gives a brief description of each of the following chapters.

Chapter Two: Satellite image processing and parallel processing: It includes four sections, section 2.1 Remote Sensing: describes remote sensing, what is it?, Its characteristics, related fields of remote sensing, and technical fundamentals of remote sensing. Section 2.2 (GIS): it describes the GIS definition, and it's applications.

Section 2.3 Fundamentals of satellite images and image processing describes the basics of satellite images, types of satellite images, storage and supply of digital satellite images, and image structure. Digital image processing will be presented in this section as digital image definition, image-processing definition, and reasons for digital image processing, digital image processing operation, image histogram, and spatial filtering.

Section 2.4 parallel processing and implementation: describes parallel processing methods, concept and terminology in parallel processing, types of parallel computations. Parallel implementation measurments will be taken into account like speed-up and efficiency to show performance of computing, parallel programming models, design parallel program, synchronization and parallel processing with parallel processing.

Chapter Three: Threads programming and application: describes related concepts and terminology of threads, synchronization concepts, advantage and disadvantage of threads, challenges of multithreaded programming. The chapter describes threads interface, thread management. The most important section in this chapter is threads in Distributed applications.

Chapter Four: Simulation of the system: describes the definition of the system, how it works, its simulation procedure, its input and output, and its adaptation to run on a LAN. Finally the system will be tested on collecting large satellite images. The implementation results with their analysis and performance comparison will be presented.

Chapter Five: conclusions, suggestion, and future work: This chapter gives the conclusions, suggestion, and future work.

2. SATELLITE IMAGE PROCESSING AND PARALLEL PROCESSING

There are numbers of people who want to use remotely sensed data and Geographical Information System (GIS) data. The different applications that they want require increasing amounts of spatial, temporal, and spectral resolution. Some users are satisfied with a single image a day, while others require many images per an hour (Escobar et al., 2000).

The subject matter of satellite remote sensing is expanding very rapidly. In only two and a half decades the exacting technology of earth observation satellites has progressed from experimental and limited to quasi-operational and global. The next decade will see yet more operational satellite systems for earth observation, with developments such as the polar platform and imaging radar system (Gupt and Ravi.P, 1991). By the next century this little planet will be closely and continuously monitored by a band of satellites and sensors in space.

2.1 Remote Sensing

Remote sensing is defined as the art and science of obtaining information about an object by a device that is not in direct contact with it (Sabins and Jr.F, 1987). In a sense most information is obtained by remote sensing since even our eyes are not in direct contact with the object we see. However, in a technological context, remote sensing usually refers to data gathered by sensors and instruments that measure emitted or reflected electromagnetic radiation which is formatted digitally so it can be viewed pictorially or analyzed by computers (Sanchez and Canton, 1999).

Data acquired by remote sensing can be converted into information by visual study, computer analysis and classification, microcomputer processing, or by a combination of visual and digital analyses (Richason, 1983).

Satellite remote sensing is the use of sensors, normally operating at wavelengths from the visible (0.4 μm) to microwave (25cm), on board satellite to collect information about the earth's atmosphere, ocean land and ice surface. Commonly the information is collected in two-dimensional form an array of digital data. In atmosphere and oceanographic application the data collection may be one-dimensional, for example, the vertical temperature profile of the atmosphere (Harris, 1987).

Remote sensing satellites orbit the Earth at the variety of altitude from low polar (200km) to high equatorial (36,000km). Their sensor gathers electromagnetic energy reflected emitted or back scattered from part of the Earth-atmosphere system below the satellite (Harris, 1987).

There are some of the reasons why satellite is a useful source of information. One of these reasons is that the data from satellite remote sensing commonly in a form suitable for computer processing. In fact, the vast quantities of data produced by satellite remote sensing demands rapid computer processing. For example, a Landsat Thematic Mapper (TM) scenes has 6500x6920 pixel and seven wavebands: a total of 273 Mbytes of data for just one scene (Harris, 1987).

2.1.1 Related Fields

The images obtained by satellite and spacecraft are varied in nature. In the first place, the imaging instrument can be aimed toward earth, or toward any other part of the universe. In the context of remote sensing concentrate on images obtained by earth-orbiting and earth observing satellites and let the astronomers and planetary scientists deal with images of other celestial bodies (Sanchez and Canton, 1999). For example,

1- Satellite images contain geographical information that is useful for creating or updating charts or maps. In this case, the image processing operation refers to the geophysical sciences, such as geodesy and cartography.

2- Satellite images of the earth are of interest to plant science. The detailed investigation of plant life is possible because the living leaf shows a unique spectral behavior. For this reason, remotely sensed images are used in classifying and mapping vegetation.

3- Remotely sensed images give geo-scientist a board-scale perspective that is not available from ground observation, as well as a measurement of the reflected and emitted wavelengths of large-scale geological objects.

2.1.2 Technical Fundamental

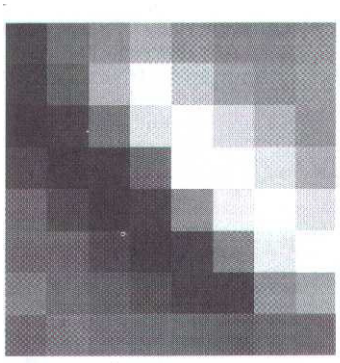
Remote sensing is a relatively new field: many of its fundamental facts are not yet known. Every year new investigation methods and interpretation of exiting data are discovered. However, there are certain scientific certainties and research principles that appear to be fundamental to the technology (Sanchez and Canton, 1999). For example,

1- Every sensor is limited by the size of the smallest area that can be recorded and stored. A sensor's resolution as being of so many kilometers or meters. For example, the multi-spectral scanner (MSS) on board the Landsat satellites has a

resolution of approximately 80m, while the Thematic Mapper (TM) instrument on the same series of satellites has a resolution of approximately 30m. These numbers indicate the size of the discrete elements in the image, called picture element or pixels.

2- Different objects or features may have different intensities in their radiometric response over the same spectral region. This feature of remotely sensed images allows detecting objects and details of the landscape based on their relative brightness.

3- Most remote sensed images are stored in digital format by first dividing the image into small, equal-sized areas called pixels and then assigning to these areas a value depending on its brightness. In this manner, graphic information can be represented digitally in order to make possible the use of computerized quantitative processing methods. By the same token, digitally stored images can be displayed graphically using conventional computer graphics techniques. Figure 2.1 shows a portion of an image and the corresponding digital values. The square blocks represent pixel densities in the range 0 to 255. Black is represented with a pixel value of 0 and white with value 255. The values between 0 and 255 are a scale of gray shades. Although the digital values of the image pixels are often derived directly by the sensor device, these values can also be obtained from the image data by means of an optical scanner's device. The digital values can be used to reconstruct the image using standard computer graphics operations.



20	128	180	220	180	160	165	160
20	25	128	230	250	240	220	150
20	25	128	230	250	240	220	160
25	20	15	120	245	250	240	170
110	25	20	25	180	230	255	240
135	125	30	20	30	180	245	250
120	128	120	80	20	25	150	170
110	128	110	120	110	100	120	110

(a) (b)

Figure 2.1 a) a portion of an image b) corresponding digital values

2.1.3 Types of Remote Sensors

Sensors can be divided into two groups: passive and active. Passive sensing resembles the way our eyes operate, by detecting the radiation reflected by the sensed objects from the sun (or another source of illumination). Active sensing is achieved in the reverse manner: the energy is transmitted by the sensor itself, bounces back on impact with the object, giving a 'backscatter' echo which is registered by the sensor. In remote sensing, microwave radar (Radio Detection and Ranging) operates by active sensing (Kingston center of GIS, 2000).

2.2 Geographic Information System (GIS)

A GIS is a system of hardware, software and procedures to facilitate the management, manipulation, analysis, modeling, representation and display of georeferenced data to solve complex problems regarding planning and management of resources (Escobar et al., 2000).

A more comprehensive and easy way to define GIS is the one that looks at the disposition, in layers (Figure 2.2), of its data sets. "Group of maps of the same portion of the territory, where a given location has the same coordinates in all the maps

included in the system". This way, it is possible to analyze its thematic and spatial characteristics to obtain a better knowledge of this zone (Escobar et al., 2000).

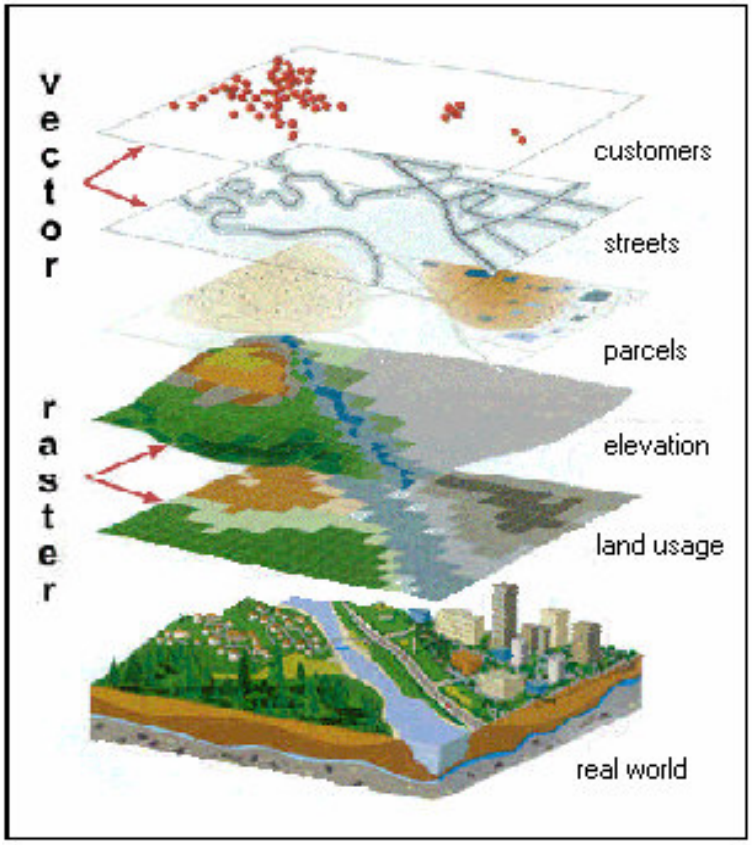


Figure 2.2 The concept of layers in GIS

GIS systems store and process data in tow formats, vector and raster. In vector data model, the world is represented as a mosaic of interconnecting lines and points representing the location and boundaries of geographical entities. The raster data model has come out of aerial and satellite imaging technology, which represents geographical objects as grid-cell structures known as pixels (Kingston center of GIS, 2000).

In broad terms, a Geographic Information System could be defined as a set of principles and techniques employed to achieve one (or both) of the following objectives (Kingston center of GIS, 2000):

1- Finding a suitable location that has the relevant attributes. For example, finding a

suitable location where an airport, a commercial forest or a retail outlet can be established. This is usually achieved through the use of Boolean (logical) operations.

2- Querying the geographical attributes of a specified location. For example, examining the roads in a particular locality, to check road density or find the shortest path, and so on. This is often achieved by 'clicking' onto the location or object of interest, and examining the contents of the database for that location or object

GIS Application

Mapping locations, Mapping quantities, Mapping densities, Finding distances, and Mapping and monitoring change are all applications of GIS.

1- Mapping locations: GIS can be used to map locations. GIS allows the creation of maps through automated mapping, data capture, and surveying analysis tools.

2- Mapping quantities: People map quantities, like where the most and least are, to find places that meet their criteria and take action, or to see the relationships between places. This gives an additional level of information beyond simply mapping the locations of features.

3- Mapping densities: While you can see concentrations by simply mapping the locations of features, in areas with many features it may be difficult to see which areas have a higher concentration than others. A density map lets you measure the number of features using a uniform real unit, such as acres or square miles, so you can clearly see the distribution.

4- Finding distances: GIS can be used to find out what is occurring within a set of distance features.

5- Mapping and monitoring change: GIS can be used to map the change in an area to anticipate future conditions, decide on a course of action, or to evaluate the results of an action or policy.

2.3 Fundamentals of Satellite Images and Image Processing

Many types of remote sensing images are routinely recorded in digital form and then processed by computers to produce images for interpreters to study. The simplest form of digital image processing employs a microprocessor that converts the digital data tape into a film image with minimal corrections and calibrations. At the other extreme, large mainframe computers are employed for sophisticated interactive manipulation of the data to produce images in which specific information has been extracted and highlighted.

Satellite has a spatial resolution of about 1 kilometer and becomes increasingly distorted further a way from the orbit path because of the wide viewing angle and the curvature of the earth. After data are transmitted to ground stations, the measurements are calibrated and corrections are applied. Data are corrected geometrically to reduce distortion, and location information is added. The data are then released for further processing (Anderson and Stonbarker, 2000).

Types of satellite images

Satellite images can be classified as in table 2.1

Table 2.1 classification types of satellite images.

Image Type	Explanation	Featurs
Standard Infrared Satellite image	Give an excellent representation of the location and intensity of the actual clouds despite being based upon temperatures. The temperatures shown are the highest levels of clouds above the earth. When there are no clouds at all, the temperature of the earth is sensed and this is typically warmer than if there were clouds above the location (Canada Avalanche Association, 2000).	Sometimes on clear, very cold nights the temperature at the earth's surface is colder than temperature a loft (this is referred to as an inversion).
Enhanced Infrared Satellite image	Similar to standard infrared satellite images, it turns from white to vibrant colors with colder temperatures. Highlighting the stronger storms. The raw satellite data also allows the user to see more detail than the standard image, as well as land and sea surface temperatures (AccuWeather, 2000).	It is difficult to see the small differences in temperature of the clouds near the edge of the scale, and turning gray to white is sometimes not visible enough to the naked eye
Visible Satellite image	It is literally photographs of the earth. It shows clouds in the visible portion of the electromagnetic spectrum. They will be blank during the night (Commonwealh of Australia, 2002).	Visible satellites generally show low clouds, fog and snow cover well than infrared images and can very detailed
Visible Infrared Satellite image	Combines the visible satellite with the infrared satellite to make one image, which shows the most detail, possible, 24 hours a day. Often quick moving upper-level clouds can be seen skirting over lower-level clouds, especially in animation. During the night time the image is a standard infrared image, which is gray-scale, with black being the hottest temperatures, and white being the coldest. Generally colder cloud tops indicate stronger storms (Commonwealh of Australia, 2002).	When the sun rises, the standard infrared image is mixed with the visible satellite image. Blue areas are from the infrared; yellow areas are from the visible; white area is a mix of two. Without changing to a separate image or comparing images, you are able to see the temperature data from the IR
Water vapor Satellite image	Shows detailed water vapor data on an eye-appealing scale from deep red to bright green (dry to moist) which enables the user to easily pick out upper-level circulation's and intrusions of wet or dry air (AccuWeather, 2000).	Water vapor images primarily show moisture from 18000 feet above the earth's surface to the top of the atmosphere. help in identifying upper level winds and jet streams for use as a tool to

		predict weather system movement

2.4 Digital Images

A digital image is composed of discrete point of gray tone, or brightness, rather than continuously varying tones. It is composed of a rectangular array of pixels, each representing a particular (x,y) location, and each with a digital brightness data value. The digital image exist merely as a large array of numbers (data values) that, when arranged properly and displayed as brightness, forms an image (Baxes, 1994).

An image is generally sampled into a rectangular array of pixels. Each pixel has an (x,y) coordinates that corresponds to its location within the image. The x coordinate is the pixel's horizontal location; the y coordinate is its vertical location. We can think of x location as the column in which the pixel is located and the y location as the raw in which the pixel is located, the pixel at location (0,0) is in the upper left corner of the image (Baxes, 1994).

Satellite remote sensing produces very large quantities of digital data. A single Landsat TM scene covering a ground area of 170Km x 185Km contains 273 Mbytes of data and occupies seven magnetic tapes when written at a tape density of 1600 bits per inch. Each Satellite Probatoire d' Observation de la Terre (SPOT) High-Resolution Visible (HRV) multi-spectral scene covering a ground area of 60Km x 60Km contains 27 Mbytes of data, and the SPOT operators have confirmed that the data supply is assured for over a decade.

National Oceanic and Atmospheric Administration (Noaa) satellites can produce over 2,500 Mbytes of data from the Advanced Very High Resolution Radiometer (AVHRR) each day. Clearly there is a data mountain in satellite remote sensing, and equally digital processing is the only sensible way of handling these vast quantities of information about our environment (Harris,1987).

2.4.1 Image Processing

Image processing, in general terms, refers to the manipulation and analysis of pictorial information. In this case, pictorial information means a two-dimensional visual image. Any operation that acts to improve, correct, analyze, or in some way change an image is called image processing (Baxez, 1994).

Certainly the most powerful image processing system we see and use every day is the one composed of the human eye and brain. This biological image processing system focuses, acquires, enhances, restores, analyzes compresses, and stores images at astounding rate.

2.4.2 Storage and Supply of Digital Satellite Images

Many types of remote sensing images are routinely recorded in digital form and then processed by computers to produce images for interpreters to study. The simplest form of digital image processing employs microprocessors that converts the digital data tape into a film image with minimal correction and calibrations. At the other extreme, large mainframe computers are employed for sophisticated interactive manipulation of the data to produce images in which specific information has been extracted and highlighted (Sabins, 1987).

Most commonly, the remote sensing data, as received from satellites, is stored on High Density Digital Tapes (HDDT's) as in Figure 2.3. The data is partially corrected, Per-processed and reformatted for supply to users in various media. The computer compatible tape is the most widely used medium to distribute digital data. The flexible diskette is also used to distribute data; however, they have limited storage capacity and are normally used for pack-up purpose. Further, with the development of the electronic technology, there seems to be a good prospect of utilizing compact disks (CD-technology) for this purpose in the future, additionally, as is obvious, the scanner digital imagery is widely used as analogue film products for visual interpretation (Gupta, 1991).

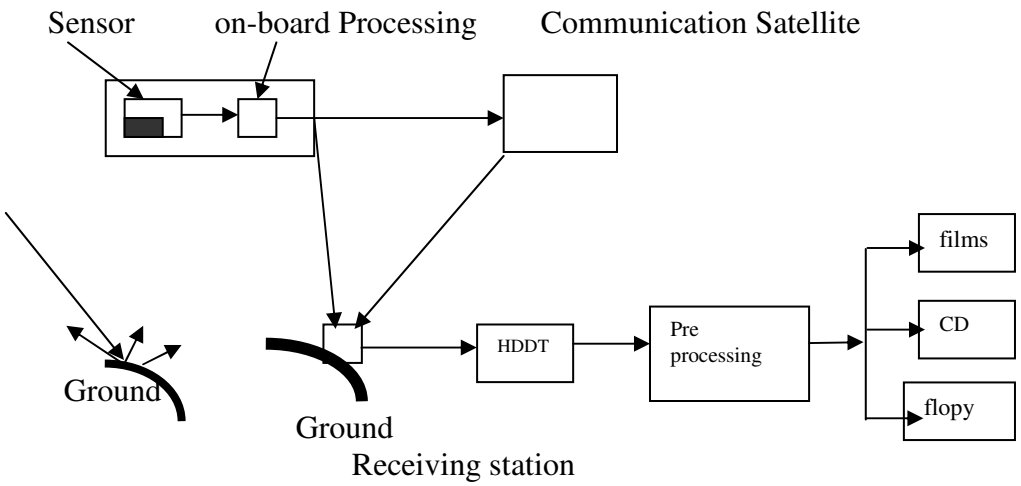


Figure 2.3 Schematic of data flow in a non-photographic remote-sensing mission.

However, the storage and distribution on film has several limitations:

1- The quality of the film may deteriorate with age.

2- Reproduction, for example for distribution is accompanied by loss of information.

3- Repeatability of result is not ensured.

4- Out of the total range of 128 or 256 gray levels, the human eye on film products differentiates only 20-30.

5- Digital processing requires re-conversion into digital data.

2.5 Image Structure

One can think of any image as consisting of tiny, equal areas, or picture elements, arranged in regular rows and columns. The position of any picture element, or pixel, is determined on an XY coordinate system; in the case of landsat images, the origin is at upper left corner of the image. Each pixel also has a numerical value, called a digital number (DN), that records the intensity of electromagnetic energy measured for the ground resolution cell represented by that pixel. Digital numbers range from zero to some higher numbers on a gray scale. The image may be described in strictly numerical terms on a three-coordinate system with X and Y location each pixel and Z giving the DN, which displays a gray-scale intensity value. Images may be originally recorded in this digital format, as in the case of Landsat. An image recorded initially on photographic film may be converted in to digital format by a process known as digitization (Sabins, 1987).

2.6 Image Analysis

Image Analysis operation generally do not produce pictorial results. Instead, they produce numerical or graphical information based on characteristics of the original image. Image analysis breaks an image into discrete objects and then classifies those objects using some measurement process. Additionally, an analysis operation can produce image statistics. Common image analysis operation includes extraction and

description of scene and overall image features, automated measurements, and object classification (Baxes, 1994).

2.7 Image Histogram

A histogram is a graphic plot of pixel values within an image. The simplest histogram, called a brightness histogram, shows the distribution of brightness levels by representing pixel intensities along the horizontal axis and the number of pixel along vertical axis. The brightness histogram is a powerful tool for image processing operation since it provides information about the distribution of gray levels, which can be used in many transformations (Sanchez and Canton, 1999).

The calculation of image histogram requires sampling every pixel value while keeping count of number of pixel at each intensity level. For example, a pixel intensity range of 8 bits (0 to 255) can be represented along the x-axis, while the count of the number of pixels at each intensity level is represented along the y-axis (Baxes, 1994). The resulting histogram allows us to visually appreciate how he gray levels are distributed in the image. One objection to this scheme is that the vertical size of the histogram is undetermined. For example, an image with 1000 pixels in which half of them fall at the same intensity level would require 500 vertical units, while another one in which the highest intensity level had only 100 pixels would generate a graph one-fifth the vertical size. A possible solution is to normalize the vertical element of the histogram by assigning a maximum height to the most abundant intensity level and

making all other pixel counts a function of this most abundant one. Figure 2.4 show normalized histogram (Sanchez and Canton, 1999).

The horizontal axis of the histogram show pixel intensities, the distribution pattern can be used to determine image contrast as ell as its prevalent lightness or darkness. if most of the image data is concentrated toward the lower brightness values, then the image is dark. On the other hand, if the data concentrated towards the higher brightness values, then the image is bright. If the data is distributed though a large portion of the brightness rang, the image shows high contrast; if the contrast is not low (Sanchez and Canton, 1999). Figure 2.5 shows these four basic image types.

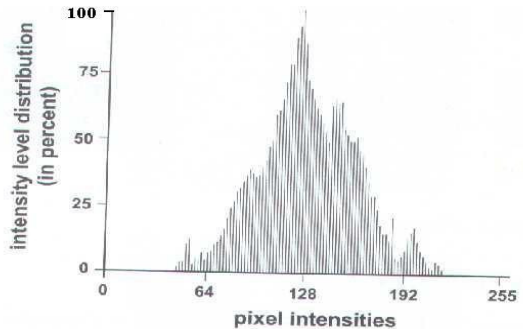


Figure 2.4 Normalized histogram

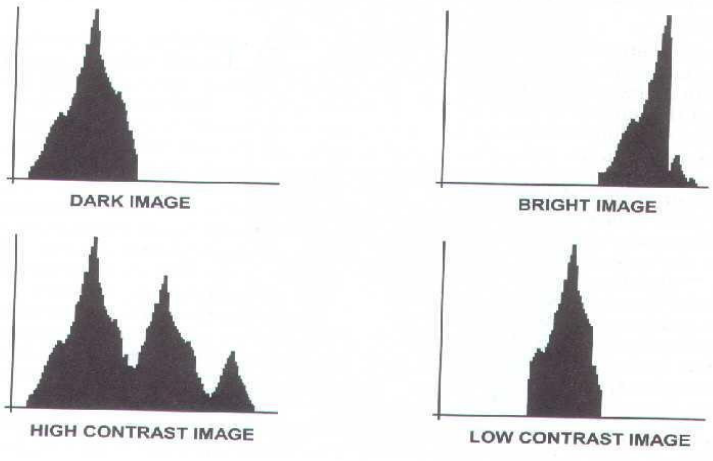


Figure 2.5 Histogram of basic image types

2.8 Spatial Filtering

Before any remote sensing image can be made available for mapping, it must be cleared of errors caused by geometric displacement or atmospheric interference. The common sources of error in satellite imagery are Orbit and flight errors, Earth curvature, Earth rotation, Rough terrain, and atmospheric interference (cloud, fog and smog).

Digital correction of image errors is made possible by computerized image processing techniques. Remote sensing images are normally kept in digitized form (recorded in binary code). Correcting the geometric errors of individual images is usually undertaken by giving the computer the correct coordinates of a number of control points (points whose position is known exactly) on each image (Sanchez and Canton, 1999).

Spatial filters are implemented by means of a process called spatial convolution or finite impulse response (FIR) filtering. In this case, a two-dimensional pixel kernel is moved across the image, pixel by pixel. The result of a mathematical operation on the elements in the kernel is placed in the output set. The calculation is defined as a linear process since each of the elements in the kernel is multiplied by a constant factor called the convolution coefficient (Spencer, 2000). The convolution coefficient (cc) can be expressed as in equation 2.1

$$cc = \frac{f}{e} \qquad \text{.....(2.1)}$$

Where *f* is a multiplier and *e* is the number of element in the kernel. The factor *f* can vary for different element in the kernel, while *e* remains the same. For a 3 by 3 kernel, the convolution coefficients are labeled as follows:

a b c
d e f
g h i

The array of cc is called convolution mask. Every pixel in the input image is evaluated with its eight neighbors, using this mask to produce an output pixel value as show in Figure 2.6

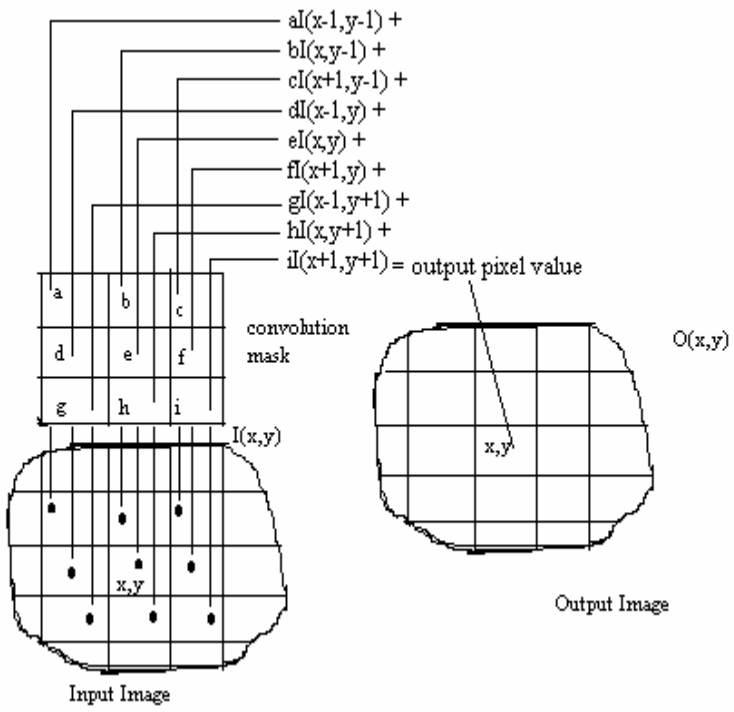


Figure 2.6 The spatial convolution process for the input pixel at location I(x,y), creating the output pixel at location O(x,y).

The result is placed in the output image at the same center pixel location. This process occurs for each pixel in the input image. Equation 2.2 computes the spatial convolution process

$$O(x,y) = aI(x-1,y-1) + bIi(x,y-1) + cI(x+1,y-1) + dI(x-1,y) + eI(x,y) + fI(x+1,y) +$$

$gI(x-1,y+1) + hI(x,y+1) + iI(x+1,y+1)$ ……..(2.2)

Every input pixel is processed through the equation, creating a corresponding output pixel value (Sanchez and Canton,1999).

Once the convolution coefficient has been determined for every kernel element, then the calculation of the output value consists of multiplying each value input by the cc and adding all of them. Figure 2.7 shows a convolution mask applied to an input pixel.

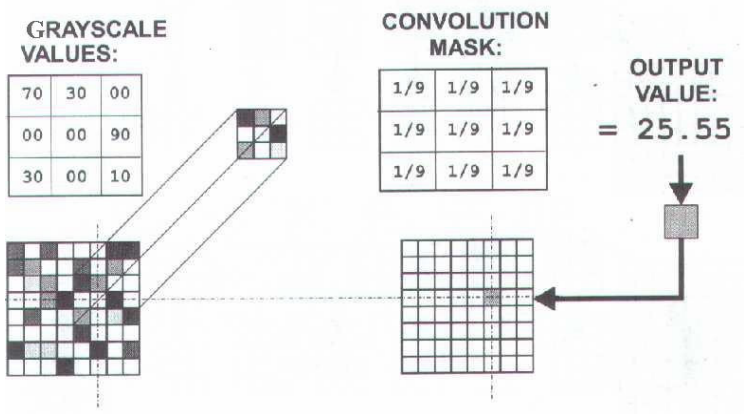


Figure 2.7 Use of convolution mask

2.8.1 Sharpening filter

Sharpening filters help reduce blur in an image by raising the proportion of its high frequency components. Because of this they are also known as high pass filters. They allow high frequencies to pass unchanged but attenuate low frequencies. It's a neighborhood operation, which passes a small window (often 3x3 or 5x5 pixels in size), over an image and modifies each pixel based on its neighbors

(Sanchez and Canton, 1999). A common sharpening pass mask is composed a nine in the location with –1s in the surrounding locations:

-1 -1 -1
-1 9 -1
-1 -1 -1

The most visible effect of a sharpening filter is a general increase in the contrast, which makes the resulting image appear sharper than the original one. Figure 2.8 is an example of image sharpening as applied by sharpening filter.

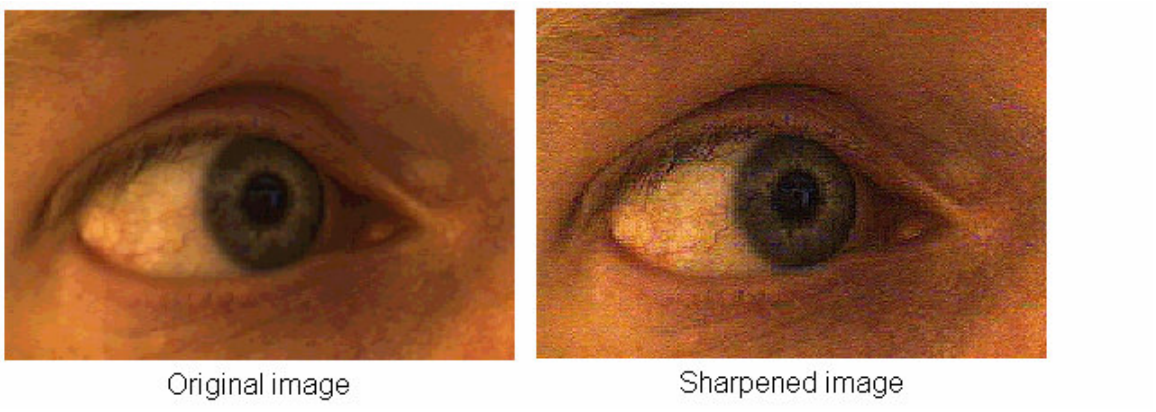


Figure 2.8 Sharpening filtering

2.8.2 Smoothing filter

Smoothing filter reduce noise in an image by lowering the proportion of its high frequency components. Because of this they are also known as low pass filters.
They allow low frequencies to pass unchanged but attenuate high frequencies.
The most visible effect of a smoothing filter is a general softening of the contrast which makes the resulting image appear less sharp that the original one. A common smoothing convolution mask is composed of all nine coefficients having the value of `1/9`:

$$\begin{matrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{matrix}$$

This mask carries out a straight pixel brightness averaging process, as describe earlier. It is often referred to as 1 (9 x 1/9 = 1) and that they are all positive numbers. These tow facts hold true for all low-pass masks. A smoothing filter is a neighborhood function, it

emphasizes areas with gradual change and de-emphasizes areas with rapid change. This effectively smoothes an image and makes edge features less apparent. Figure 2.9 is an example of image smoothing.

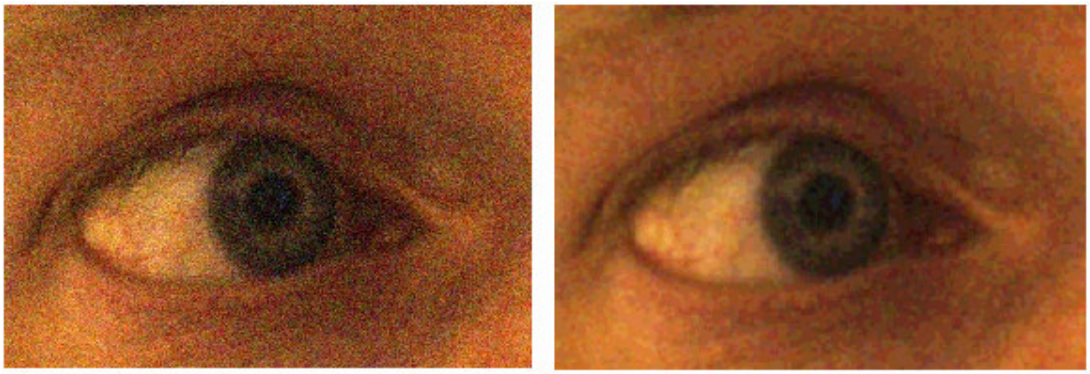


Image with noise Same image after smoothing filtering

Figure 2.9 Smoothing filtering

2.9 Parallel Processing and Implementation

Scientists and engineers are continually looking for ways to test the limits of theories, using high-performance computing to allow them to simulate more realistic systems in greater detail. Parallel computing offers a way to tackle these problems in a cost-effective manner.

Current strategies for supporting high performance parallel computing often face the problem of large software overheads for process switching and interprocessor communication (Hum et al., 1994). These overheads limit the range of applications, which can be effectively mapped to this architecture. While some architectures have been proposed to deal with the problems of overheads, like thread, which is a new approach, can execute a wide range of application efficiently.

Parallel Computing

Traditionally, software has been written for serial computation, to be executed by a single computer having a single Central Processing Unit (CPU). Problems are solved by a series of instructions, executed one after the other by the CPU. Only one instruction may be executed at any moment in time.

In the simplest sense, parallel computing is the simultaneous use of multiple computing resources to solve a computational problem.

The computing resources can include:

- A single computer with multiple processors;
- An arbitrary number of computers connected by a network;
- A combination of both.

The computational problem usually demonstrates characteristics such as the ability to be:

- Decomposes data space into discrete pieces of work that can be solved simultaneously;
- Execute multiple program instructions at any moment in time;
- Solve in less time with multiple computing resources than with a single computing resource.

There are two primary reasons for using parallel computing:

1- Save time - wall clock time

2- Solve larger problems

Most high-performance modern computers exhibit concurrency (Quinn, 1994). For example, multiprocessing is a method used to achieve concurrency at the job or program level. Instruction prefetching is a method of achieving concurrency at the Inter-instruction level, however, it is not desirable to call every modern computer a parallel computer. The concurrency of many machines is invisible to the user.

Parallel processing is information processing that emphasizes the concurrent manipulation of data element belonging to one or more processes solving a single problem. **A parallel computer** is a multiple-processor computer capable of parallel processing.

Speedup is the time needed for the most efficient sequential algorithm to perform a computation and the time needed to perform the same computation on a machine incorporating parallelism.

Some measure of parallel performance is requiring evaluating the success of any parallel implementation. For this purpose two comparison indices were observed for each work dividing algorithm; speed-up, and processor efficiency (Carriero and Gelernter, 1990). Speed up relates to the time taken to compute a task on an uni-processor system to the time taken to compute the same problem using the parallel implementation. Speed up is thus expressed as in equation 2.3

$$S = \frac{Ts}{Tp} \quad \text{.....(2.3)}$$

The Ts depends on the time of communication between processors and the time of computation on each sub domain. Tp expressed in equation 2.4

$$Tp = T\,comm + T\,comp \quad \ldots\ldots\ldots\ldots (2.4)$$

The relative efficiency, based on the performance of the problem for one processor, can be a useful measure as to what percentage of a processor time is used on computation. Any implementation penalty would be immediately reflected by a decrease in the

relevant efficiency. The efficiency (E) is expressed in equation 2.5

$$E = \frac{S}{P} \quad \ldots\ldots\ldots\ldots (2.5)$$

The solution will be cost optimal if the cost of solving a problem on a parallel computer is proportional to the execution time in a single processor

(Kumar Vipin et al., 1994). The cost is expressed in equation 2.6.

$$Cost = Tp * p \quad \ldots\ldots\ldots (2.6)$$

In terms of comparison, the relative efficiency indicator is the major factor since it has an implied time element in term of implementation overheads and shorter execution. The efficiency factor also gives an indication as to the effective load balancing of the decomposition algorithm (Rallings et al.,1998).

2.10 Parallel Programming

In fact there are many factors must be taken into account when designing parallel programs. There are many methods used for designing parallel programs, and we will discuss some of these methods.

One of the first steps in designing a parallel program is to break the problem into chunks of work that can be distributed to multiple tasks. This is known as decomposition or partitioning (Hord, 1999). There are two basic ways to partition computational work among parallel tasks: domain decomposition and functional decomposition.

In the domain decomposition, the data associated with a problem is decomposed. Each parallel task then works on a portion of the data. Figure 2.10 shows the domain decomposition partitioning.

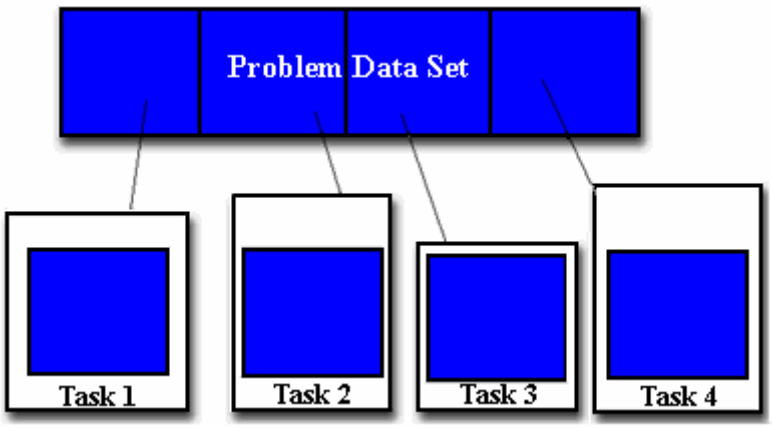


Figure 2.10 Domain decomposition partitioning

There are different ways to partition data that are illustrated in Figure 2.11

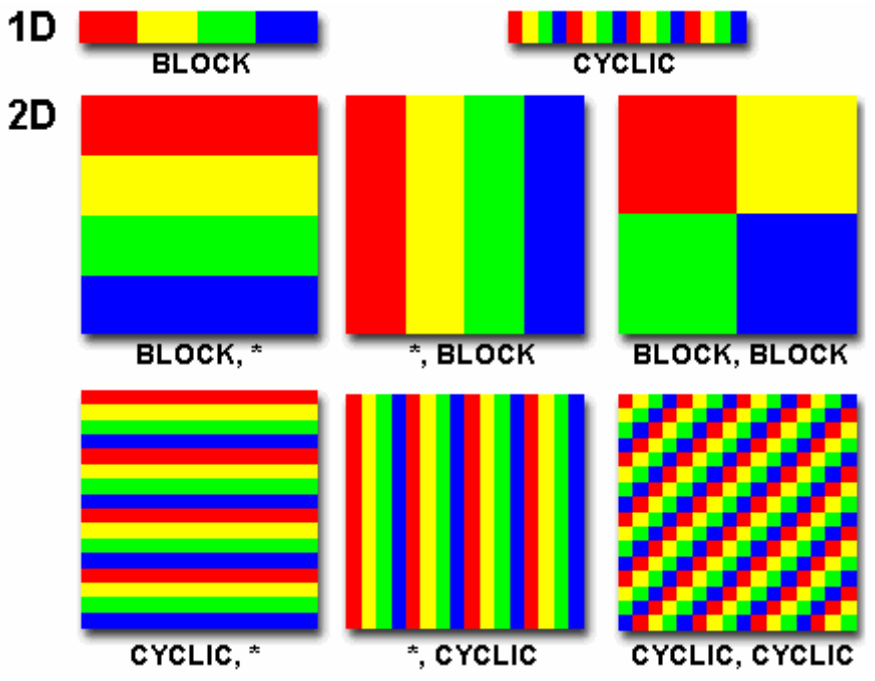


Figure 2.11 Types of domain decomposition

In Functional Decomposition approach, the focus is on the computation that is to be performed rather than on the data manipulated by the computation. The problem is decomposed according to the work that must be done. Each task then performs a portion of the overall work. Figure 2.12 shows functional decomposition.

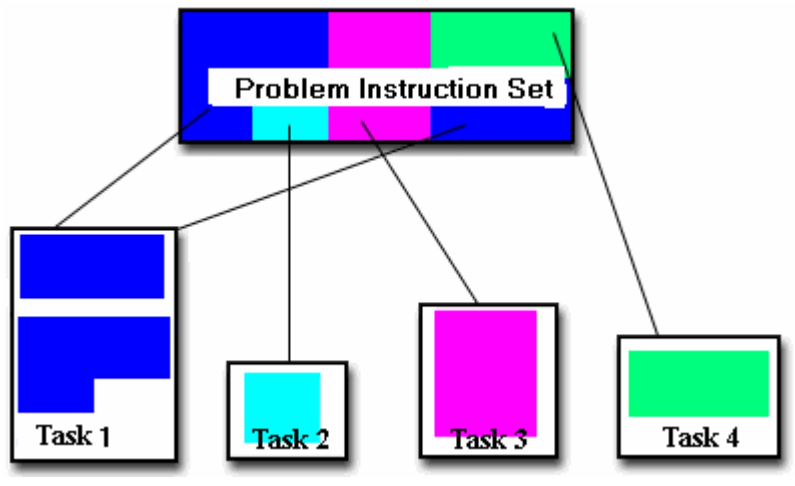


Figure 2.12 Functional decomposition

Functional decomposition lends itself well to problems that can be split into different tasks. For example:

Climate Modeling**: -** Each model component can be thought of as a separate task. Arrows represent exchanges of data between components during computation the atmosphere model generates wind velocity data that are used by the ocean model. The ocean model generates sea surface temperature data that are used by the atmosphere model, and so on.

Load balancing refers to the practice of distributing work among tasks so that all tasks are kept busy all of the time. It can be considered a minimization of task idle time. Load balancing is important to parallel programs for performance reasons. For example, if all tasks are subject to a barrier synchronization point, the slowest task will determine the overall performance as illustrated in Figure 2.13.

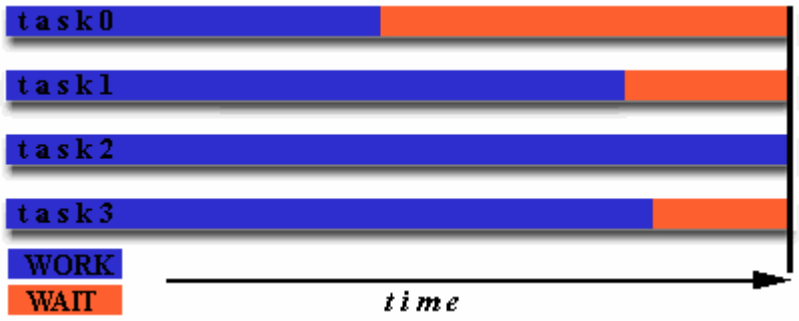


Figure2.13 Load balancing

There are some reason why image processing in an appropriate area for the application of parallel computing (Fountain, 1994). The first reason has tow aspects: the deviation quantities of data which may be involved, and the speed at which processing of these images is required to proceed. There are tow quite separate ways in which this requirement arises.

First, many Image Processing applications occur in environments where he repetition rate for processing images is fixed by some external constraint. The second aspect of the need for speed lies in the requirement to program system to perform the required

tasks. The second reason why image processing is such an apposite area of application for parallel computing. It lies in the structured nature of the data sets involved.

3. THREADS PROGRAMMING AND APPLICATION

The central processing unit (CPU) of a computer is designed to perform one action at time. It designed to perform actions at an extremely rapid rate. The structure and execution of programs reflects one-step-at-a-time point of view. The writers of code create programs by arranging single sequence of statements that will be executed, some perhaps repetitively to form the program's flow of control that moves from one statement to the next. Debugging commands that allow the program to be executed by single steps also show the single, sequential pattern of the program's execution.

Users, however, frequently want to perform many actions, or applications, all at the same time. The ability to be executing more than once application at a time is often described as "multi-tasking" since the computer is engaged in performing multiple tasks simultaneously. The term "concurrency" is also used to describe this effect since the actions are, or at least appear to be, performed at the same time
(Thuan and Panksj, 1999).

3.1 What is a Thread?

A thread known as a lightweight process, which is one of potentially many sub- processes that may run concurrently to perform a task. A thread-driven architecture reduces over head by reducing the amount of information regarding the state of execution of a process that has to be saved and loaded into memory by sharing sections of the same memory areas with other weight processes
(Zhang et al., 1999). Thus, thread is essentially a program counter, a stack, and a set of registers.

Since threads are very small compared with processes, a thread creation is relatively cheap in terms of CPU costs. As processes require their own resources bundle, and threads share resources. Threads are like wise memory frugal, they can increase performance in a uniprocessors environment when application performs that are likely to block or cause delays, such file or socket I/O
(Wagner and Towsley, 2000).

3.2 Multithreaded Programming

Multithreading is a form of concurrent programming _ away to design and implement parallel application programs. Traditionally, a concurrent application has multiple concurrently executing process.

With threads, you can apply concurrent programming concepts and techniques within the address space of a single process as well. Multiple threads enable us to write efficient parallel programs and naturally model programming problems that are inherently parallel. Multiple threads of execution can not share the same address space when an address space is exclusively linked to one thread
(Pham and Grag, 1999).

3.3 Advantages of Threads

Economy and Speed, Shared Memory and Resources, Efficient Hardware Utilization, Remote Services, and Concurrent Programming are all some of multithreaded advantages (Thuan and Pankaj, 1999), (Tom and Don, 2000).

Economy and Speed

Concurrent programs using multiple threads are faster than similar programs with multiple Processes. Using multiple processes to achieve concurrency is expensive because process Creation is slow. On the other hand creation, switching, and destruction of threads are faster because they all share same process's resources and have a little unique context. For This reason, some implementers also refer to threads as lightweight processes.

Shared Memory and Resources

A significant advantage of threads is their ability to share memory and resources within a process. All threads in a process share its entire address space. Since concurrent programs often require communication and data sharing among constituent sequential tasks. These needs can be met efficiently with threads.

Efficient Hardware Utilization

An important advantage of multithreaded programming is the possibility of performing I/O and computations concurrently. This lead to better utilizing different components of a computer's Hardware (CPU, disk, and peripherals). Using multiple threads can truly run in parallel, allowing a process to use more than one processor simultaneously. This increases the computations.

Remote Services

In distributed computing, threads are essential for remote service requests, because there is no guarantee that the results will be available immediately. If a single-threaded client process uses a synchronous communication mechanism, such as remote procedure call (RRC), the entire client process can hang indefinitely while waiting for a result.

Using a multithreaded program, a client program can dispatch a thread to service each remote request, and continue doing other useful work. You can structure a server process cleanly by having a main thread watch the queue of incoming service requests, then dispatch a thread to handle each request. While the server is busy servicing client requests, it will still be available to accept and process additional requests.

Concurrent Programming

Applications with multiple threads of control are naturally concurrent. Concurrent Programming was developed as a technique for addressing the complexity of writing large programs (e.g., an operating system) by allowing a designer to break it up into smaller units and deal with each one independently. With multithreaded programming, similar benefits are now available to application programmers.

3.4 Challenges of Multithreaded Programming

Multithreaded programming involves many threads executing in parallel, and possible interactions between them must be taken into account. (1) When sharing data between different threads, you have to ensure that one thread doesn't corrupt data in another thread. (2) If one thread's activity depends on others, you must synchronize the threads' execution to honor this Dependency (Thuan and Pankaj , 1999).

(3) You have to determine the number of threads in the application. (4) You can create too many threads resulting in poor performance. (5) You always have to consider communication and synchronization costs. (6) You should also be careful that the applications design results in fairness: all threads get a chance to do their work. (7) Debugging a multithreaded program is harder (Thuan and Pankaj,1999).

3.5 Thread Management Concepts

A thread is a basic unit of exestuation. At any given time, one thread gets to execute on a processor while other wait for resources or a chance to run. The operating system decides which thread gets the processor, and for how long, passed upon its internal scheduling algorithms and policies.

Thread States

A thread can be in running or not running state. In Windows NT, a thread can be in one of six states: **ready, standby, running, waiting, transition, or terminated**

In the **ready** state, a thread may be scheduled for execution. The Kernel keeps of the number of ready threads and their priorities. When the dispatcher is ready to pick a thread to run next, it chooses the thread with the highest priority from the set of ready threads. The state of the selected thread changes to standby and the thread waits it's run to run on a processor at the next context switch.

In the **standby** state, the thread designated to run on a particular processor waits for the processor to become available. There can only be one standby thread per processor. If the priority of a standby thread is not high enough to preempt the running thread, it must wait for the next context switch. When the running thread blocks or finishes its time slice, a context switch occurs, and the standby thread begins to run on the processor. At this point, its state changes to running.

The **running** states indicates that a thread is executing instructions on the processor until its time slice runs over, or it is preempted, blocked, or terminal.

In the **waiting** state, a thread is suspended while a waiting for something to happen, such as keyboard or mouse input, the signaling of a synchronization object, and so forth. When the event happens, the thread can directly enter the ready state if the resources it needs to run are available. Otherwise, it goes to the Transition State.

In the **transition state,** a thread is ready to run but the resources it needs are not readily available. When the recourses are available, the thread goes to the ready state, and is enable foe schooling.

In the **terminated** state, a thread can terminate itself, be terminated by other threads, or die when its parent process is terminated. At this point, the thread is removed from the system if there are no pending handles to it. When all handles to a terminated thread are closed, the system may reclaim the thread resources and reinitialize it for another thread creation (Thuan and Pankaj, 1999). Figure 3.1show thread states diagram.

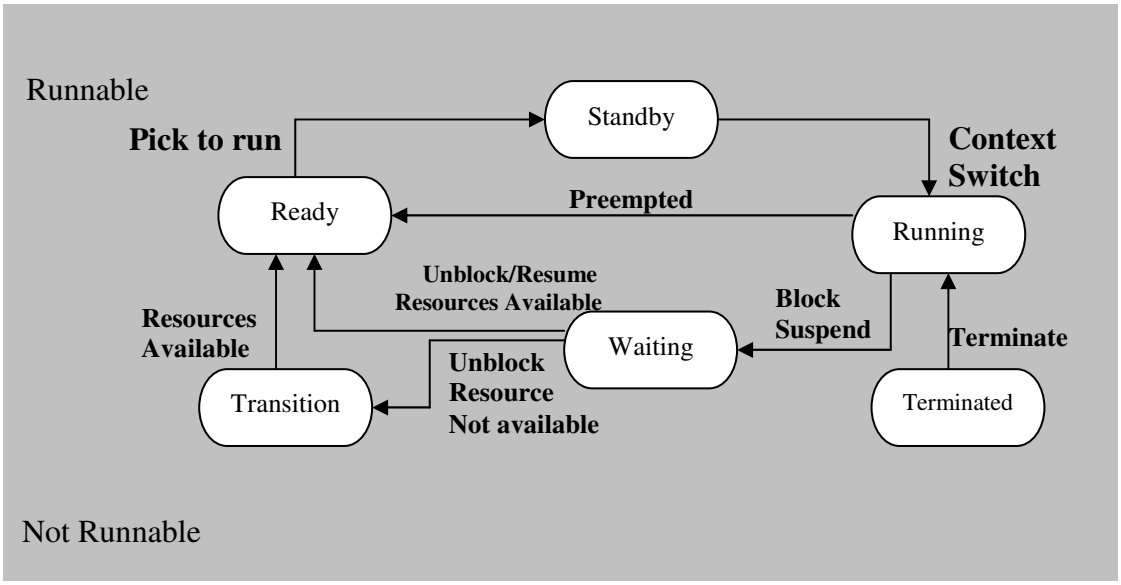


Figure 3.1 Possible thread states.

3.6 Parallel Threads Model

In the threads model of parallel programming, a single process can have multiple, concurrent execution paths (Akl, 1997). Perhaps the simplest analogy that can be used to describe threads is the concept of a single program that includes a number of subroutines shown in Figure 3.2

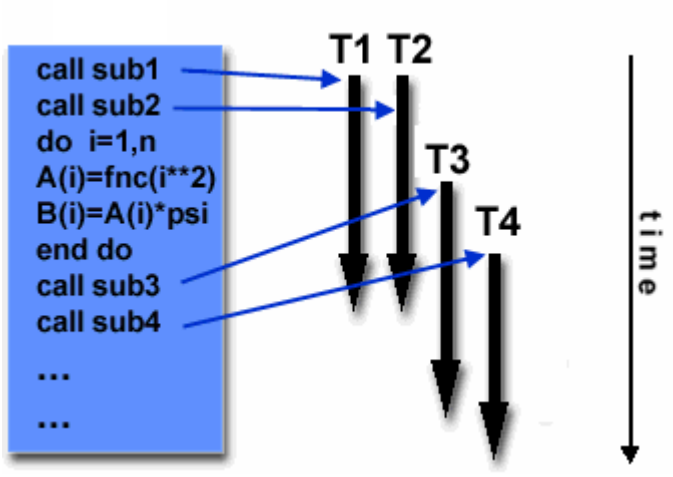


Figure 3.2 Single program illustrates parallel threads.

A thread's work may best be described as a subroutine within the main program. Any thread can execute any subroutine at the same time as other threads.

3.7 Thread Synchronization

One advantage of multithreaded programming is that you can speed up program by dividing it into independent threads; the threads can execute concurrently. For example, suppose we have to program a server application that accept requests from and provides service to client application. Using multithreaded programming, we can implement this application with a dispatcher thread that listens to client requests, and then create a separate thread to handle each such request concurrently. The problem with this strategy, however, is that we must properly synchronize the several server threads if they access shared data, in order to ensure that updates do not interfere with one another. In other words, one thread may need to wait for another to finish before it can proceed (Hum et al., 1994).

3.8 Threads in Client-Server Model

Distributed computing is a popular and important modern programming paradigm. Each of the computers participating in a distributed application runs a part of the application. Each part of distributed objects participates in a client-server relationship (Minoli and Schmidt, 1997).

The client-server model is a popular choice for migration from a centralized mainframe computer to a distributed network of workstations and personal computers. An instance of this model has to software computers, client and server who are independent programs (often running on separate machine) that communicate with each other using a predefined interface. A single server program services one or more client programs. In a large distributed computing environment with many clients, server processes can be replicated to share the load and improve response time. Figure 3.3 shows the high-level-client-server architecture of a distributed application.

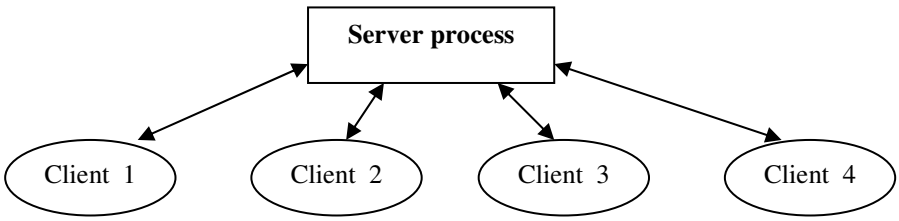


Figure 3.3 Client-server architecture of distributed application.

One reason for the client-server model's popularity is that it provides a natural and effective way to modularize an application. For example, instead of running many instances of large, interactive program, we can implement the core of the application as a server program, and implement the user interface as a client program. (Welch and Attiya, 1998).

Both clients and servers can benefit greatly by using threads. The minimal architecture of a server process may include a thread to receive and dispatch client requests, a thread to handle signals and exceptions, and one or more threads to carry out computation for each client request. On the client side, threads can be very effectively used to improve the responsiveness of application, with dedicated threads handing the user interface, while the others pre-fetch load, and compute data in the background. Figure 3.4 shows typical threads in a distributed application.

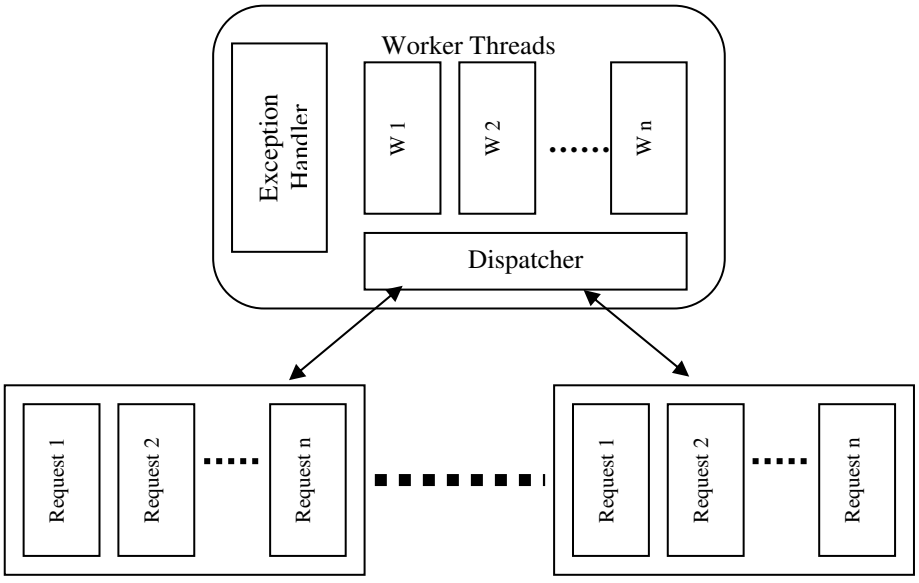


Figure 3.4 Multithreaded client-server architecture.

4. SIMULATION AND IMPLEMENTATION OF THE SYSTEM

This chapter describes the model for parallel processing for large satellite images via threads on a LAN. The implementations focus on filtering, and analyzing histogram data information. The distributed parallel system was built based on threads around the client-server-computing model across LAN.

Images are loaded to the server which they are partitioned to sub-images on the server, and then sent to client for processing. Then every processed sub-image is sent back to the server. Image processing application will be very important when the size of the image is very large. This is especially for satellite images. In this domain, the size of image can be more than 5000x5000 pixels some times. The required computational power is very high and the time needed to process images on a single CPU is very important.

4.1 Basics of Satellite Image Data Samples

Satellite images are really something different: They are unique and immensely powerful in conveying information about the environment. They are powerful not in themselves but in the way that they empower every one of us who has the opportunity to use them. Satellite images are larger extent, fewer details (comparing to air photography), suited for homogenous areas, and getting a survey.

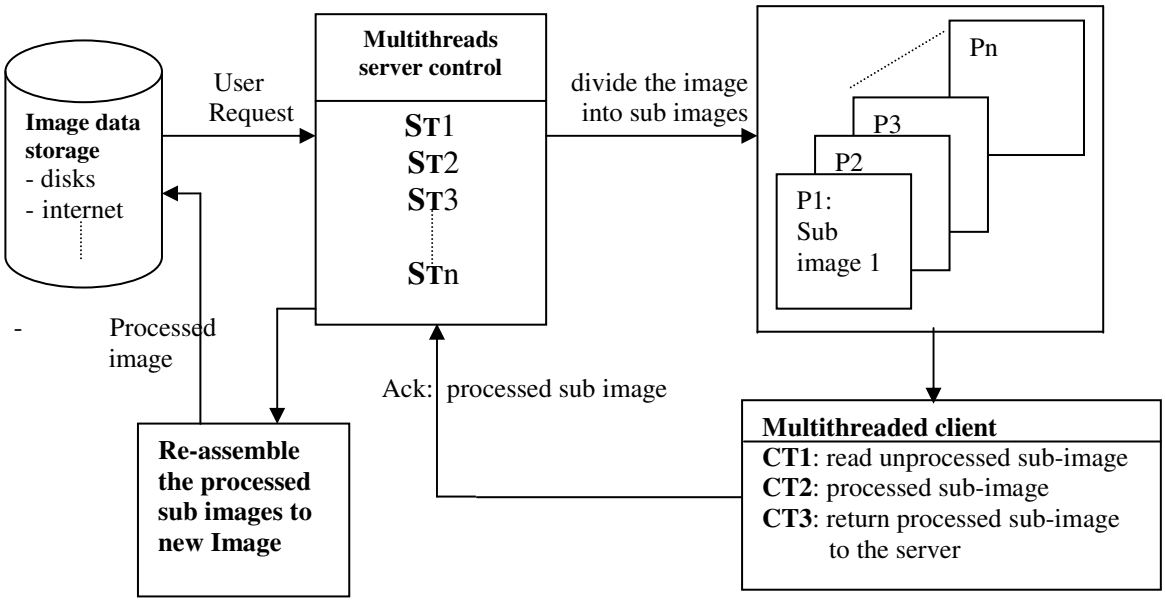
The data samples that we applied in the proposed system are concerned with (AVHRR) Sensors, which has five channels that collect measurements in different wavelengths. The first two channels collect visible and near-infrared measurements and the last three collect thermal measurements in different spectral resolutions. Where the

pixel sizes is 30 meter. Most of these samples are JPG, JPEG format. Its dimensions are ranged between 890x1120pixel and 3120x2990 pixel. Every image is differ from other by its represented information, where, some of these has a homogeneous levels of pixel intensities and digital values (like a water section), where, all pixels in this area are nearly equivalent in their features. And the others has a heterogeneous levels of pixel intensities and digital values (like urban section), where, all pixels in this area are nearly not equivalent in their features.

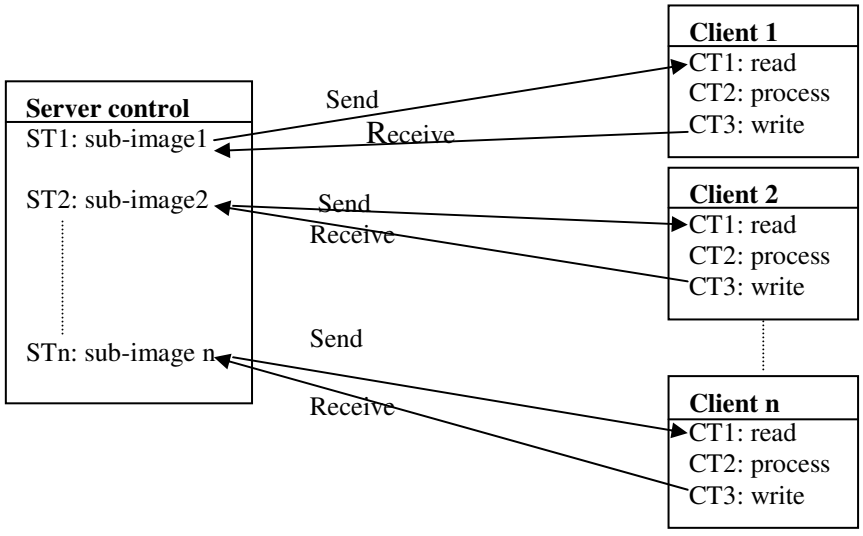
Some of collected data samples (satellite images) are obtained from Jordanian GIS center, like Amman city, Jordan valley, Rum mountains, and some internet image, like Mississippi river, little Colorado river, and others.

4.2 General description of the Proposed Approach

The work presented here concerns with a parallel processing of some applications of image processing, like sharpening filter and normalized image histogram that are described later. It involved of using multithreaded around client-server model. The data flow diagram and transmission is shown in Figure 4.1. Images were partitioned by the server threads (ST) and sent to Clients threads (CT), the CTs process and send back the processed parts to the server. The later resample the parts to new processed image and restores it.



(a) Data diagram



(b) Data transmission

Figure 4.1 Data diagram and transmission for the system

The behavior of the system and the steps passed on it as follows:

Input: unprocessed image (2D array image value)

Step1: load the unprocessed image from storage media to multithread server control.

Step2: apply the domain decomposition technique to divide the image into number of sub-images (partitions), P(n,m), where n,m are number of pixel.

Step3: create N threads (ST), where 3N threads running in parallel (CT) as:

N threads reading the N unprocessed sub image parts.

N threads processing N sub images partition.

N threads written back N processed sub images parts to new image (Processed image).

Step4: apply filtering and image histogram process to each sub-image part in each client.

Step5: resample the processed sub-image partitions into new image (Processed image) in the server.

Output: processed image (2D array image value).

4.3 Mathematical Model and Implementation of the System

4.3.1 Mathematical Model

Assume there is an unprocessed Image **G**, and a server **S,** contains **N** of threads **ST**, that defined as **ST1**, **ST2**, ---, **STn**. It's communicate with **N** of Clients **Cs**, each one contains 3 threads **CT**, that defined as **CT1**, **CT2**, and **CT3**.

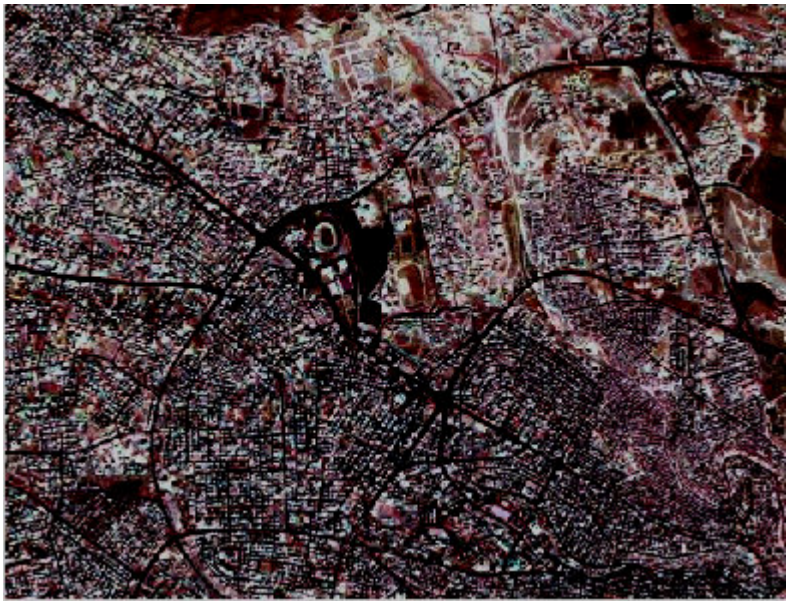
The **S** decomposed **G** into **N** sub-images (partitions): **P1**, **P2**, ----, **Pn,** then it sends each **Pi** to **Ci** (i = 1,….,n). For example, **ST1** sends **P1** client **CT1, CT2** performs the computations, and **CT3** send back the results.

The **S** will receive the processed partitions **Pi**, (i = 1,….,n) from all clients **C**, and resample it to get new Image (Processed image) .

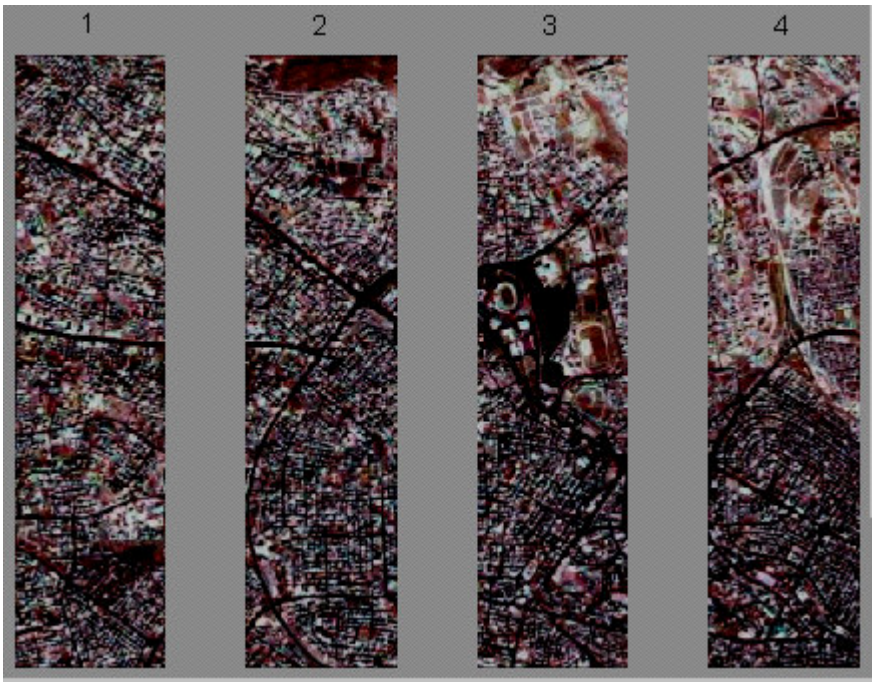
4.3.2 Methodology of the System

The target hardware consists of a heterogeneous collection of distributed workstations, and a server on a LAN. The multithreaded architecture was applied to get a high performance. First, the whole requested image is loaded to the server control on a LAN from storage device which may be a disk or from the internet or other devices.

The server starts by decomposing the data domain to sub-images according to number of clients (workstation connected to the server). The images are partitioned into vertical Blocks. In fact, this technique was chosen for some reasons. The image dimensions (width, height) are not equivalent; it is easy to construct the check board in parallel, provides greatest flexibility in parallel image processing, and applicable for writing the source code programs. The calculations regarded to partitioning are independent. This avoids the conflicts between blocks, in order to avoid the false sharing. Figure 4.2 shows an example of domain decomposition technique for Amman city image.



(a) Original image for Amman city



(b) Image decomposition of the original image due to four threads

Figure 4.2 a) the image before decomposition b) the image after decomposition

When the image was partitioned according to the number of threads on the server, each server thread (St) will deal and communicate with one client on a LAN, by sending its sub-image to the client who contains three threads. Here, the benefit of multithreaded will arise. One thread reads the sub-image, a second one does the calculation or processing, and a third one writes back the processed sub-image to the server.

We applied some of image processing operation for each sub-image in each client on a LAN, like sharpening filter that accentuates the contrast and the high frequency component of the image, and the normalized histogram (brightness histogram). Which is a graphic plot of pixel values within the image that shows the distribution of brightness level by representing pixel intensities (Sanchez and Canton, 1999). In fact, it's not necessary to apply these image process operations, or just only satellite images on the system, so, we can apply any other operations of processing. While the objectives of the research is to find a new technique to process any kind of image processing operations effectively.

When all distributed sub-images are back to the server, the server re-assembles and stores it in storage device as show in Figure 4.1. An example of decomposing of input image (unprocessed) with size 220x200 for fragment of Amman City. Some image processing operations, and the output image (processed) are shown in Figure 4.3.

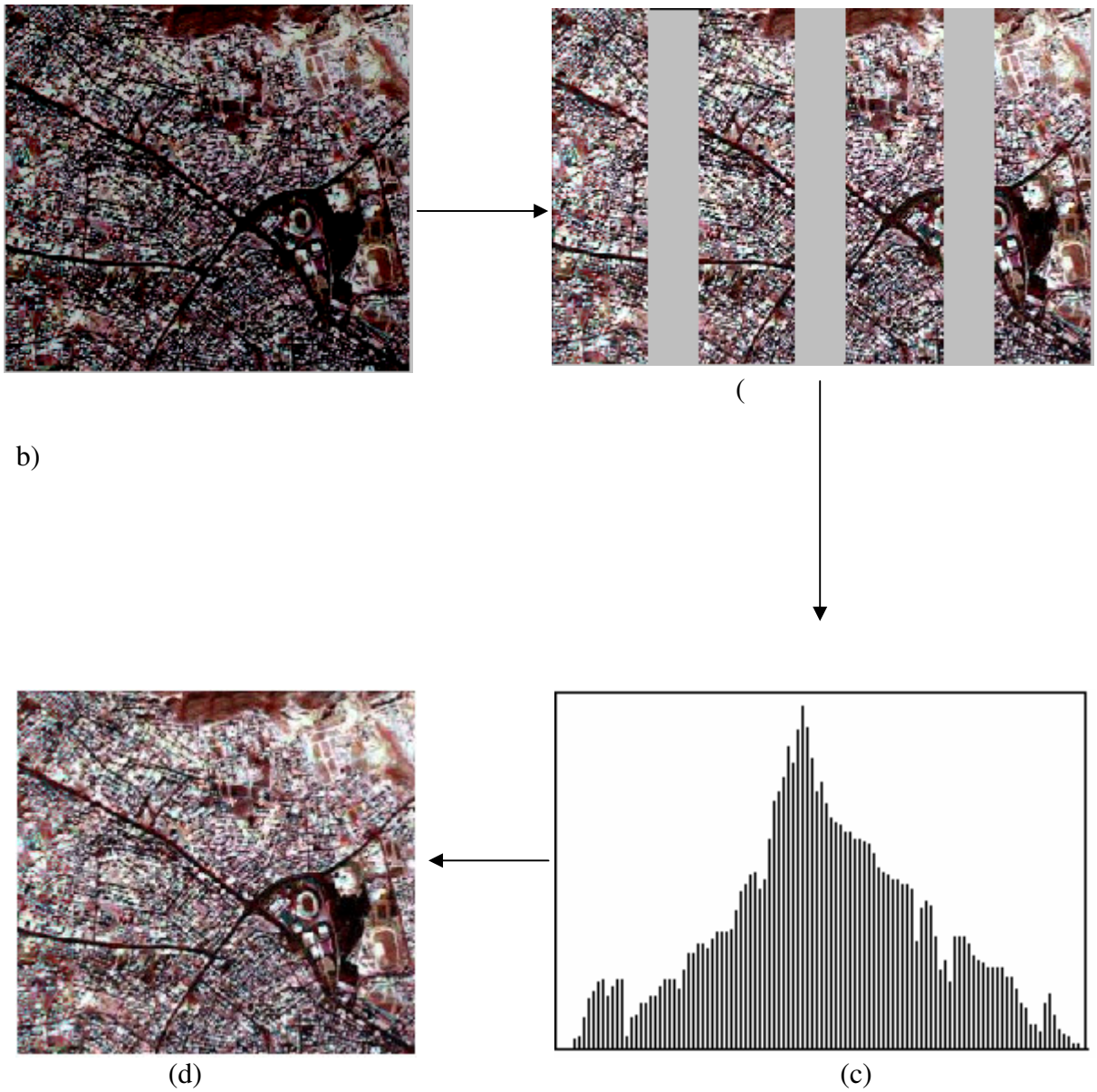


Figure 4.3 The processing steps: a) the input image b) image partitions after filtering c) histogram for the image after resample d) the output and processed image

4.3.3 Multithreaded Parallel Processing Model

There are number of threads created on the server and three threads included in each client: one for reading the sub-image, one for processing, and one for writing back to the server. This is possible by taking advantage of the multithreading features of current operating system. One thread is created for one of these three tasks, so, it possible to read the next sub-image while the current sub-image is processed and the previous one is written back to the sever.

It possible to distribute the workload on different processors by doing N times these three threads in parallel. This means that 3N threads run in parallel in an application. N threads can read the sub-image, N thread process the sub-images, and N threads write back the sub-image.

The pseudo code below shows the operations applied in both, the server and each client

```
SERVER_THREADS(Num_of_Thread, Client_connection, Creat_Thread[])
Begin
        In_image =  im_read(size,format)
        For I =  1 to Num_of_Thread  do
        Begin
                Server_decom_image(In_image,Num_of_block,size)
                Creat _Thread[I] = new thread
        If  (I = Num_of_Thread) then
         For J = 1 to num_of_block  do
        Begin
                While client = accept(server,socketadd)
```

```
                begin
                  Client= clientssocket[J]
                  Send_block(client,type,source,massage)
                  J = J + 1
                End
        End
                Else
              I = I +1
        End
End.

Server_res_image(Out_image,Num_of_block,bufer)
Begin
    For k = 1 to Num_of_block
    Beign
      Bufer = Rec_block(client,type,source,massage)
            If  (k = Num_of_block) then
              Out_image = writ_image(bufer)
            Else
                K = k +1
        End
End.

CLIENT_THREADS(Num_of_thread = 3, clientconnection,creat_Thread[])
Begin
         For I = 1 to Num_of_Thread
            Begin
              Creat_thread[I] = new thread
```

```
I= I +1
While  client(true)
Begin
  Thread[1].start = Blo_read(sblock,type,sourc,massage)
        Thread[1].teminate
  Thread[2].process = get_hpassfilt(sblock,matrix)
                      Get_normhistg(sblock,color,matrix)
        Thread[2].terminate
  Thread[3].send = Blo-send(sblock,bufer,type,source,masasage)
        Thread[3].terminate
Closesoket(client)
  End
End
End.
```

4.4 System overview

4.4.1 Hardware

The proposed parallel system was applied over a LAN in star topology form. It is a multi computer architecture, which can be used for parallel computation. It is a system built using commodity hardware components such as PC's connected via Ethernet or other network. The proposed system consists of one server node and number of client nodes. All nodes are connected by 24-port 100 Mbps switch Hub. Figure 4.4 shows the architecture of the system.

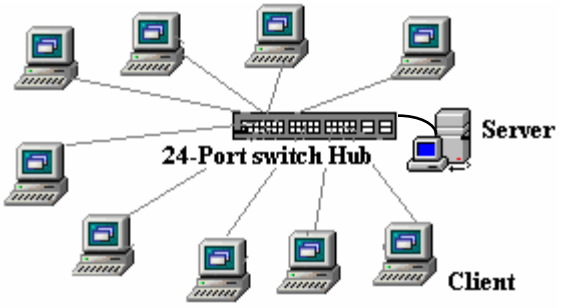


Figure 4.4 System architecture

For the parallel computing system, the server node has Intel Pentium II 366 MHz processor and 128 Mbytes of shared local memory, and each client node has Intel Pentium I 233 MHz processor.

4.4.2 Software

C++ was chosen to be the software that we write the parallel system implementation. Because, it support a more flexible using of threads features, and also, it is flexible to demonstrate the communication over the parallel system.

4.5 Calculations and Measurements

This section describes how the whole computations are calculated in the system. First, the start time was initiated to zero before applying the system. When the system started, the time (start time) for all threads on the server control was recorded, so when all clients sent back the processed sub-image we end the time, and then make the calculation. Figure 4.5 show a calculation model of the system.

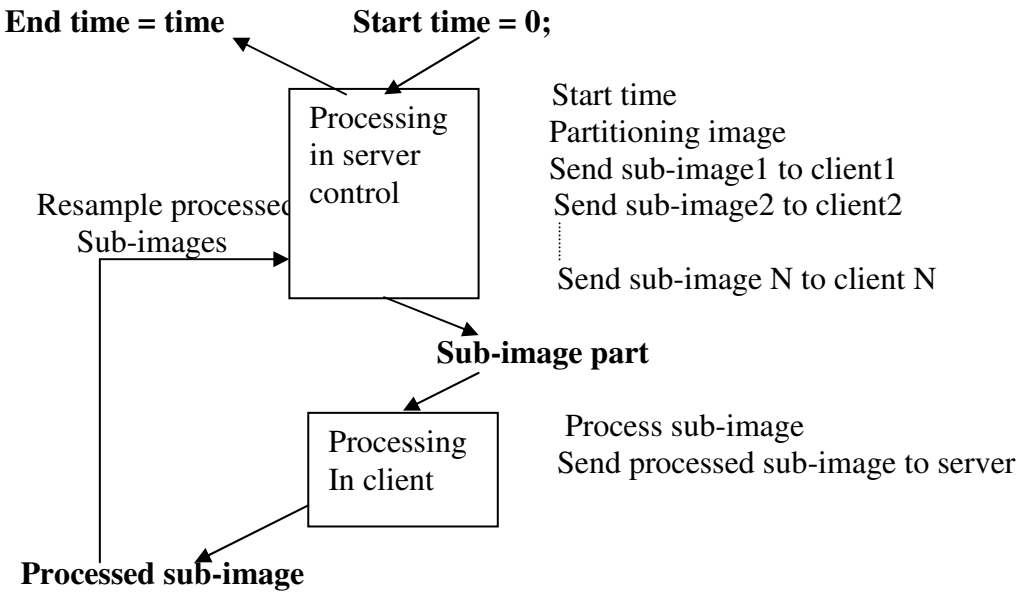


Figure 4.5 Calculation model of the system

The Tp is the sum of time to partition the images, sending the sub-images, processing sub-images, sending back the processed sub-images, and re-assemble the sub-images. The communication time is the time to send the sub-images and receive them back. The computation time is the time to decompose the data, process the sub-images, and combine the results.

4.6 Results

Many experiments and tests of different sample data were done. Every image is tested 15 times and the median of computation time was taken. The results of the experiments are shown in Tables 4.1 through 4.4 and Figures 4.6 through 4.11.

Table 4.1 Execution Time for Images of different size on different number of threads

	Execution Time in Seconds								
No. of Thread	**Image Size 200x220**	**Image Size 550x510**	**Image Size 890x780**	**Image Size 1190x1110**	**Image Size 1500x1480**	**Image Size 1725x1640**	**Image Size 2260x2210**	**Image Size 2690x2580**	**Image Size 3120x2990**
1 Serial	4.05	17.96	50.51	102.32	155.15	237.52	336.53	461.22	598.6
3	3.35	13.81	35.82	66.44	89.6	124.35	148.9	178.76	208.57
6	2.1	8.93	23.06	45.47	58.1	72.41	91.2	107.01	120.2
9	1.81	5.93	14.39	24.89	32.05	47.4	56.75	70.84	83.37
12	.99	4.09	10.76	20.26	27.65	38.3	47.26	54.71	64.29
16	**.69**	**2.99**	**7.86**	**14.98**	**21.69**	**29.87**	**38.22**	**44.89**	**49.11**
20	.82	3.4	8.61	16.6	23.88	33.1	44.12	51.35	59.81

Table 4.1shows the execution time of experiments that were tested on machines with 1, 3, 6, 9, 12, 16, and 20 threads. The images size where different and variable, the smallest image has 44000 pixels and the largest image has 9328800 pixels.

Table 4.2 Speedup for Images of Different size on different number of threads

	Speedup – S								
No. of Thread	**Image Size 200x220**	**Image Size 550x510**	**Image Size 890x780**	**Image Size 1190x1110**	**Image Size 1500x1480**	**Image Size 1725x1640**	**Image Size 2260x2210**	**Image Size 2690x2580**	**Image Size 3120x2990**
1 Serial	1	1	1	1	1	1	1	1	1
3	1.21	1.3	1.41	1.54	1.73	1.91	2.26	2.58	2.78
6	1.93	2.01	2.19	2.25	2.67	3.28	3.69	4.31	4.98
9	2.23	3.03	3.51	4.11	4.84	5.01	5.93	6.51	4.18
12	4.09	4.39	4.69	5.05	5.61	6.2	7.12	8.43	9.31
16	**5.86**	**6.01**	**6.42**	**6.83**	**7.15**	**7.95**	**8.8**	**10.37**	**12.18**
20	4.93	5.28	5.86	6.16	6.49	7.17	7.62	8.98	10.01

Table 4.2 shows the speedup (S) for images of different size, where the highest S is 12.18. This achieved when image size is 3120x2990 on 16 threads.

Table 4.3 Efficiency for Images of different size on different number of threads

	Efficiency								
No. of Thread	**Image Size 200x220**	**Image Size 550x510**	**Image Size 890x780**	**Image Size 1190x1110**	**Image Size 1500x1480**	**Image Size 1725x1640**	**Image Size 2260x2210**	**Image Size 2690x2580**	**Image Size 3120x2990**
3	0.40	0.43	0.47	0.51	0.57	0.63	0.75	0.86	0.95
6	0.32	0.33	0.36	0.37	0.44	0.55	0.60	0.71	0.83
9	0.52	0.34	0.38	0.45	0.53	0.56	0.65	0.72	0.80
12	0.34	0.36	0.39	0.42	0.46	0.52	0.59	0.70	0.77
16	0.37	0.38	0.40	0.43	0.44	0.50	0.55	0.64	0.76
20	0.24	0.26	0.29	0.30	0.32	0.36	0.38	0.44	0.50

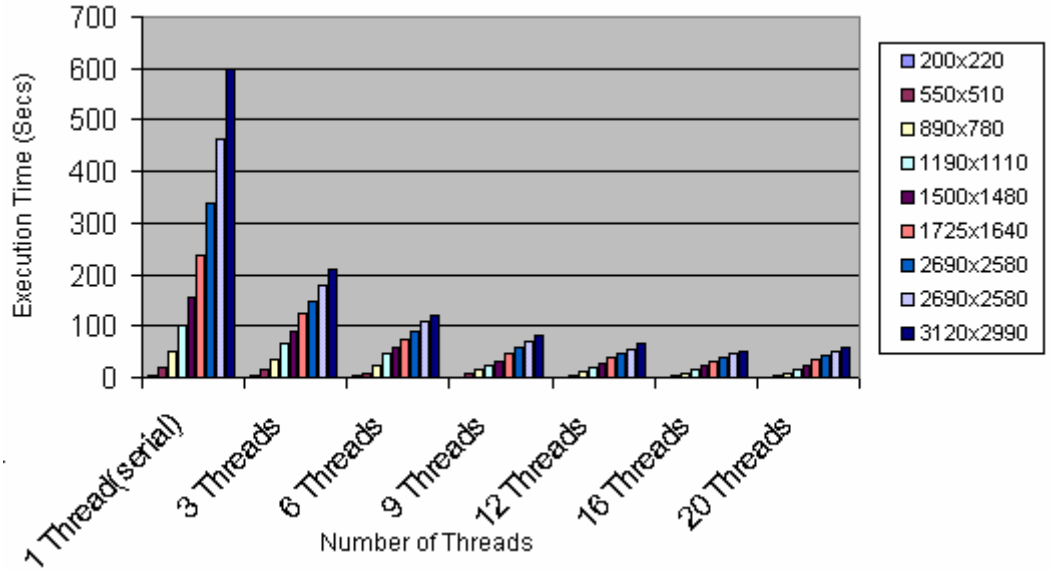
Table 4.3 shows the efficiency (E) for different image sizes, the highest value of E = 0.95. It was achieved when the size of the image was 3120x2990 and number of threads was three. The ratio of the image size to number of threads was scaled as log10 (3120x2990 / 3) = 6.49

Table 4.4 Ratio of image size to number of threads, scaled using logarithmic scale to base 10 .

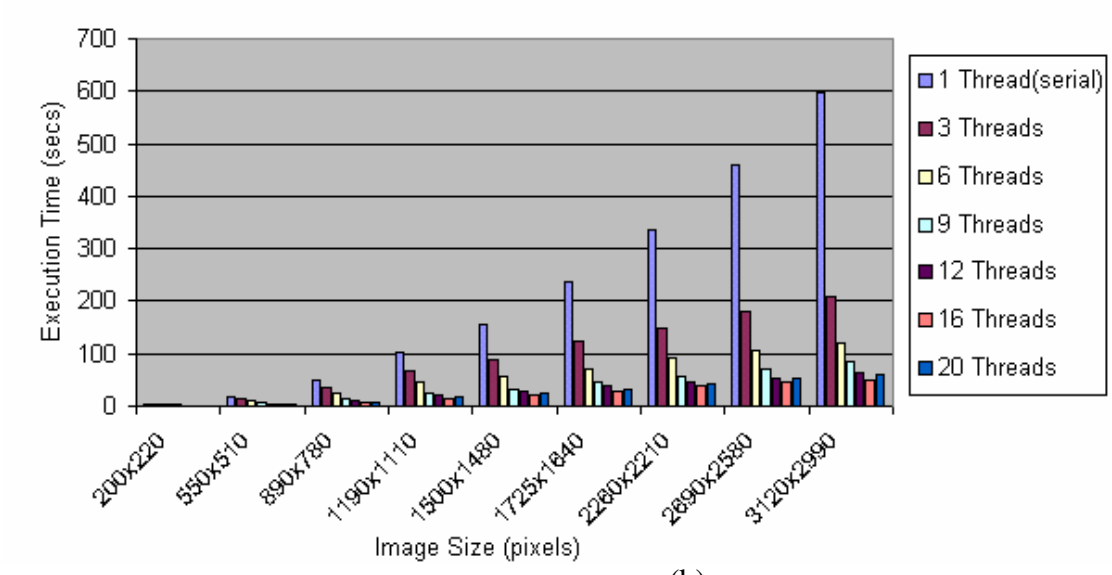
	Ratio of image size to number of threads								
No. of Thread	**Image Size 200x220**	**Image Size 550x510**	**Image Size 890x780**	**Image Size 1190x1110**	**Image Size 1500x1480**	**Image Size 1725x1640**	**Image Size 2260x2210**	**Image Size 2690x2580**	**Image Size 3120x2990**
3	4.16	4.97	5.36	5.46	5.86	5.97	6.22	6.36	6.49
6	3.86	4.66	5.06	5.34	5.56	5.67	5.92	6.06	6.19
9	3.68	4.49	4.88	5.16	5.39	5.49	5.74	5.88	6.01
12	3.56	4.36	4.76	5.04	5.26	5.37	5.61	5.76	5.89
16	3.43	4.24	4.63	4.91	5.14	5.24	5.49	5.63	5.76
20	3.34	4.14	4.54	4.81	5.04	5.15	5.39	5.54	5.66

Table 4.4 shows the ratio of different images size to different number of threads as

Log10 (image size/number of threads)



(a)



(b)

Figure 4.6 a) Execution time (Secs) versus different number of threads for several images with different size
b) Execution time (Secs) versus images size on different number of threads with different sizes.

Figure 4.6 show how the execution time decrease while number of thread increased.

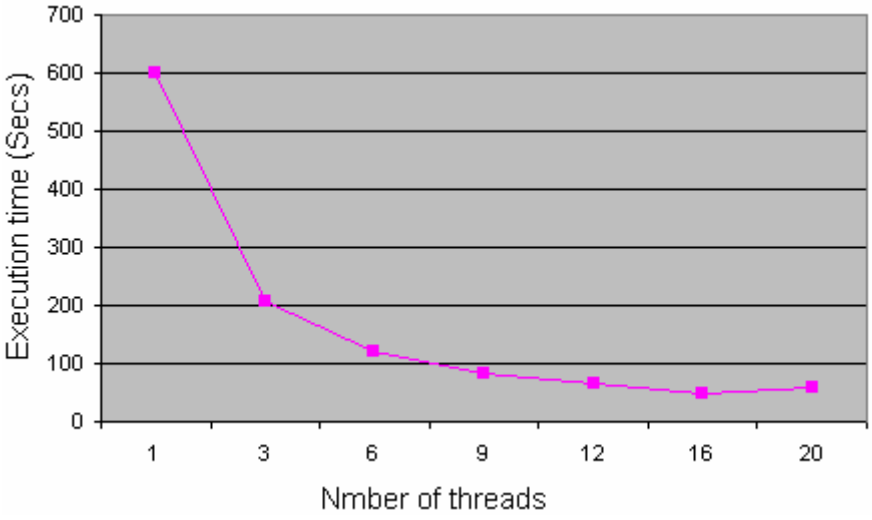


Figure 4.7 Execution time (Secs) versus different number of threads on image Size 3120x2990.

Figure 4.7 shows the reduction in execution time when there was increasing the number of thread (n) for a fixed size of an image. The relation close to

Tp $\propto$ constant / n x m.

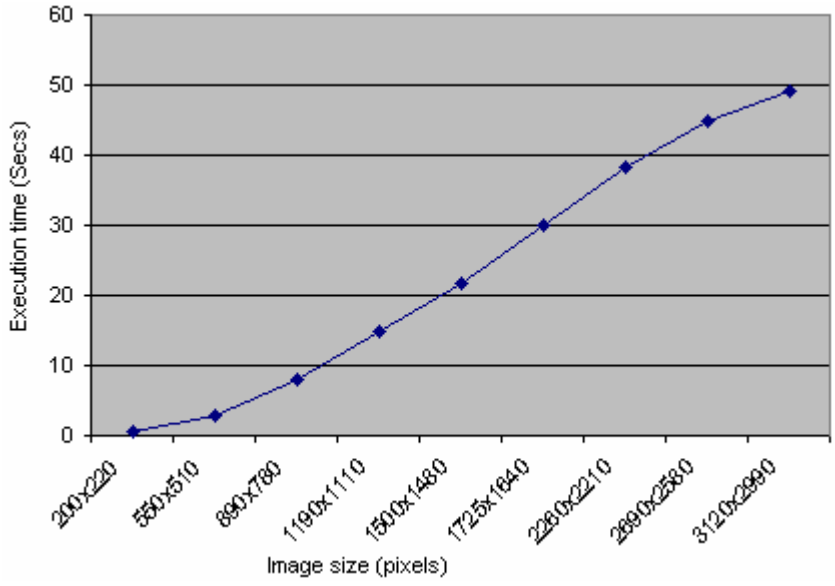
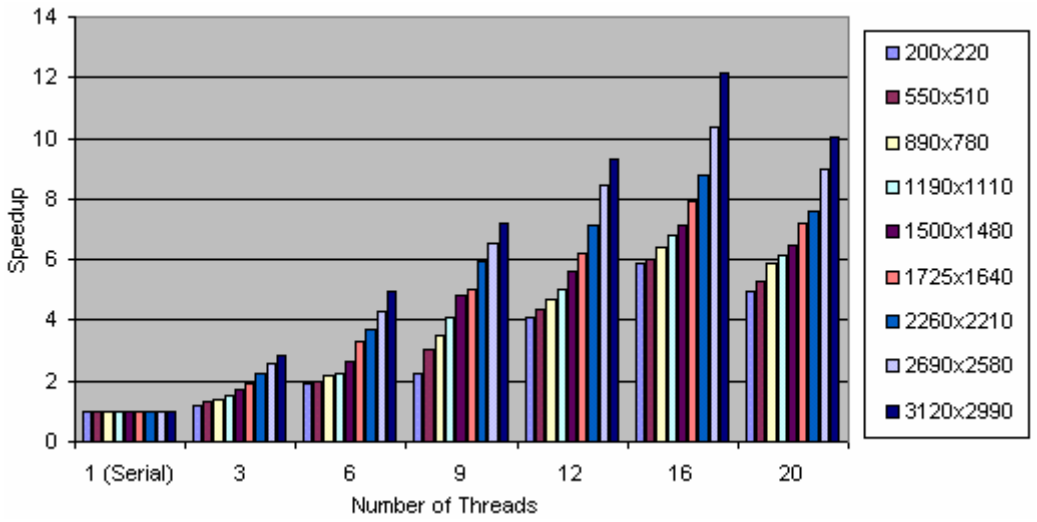
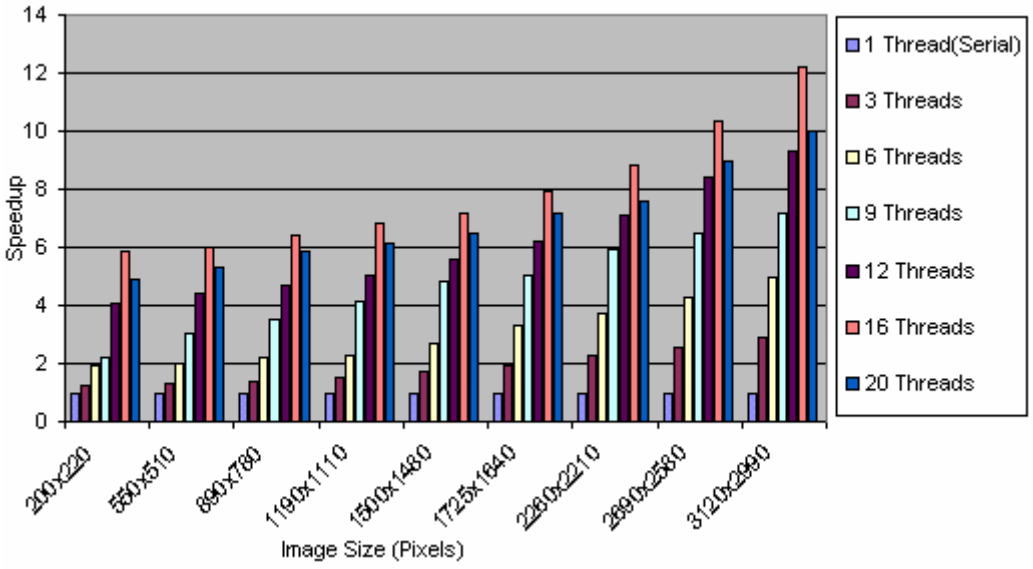


Figure 4.8 Execution time (Secs) versus different image size on 16 threads.

Figure 4.8 shows the relation of Tp with size of image when number of threads is fixed. The Tp $\propto$ constant x L, where L is the number of pixels (image size).



(a)



(b)

Figure 4.9 a) Speedup versus number of threads working on different images size
b) Speedup versus different images size on differnet number of threads

In Figure 4.9, the maximum speed up was achieved when the number of threads was 16, where Sp = 12.18.

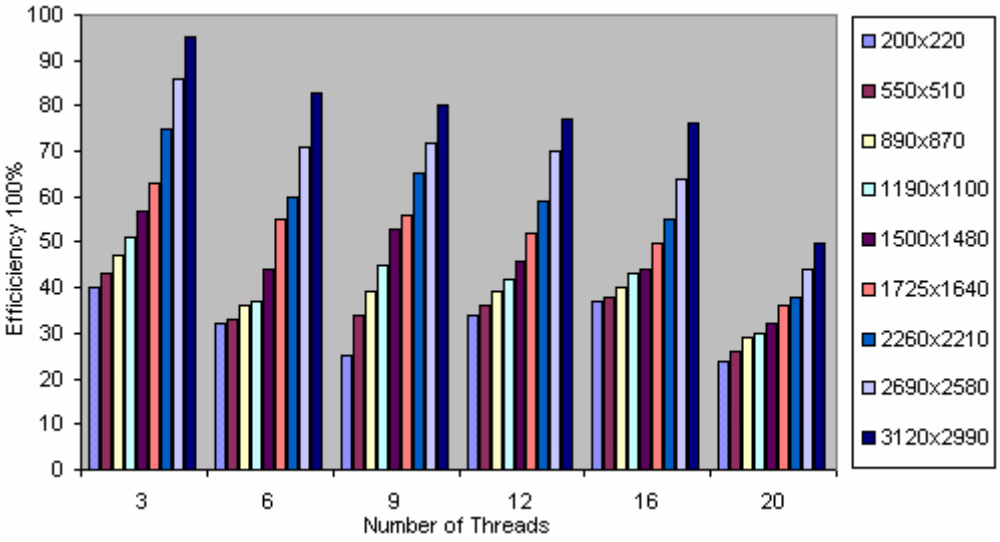
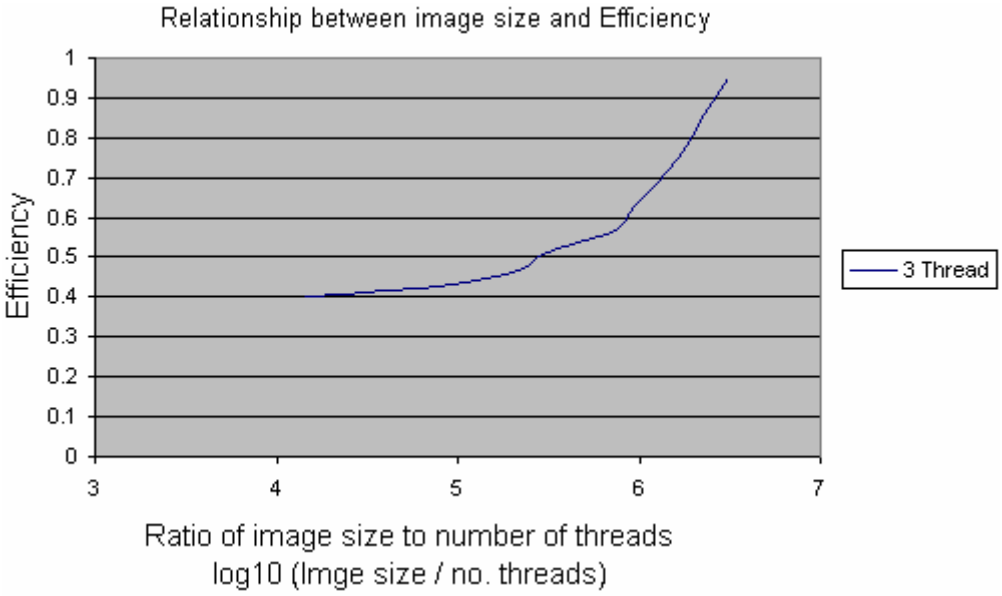
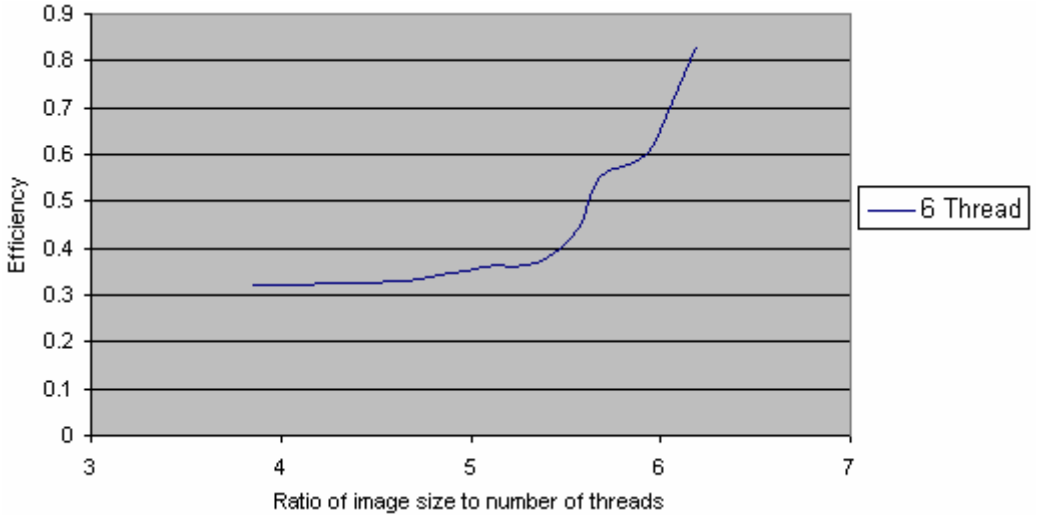


Figure 4.10 Efficiency 100% versus number of threads working on different.

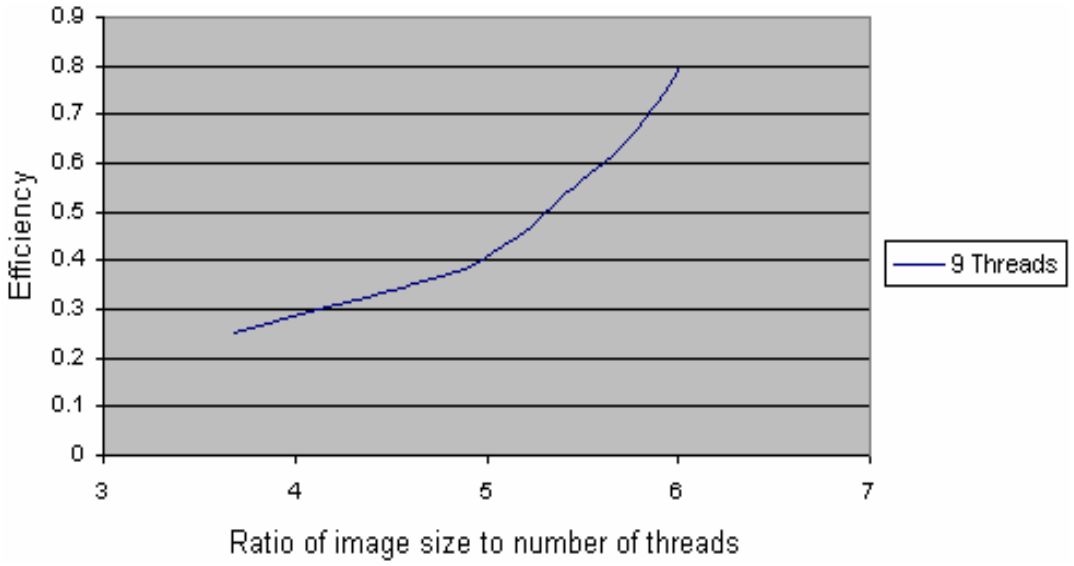
In Figure 4.10, the highest value of E was achieved when the number of threads was three, where E = 0.95 for images size 3120x2990.



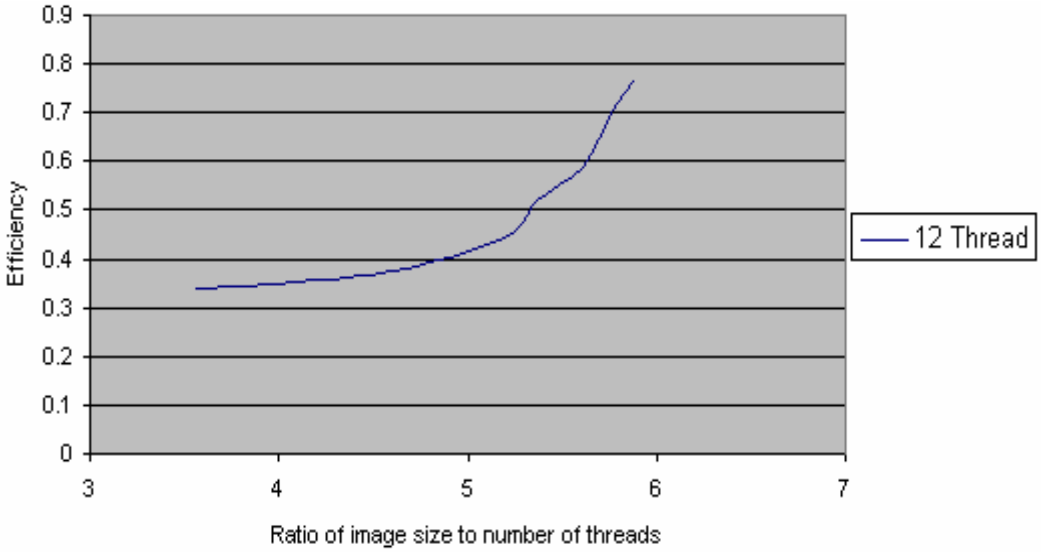
(a)



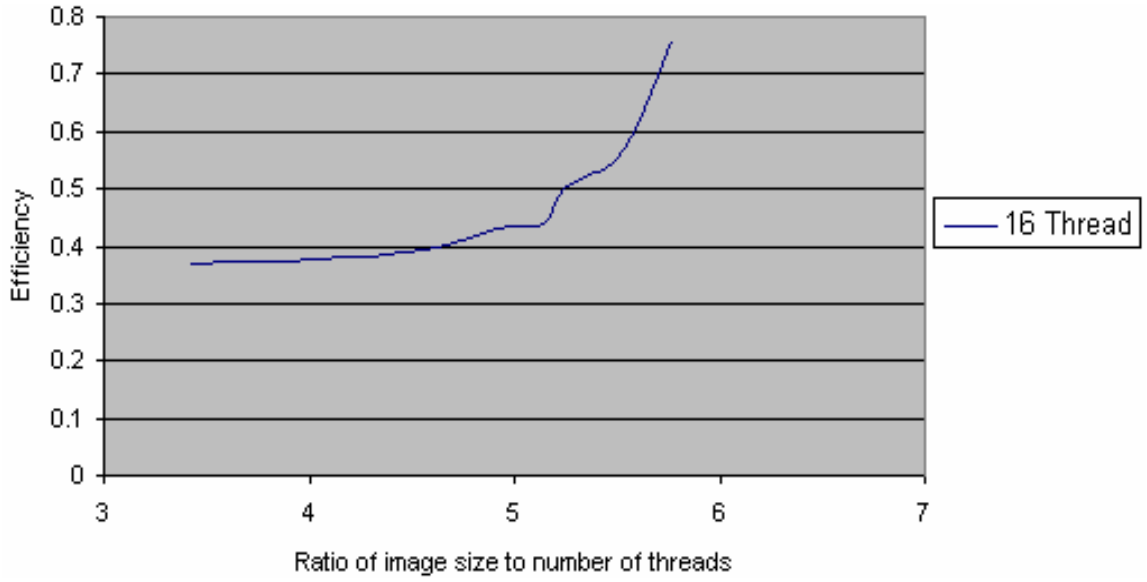
(b)



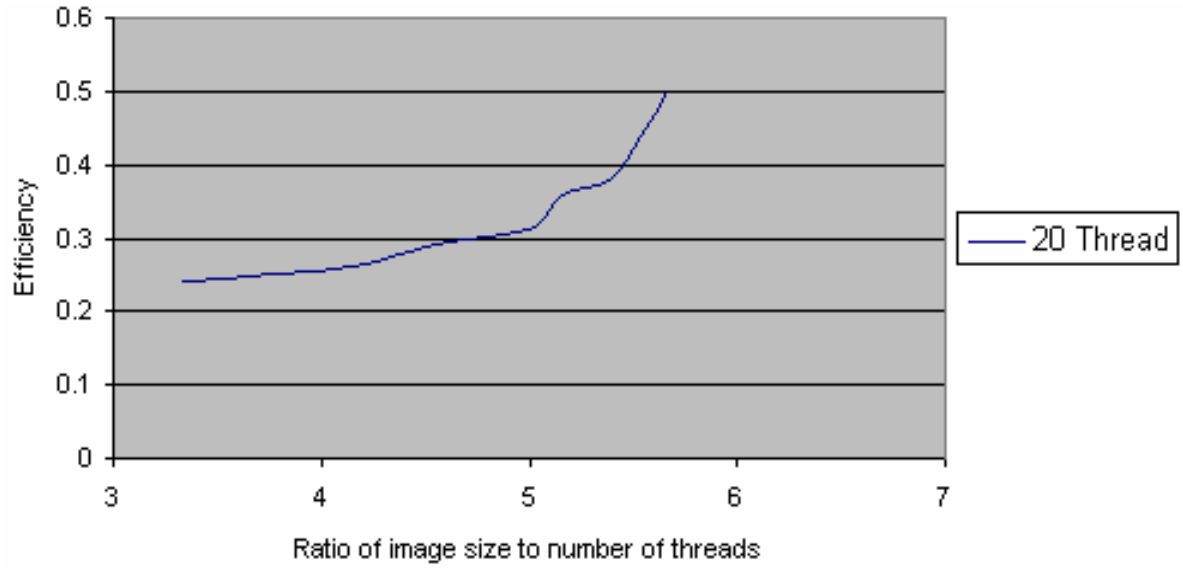
(c)



(d)



(e)



(f)

Figure 4.11 Ratio of different images size to different number of threads.
a) 3 threads b) 6 threads c) 9 threads d) 12 threads e) 16 threads f) 20 threads

Figure 4.11 shows that E is increasing as the ratio if image size to number of thread increase. Because the computation is dominant. This indication explains that the system will be more applicable to implement for large images.

4.7 Performance Analysis

As the results, that had shown in the Tables 4.1, 4.2 and the graphs in Figures 4.6 through 4.9. We see that 16 threads is best number that was taken less execution time for all different image sizes and the maximum speedup 12.18 for these different image sizes. But this not necessary to be the cost effective.

For the Efficiency that is shown in Table 4.3 and Figure 4.10, the highest value of E is achieved when image size =3120x2990 and N=3 threads, where the ratio of image size to number of threads was 3,109,600 pixel/thread, which expressed as log10(3120x2990 / 3) = 6.49 pixel/thread as shown in table 4.4. where E =0.95. this indication support the relation in equation 4.1

$E(n \times m, p) \propto n \times m/p$ ……..(4.1)

Where E is efficiency, n x m represent the problem size (image size), and p the number processors (in our case p represent number of threads), so when problem size increases, the efficiency will increase. Figure 4.11 explains the proof of this relation.

But for cost effective, we see that E is higher than 80% for treads: three to nine for image size 3120x2990. From this, we conclude that, the system with nine threads would be cost effective. This will be shown clearly, when it applied to larger satellite images (i.e. 4000x4000 and more) than we used.

From the performance analysis, we inferred that, the system to be applied for large image size would be more efficient and more applicable, where the computation takes the longest runtime.

The proposed system satisfied the result in execution time that is less than seven times in average comparing to other researches, where the most of these systems are depend on using multiprocessors, clustering, and PVM
(Hawich and James,1997),(Yang and Hung, 2000), (Gess et al, 1996).

5. CONCLUSION AND FUTURE WORK

5.1 Conclusions

An application of parallel and distributed computing for remote sensing data such as satellite image processing was explored. The splitting of large images (satellite images) into sub-images has been applied to the image processing using parallel processing via threads on a LAN. However, this technique is suitable in general and can be applied to many image-processing algorithms.

This thesis shows that threads are very portable for parallel applications or in distributed systems. In most cases, threads help speeding of the parallel implementations, by improving input/output operation and by distributing the workload on different processors that are available in the distributed system.

We have implemented a system for different large satellite images, and measured a good speedup performance for the parallel and distributed components on the system. The Execution time and speedup, for the tested samples, show that 16 threads were taken less execution time and maximum speedup of 12.18. But this is not necessary to be the cost effective. The efficiency indicates that, the nine threads are the best for cost-effective application on the proposed system, while E is higher than 80% for Threads: three up to nine for image size 3120x2990. Although, the highest value of E = 0.95, where the ratio of image size to number of threads was 3,109,600 pixel/thread, when image size is 3120x2990 and number of thread is three. The Results are encouraging compromising to other studies.

5.2 Future Work

Indeed, image processing and parallel processing have a very wide application in most fields, such as, remote sensed data like satellite images, where we cannot collect the most of theses application in our system. The system can be expanded for the remote sensed data processing applications, such as: image classification, image correlation and geo-rectification. The approach can be modified for distributed parallel image processing system using the advantage of multithreaded for covering an important segmentation and visualization algorithms for 3-Dimensional (3-D) remote sensed data, based on a heterogeneous Asynchronous transfer mode (ATM) networks.

REFERENCES

Accuweather organization. 2000. Frequently Asked Questions about Satellite Images. Available at http://www.accuweather.com/iwxpage/adc/help/pr_satellite.htm

Akl, Selim. G. 1997. *Parallel Computation: Models and Methods.* 1st. edition. Prentice-Hall, Inc. USA.

Anderson, Jean. T., and Stonbraker, Michael. 2000. SEQUOIA 2000 METADATA SCHEMA FOR SATELLITE IMAGES. EECS Department. University of California. Berkely. California. Available at. citeseer.nj.nec.com

Baxes, Gregory. A. 1994. *Digital Image Processing principle and applications.* 1st. edition. John wily & sons. Inc. USA.

Behrooz, Parhami. 1999. *Introduction to Parallel Processing Algorithms and Architectures*. Plenum Press. New York.

Bergman,Larry., Stolorz, Paul., Blom, Ron., Stanfill,Dan., Kwan, Bruce., Crippen, Bob., Lyster, Peter., and Li, Peggy. 1998. "CALCRUST: InteractiveThree-Dimensional Rendering of Multiple Earth- Science Datasets". Jet Propulsion Laboratory. Dave Okaya. University of Southern California. Available at www.cacr.caltech.edu

Canada Avalanche Association. 2000. Satellite images: A simple Tutorial. Mountain Guide. Cyril Shokoples. Available at. http://www.compusmart.ab.ca/resqdyn/index.htm

Carriero, Nicholas., and Gelernter, David. 1990. *How to write Parallel Programs*.1st. edition. Cambridge university press. Cambridge, London.

Chaver, D., Prieto, M., Piůel, L., and Tirado, F. 2002. "Parallel Wavelet Transform for Large Scale ImageProcessing". Departamento de Arquitectura de Computadores Automtica Facultad de C.C. Fsicas. Universidad Complutense.Ciudad Universitaria s/n 28040 Madrid. Available at. www.acya.ucm.es

Commonwealth of Australia. 2002. About the Satellite Images. Commonwealth Bureau of Meteorology. Available at. www.bom.gov.au

Digital image processing. 2000. Available at. www.csc.fi

Escobar, F., Hunter, G., and Bishop, I. 2000. Introduction to GIS. Department of Geomatics. University of Malbourne. Available at www.sli.unimelb.edu.au

Fountain, T. J. 1994. *Parallel Computing Principle and Practice. 1st. edition.* Cambridge university press. Great Britain at the university press. Cambridge, London.

Fukoda, Masayuki., Watanabe, Satoshi., Tanimoto, Toshiya., and Miyahara, Masakatsu. 2000. "High Performance Processing System with General-purpose Computer". NEC Corporation. 4035 Ikebe-cho. Yokohama. Japan.
Available at www.gsdd.spc.hy.nec.co.jp

Giess, Christoph., Mayer, Achim., Evers, Harald., and Meinzer, Hans-Peter.1996. "Medical Image Processing and Visualization on Heterogeneous Clusters of Symmetric Multiprocessors Using MPI and POSIXthreads". Deutche Krebsforschungszentrum. Dept. medical and Biological Informatics. Heidelberg. German. Available at. ipdps.eece.unm.edu

Gupta, Ravi. P.1991. *Remote Sensing Geology. 1st. edition. Springer-verlag berlin. Heidelberg. German.*

Harris, Ray.1987. *Satellite Remote Sensing.* 1st. edition. Routledge & Kegan paul Ltd. London & NewYork.

Hawich, K. A., and James, H. A. 1997."Distributed High-Performance Computation for Remote Sensing". Department of computer science. University of Adelaide. SA 5005. Australia. Available at. www.cs.adelaide.edu.au

Hawick, K.A., Coddington, P.D., and James, H.A. 2002. "Distributed Frameworks and Parallel Algorithms forProcessing Large-Scale Geographic Data". Department of Computer Science.School of Informatics. University of Adelaide.University of Wales,Bangor,SA 5005,Australia.North Wales,LL57 1UT,UK. Available at. www.pangor.ac.uk and www.cs.adelaide.edu.au

Hord, R. Michael. 1999. *Understanding Parallel supercomputing.* 1st. edition. IEEE, Inc. USA.

Hum, Herbert. H. J., Theobald, Kevin. B., and Geo, Guang. R. 1994. Building Multithreaded architecture with off-the-self microprocessors. School of computer science. Canada. Available at. www.ftp.capsl.udel.edu

Kasahara, Hironori., Obata, Motoki., and Ishizaka, Kazuhisa. 2001. "Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP". Waseda University.3-4-1 Ohkubo. Shinjuku-ku. Tokyo. 169-8555. Japan. Available at. www.scar.elec.waseda.ac.jp

Kingston Center For GIS. 2000. Introduction to GIS and Geospatial Data. Kingston Center for GIS. Kingston University. Kingston upon Thames. KT1 2EE. UK. Available at www.Kingston.ac.uk

Kumar, Vapin., Grama, Ananth., Gupta, Anshul., and Karypis, George. 1994. *Introduction to Parallel Computing, Design Analysis of Algorithms.* 1st. edition. Benjamin/Cummings Publishing Company, Inc. USA.

Minoli, Dan. And Schmidt, Andrew. 1997. *Client/Server Over ATM.* 1st. edition. Manning Publication. USA.

Pham, Thuan. Q., Garg, Pankaj. K. 1999. *Multithreaded Programing with Win32. 1st . edition.* prince hall,PTR, Uper Saddle River, New Jersey 07458.

Quinn, Michael. J. 1994. *Parallel Computing: Theory and Practice.* 2nd. edition. McGraw-Hill,Inc. Singapore.

Rallings, Philip. J., Ware, J. Andrew, and Kidner, David. B. 1998. Parallel Distributed Processing for Digital Terrain Analysis. Division of Mathematics & Computing, University of Glamorgan, Pontypridd, Rhondda Cynon Taff WALES CF37 IDL. Available at. www.geocomputation.org

Richason, Benjamin. F. 1983. *Introduction to Remote Sensing of The Environment.* 2nd. edition. The National Council for Geographic education. USA.

Sabins, Jr. F. 1987. *Remote sensing: principles and interpretation*. 1st. edition. New York: W.H. Freeman.

Saltz, Joel., Acharya, Anurag., Sussman, Alan., Hollingsworth, Jeff., and Beynon, Michael. 1999. "Tuning the I/O Performance of Earth Science Applications". University of Maryland and Center of Excellent in Space Data & Information. Science (CESDIS), NASA Goddard. Available at www.cs.umd.edu

Sanchez, Julio., and Canton, Maria. P. 1999. *Space Image Processing*. 1st. Edition. CRC Press LLC. NewYork.

Spencer, John. 2000. Image Enhancement . available at. www.cpc.nuc.edu

Taylor, Stephen. C., Armour, Bernard., Hughes, William. H., Kult, Andrew., and Nizman, Chris.1999. "Operational interferometric SAR data processing for RADARSAT Using a distributed computing environment". Atlantis Scientific Inc. available at. www.atlsci.com.

Wagner, Tom., and towsley, Don. 2000.Getting Started With POSIX Threads . Department of Computer Sience, University of Massachusetts at Amherst. Available at dis.cs.umass.edu

Welch, Jennifer., and Attiya, Hagit. 1998. *Distributed Computing: Fundamentals, Simulations and advanced Topics. 1st. edition. McCraw-Hill international (UK) limited.* Great Britain at the university press. Cambridge, London.

Yang, Chao-tung., and Hung, Chi-Chu. 2000. "Parallel Computing in Remote Sensing data Processing". Ground System Section. National Space program office. Hsinchu.Taiwan 300, R.O.C. available at. www.nspo.gov.tw

Zhang, Yunqing. P., Fracassi, John. F., Wiggins, John. E., Glenn, Scott. M., and Grassle, J.F.1999. RODAN: RUTGERS OCEAN DATA ACCESS NETWORK POWERED BY JAVA TECHNOLOGIES. Rutgers University. New Brunswick, NewJersey. Available at. marine.rutgers.edu

Appendix 1

Abbreviations and acronyms

Abbreviation	Acronym
AMI	Active Microwave Instrument, on ERS-1.
APT	Automatic Picture Transmission.
ATS	Application Technology Satellite.
AVHRR	Advanced Very High Resolution Radiometer, on Noaa satellite.
CCT	Computer-Compatible Tape.
CEOS	Committee for Earth Observation Satellite.
CGMS	Coordination of Geostationary Meteorological Satellite committee.
CRC	Color Ratio Composite.
EOS	Earth Observing System.
Eosat	Earth Observation Satellite Corporation.
Essa	ESSA satellite.
ETM	Enhanced Thematic Mapper.
GEO	Geostationary Earth Orbit.
GMS	Geostationary Meteorological Satallite.
GOES	Geostationary Operational Environmental Satellite.
HRPT	High Resolution Picture Transmission.
HRV	High Resolution Visible, sensor on the SPOT satellite.
IPOMS	International Polar Orbiting Meteorological Satellite Committee.
IRS-1	Indian remote sensing satellite.

J-ERS-1	Japanese environmental remote sensing satellite.
LEO	Low Earth Orbit.
LFC	Large Format Camera, on the space shuttle.
MSS	Multispectral Scanner, on Landsats 1-5.
NASA	National Aeronautics and Space Administration.
Noaa	NOAA satellite.
NRSC	National Remote Sensing Center.
SAR	Synthetic Aperture Radar.
SIR	Shuttle Imaging Radar.
SMS	Synchronous Meteorological Satellite.
SPOT	Satellite Probatoire d'Observation de la Terre.
Tiros	Television and Infrared Observation Satellite.
TM	Thematic Mapper, on Landsata 4 and 5.
VISSR	Visible and Infrared Spin-Scan Radiometer, on SMS satellite

Appendix 2

Summary of thread architectures for various operating system

System	Thread package architecture
Digital Equipment Corporation, CMA threads	User-level
Distributed Computing Environment (DCE)	User-level
Xerox's Portable Common Runtime (PCR)	User-level
Fast Threads	User-level
Windows NT	Kernel-level
MACH	Kernel-level
Chorus	Kernel-level
OS/2	Kernel-level
Sun Microsystem's SunOS	Multiplexed
University of Washington's Fast Thread	Scheduler activation

Appendix 3

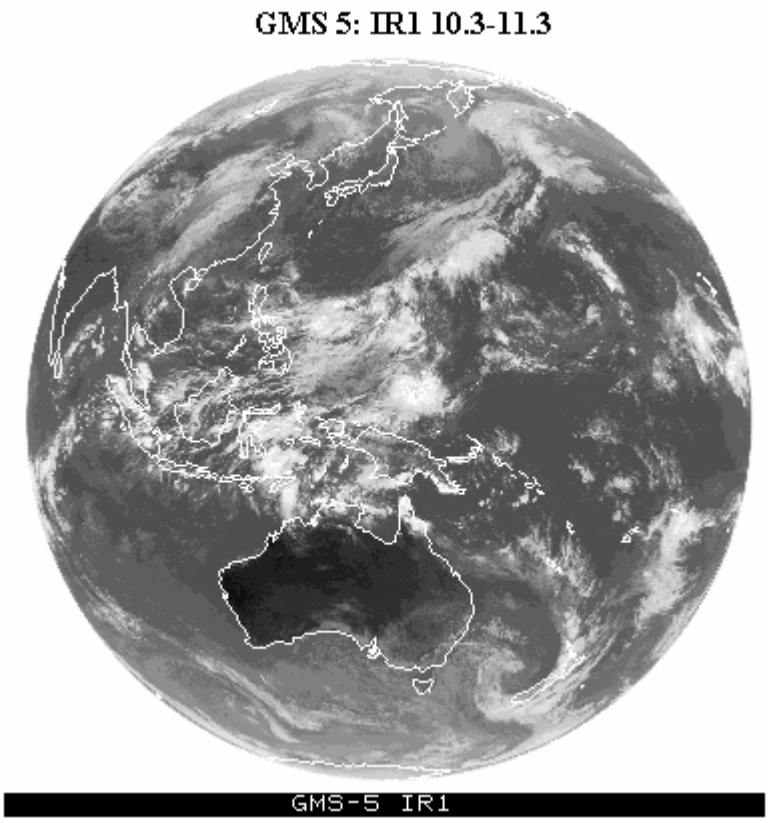
Units of Measurement Frequently Used in Remote Sensing

Length unit	Abbreviation	Value in meter
Kilometer	Km	1,000
Meter	m	1.0
Centimeter	cm	0.01 = 10^{-2}
Millimeter	mm	0.001 = 10^{-3}
Micrometer	µm	0.000001 = 10^{-6}
Nanometer	nm	10^{-9}
Angstrom	Å	10^{-10}

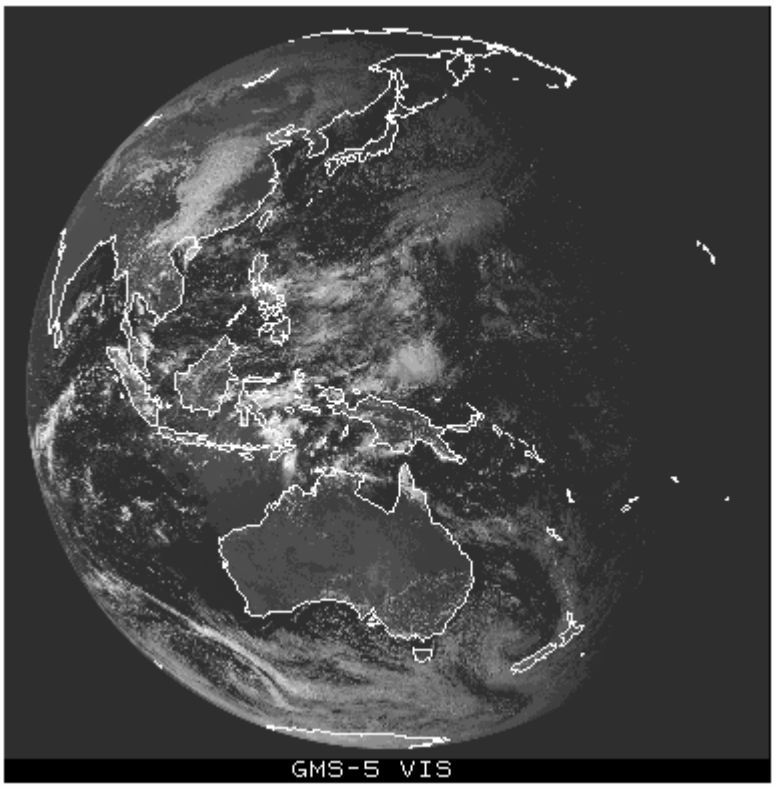
Frequency unit	Abbreviation	Value in cycles-per-second
Hertz	Hz	1
Kilohertz	Khz	1000 = 10^{3}
Megahertz	Mhz	1,000,000 = 10^{6}
Gigahertz	Ghz	= 10^{9}

Appendix 4

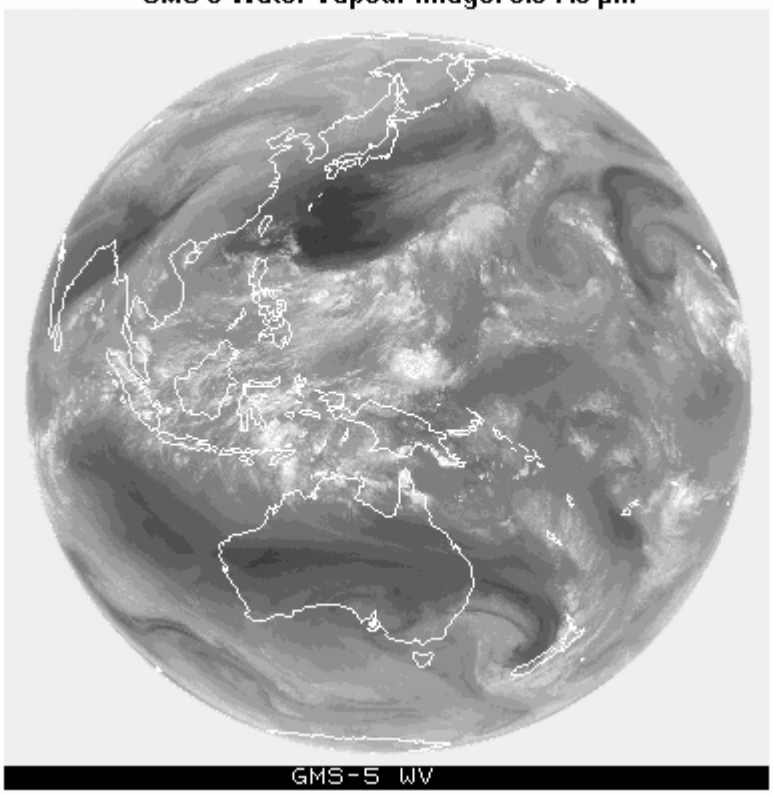
Examples of Some Types of Satellite Images



GMS 5 VIS 0.55-0.8 µm



GMS 5 Water Vapour Image: 6.5-7.0 µm



معالجة متوازية لصور كبيرة من الأقمار الصناعية باستخدام المعالجات الخيطية على شبكة حاسبات محلية

أعداد
أسامة عمر حسن قزمار

المشرف
الدكتور احمد الشرايعة

ملخـــــــــــص

اصبح هناك نمو سريع في طلب واستخدام البيانات المسجلة عن طريق الاستشعار عن بعد، وخاصة صور الأقمار الصناعية في الآونة الأخيرة، وهذا يتطلب بالتأكيد إلى عمليات معالجة كبيرة جدا والى أنظمة عالية التخزين والتي توفر كفاءة عالية وبأقل التكاليف. فنسبة لذلك، أصبحت المعالجة المتوازية و المعالجة الموزعة وبشكل سريع هي القاعدة الرئيسية أو القياسية للكفاءة العالية والمعالجة الكبيرة التي تتطلب حسابات كثيرة. فالحواسيب التي ترتبط مع بعضها على شبكة حاسبات محلية ممكن ان تستخدم كمحطة مركزية والتي لا تحتوي على حواسيب متعددة المعالجات.

هذه الرسالة قدمت حلا" يقوم بتقسيم الصورة إلى أجزاء اصغر حيث أجريت عمليات معالجة الصور على كل جزء منفرد، تم هذا عن طريق إنشاء عدد من المعالجات الخيطية (مجموعة متتالية من الأوامر) على جهاز الخادم في الشبكة المحلية، بالإضافة إلى عدد من المعالجات الخيطية على كل جهاز مستفيد في الشبكة المحلية وذلك للقراءة و المعالجة والكتابة لكل جزء من الصورة المقسمة إلى الخادم.

بعد التطبيق وتجزئة عينات من صور الأقمار الصناعية وتجربتها على شبكة الخادم–المستفيد تبين ان هذا الحل اسهم في نقصان كبير في الوقت المستغرق للعمليات الحسابية والذي يمثل دائما" التحدي الرئيسي في معالجة الصور مع نقصان في الذاكرة المطلوبة. ومن خلال التجارب التي أجريت، والقياسات والنتائج التي استخلصت من جراء التطبيق للنظام، وجد أن المعالجات الخيطية المستخدمة في شبكة الحاسبات المحلية زادت الكفاءة من خلال التوزيع للعمليات الحسابية على الشبكة. وقد دلت الاختبارات التي أجريت على هذه العينات ان تسعة معالجات خيطية هو الحل الأمثل حيث أن نسبة الكفاءة وصلت إلى أعلى من 80% عندما كان حجم الصورة 3120 x 2990، وبالتالي سوف يكون

العدد الأمثل الذي يستخدم للتطبيق في النظام. حيث أن أعلى قيمة للكفاءة هي 0.95 عندما كان عدد المعالجات الخيطية ثلاثة وحجم الصورة 3120 x 2990، و تشير هذه النتائج على أن افضل كفاءة يحصّل عليها عندما تكون نسبة حجم الصورة إلى عدد المعالجات الخيطية هي حوالي 3,109,600 بكسل لكل معالج خيطي. بالإضافة إلى ذلك، فقد حصلنا على كفاءة فعّالة ومشجّعة.