Development and Performance Evaluation of a Distributed Image Processing Model Running on LAN-Based Systems

تطوير و تقييم أداء نظام توزيعي لمعالجة الصور باستخدام شبكات الحاسوب المحلية

By

Maysoon Yousef Abu-Hammad

Supervisors

Dr. Ahmad Sharieh and Dr. Hussein Al-Bahadili

This proposal is submitted to the Department of Computer Science, Graduate College of Computing Studies, Amman Arab University for Graduate Studies in partial fulfillment for the requirement for the Degree of Master in Computer Science.

Department of Computer Science

Graduate College of Computing Studies

Amman Arab University for Graduate Studies

(November- 2008)

www.manaraa.com

Authorization

I, Maysoon Yousef Abu-Hammad, authorize Amman Arab University for Graduate Studies the right to provide copies of the dissertation to libraries, institutes, agencies or individuals when necessary.

Name: Maysoon Yousef Abu-Hammad

Signature: ____________

Date: 5/1/2009

المنارة للاستشارات

www.manaraa.com

The Discussion Committee Decision

This dissertation entitled "Development and Performance Evaluation of a Distributed Image Processing Model Running on LAN-Based Systems" was discussed and passed on November, 18, 2008.

Discussion Committee Members		Signature
Dr. Shakir Hussain	Chairman	S.M. Hussain
Dr. Muzher Al-Ani	Member	
Dr. Hussein Al-Bahadili	Member and supervisor	

المنارة للاستشارات

www.manaraa.com

Development and Performance Evaluation of a Distributed Image Processing Model Running on LAN-Based Systems

By

Maysoon Yousef Abu-Hammad

Supervisors

Dr. Ahmad Sharieh and Dr. Hussein Al-Bahadili

Abstract

Image processing applications involve different processes, such as: image enhancement, edge detection, object detection, noise removal, color quantization, etc. Image processing applications are characterized as time and memory demanding applications, and, for many practical and strategic applications, needs to be speeded up by using faster computing systems.

The main objective of this work is to develop and evaluate the performance of a Distributed Image Processing (DIP) model that can be used to perform extensive image processing computations, on continuous feed-in images, on a LAN-Based computer system. The model is based on the widely-used processor-farm parallel methodology, in which each processor executes the same program independently from other processors, each operating on a different part of the total data. Therefore, no interprocessor communication is required other than that involved in forwarding data/results to and from the processors.

For equivalent calculations, the computation times on a single processor (PC) and a LAN-based computing system of various number of processors (PCs), are compared, and the resulting speedup (*S*), parallelization efficiency (*E*), and image processing rate (□) are quoted. For the LAN-based system, the effect of

www.manaraa.com

varying the number of PCs, the number of images and execution time algorithm on the overall system speedup, efficiency, and image processing rate, are estimated.

The LAN-based DIP model demonstrates an excellent performance for recourse-demanding image processing applications in terms of simplicity, adaptability, flexibility, expandability, transparency, and high efficiency. It efficiently utilizes the computational power of a LAN-Based system using a Java Parallel Virtual Machine (JPVM) library as a data communication toolset for message passing between PCs, where, for standard image processing computations, it achieves parallelization efficiencies that vary between 87% to 60% when the number of PCs varies between 2 to 5 PCs connected through 10/100 Mbps Ethernet switch.

www.manaraa.com

تطوير و تقييم أداء نظام توزيعي لمعالجة الصور باستخدام شبكات الحاسوب المحلية

إعداد

ميسون يوسف ابوحماد

المشرفون:

د. أحمد الشرايعة و د. حسين البهادلي

Arabic Summary

ملخص

تطبيقات معالجة الصور تحتوي على تطبيقات مختلفة مثل: تحسين الصور، تحديد الحواف، تحديد الاشياء، إزالة الضوضاء و تكميم اللون. توصف تطبيقات معالجة الصور بأنها تحتاج الى وقت و سعة في الذاكرة، و بالنسبة لكثير من التطبيقات العملية لا بد من زيادة سرعة هذه التطبيقات باستخدام نظام محوسب أسرع.

الهدف الرئيسي من هذا العمل تطوير وتقييم الاداء لنظام توزيعي لمعالجة الصور (DIP) الذي يمكن أن يحسن من الوقت اللازم لمعالجة الصور في مجموعة من الصور المتتابعة، باستخدام الشبكة المحلية. و يقوم هذا النموذج على استخدام نظام واسع هو (Processor farm) وهو أحد المعالجات المتوازية، حيث يمكن لكل معالج أو جهاز حاسوب أن ينفذ البرنامج نفسه بصورة مستقله عن غيره من المعالجات أو أجهزة الحاسوب، و كل معالج يعمل على جزء مختلف من البيانات.

تم المقارنة بين زمن المعالجة في جهاز واحد و زمن المعالجة في أكثر من جهاز من خلال الشبكات المحلية،و تم رصد قيم ثلاثة معاييرهي: عامل التسريع (S)، الكفاءة (E) و معدل معالجة الصور(λ) . تم تقدير درجة تأثير كل من :عدد الأجهزة في الشبكة، عدد الصور، حجم الصورة و الوقت اللازم للبرنامج.

أداء نظام (DIP) في الشبكات المحلية كان ممتازاً في تطبيقات معالجة الصور من حيث البساطة، المرونة، الشفافية و الكفاءة العالية. و يستخدم بكفاءة نظام JPVM أداة لتبادل الرسائل بين اجهزة الحاسوب في الشبكة المحلية. في معالجة الصور الاساسية، تتراوح الكفاءة بين 87% الى 60% عندما تتراوح عدد الاجهزة في الشبكة المحلية من 2 الى 5 متصلة من خلال Ethernet بسرعة 100/10 ميجابات في الثانية.

www.manaraa.com

Acknowledgment

I would like to express my deep appreciation to my supervisors Dr. Ahmad Sharieh and Dr. Hussein Al-Bahadili for their guidance during this thesis, and for their patience in correcting all mistakes during the preparation of this thesis. I must not forget to thank Dr. Khalid Kaabnah for his comments on the image processing applications and his guidance during the preparation of this thesis.

Thanks to Sahab Secondary Girls School, for helped to use the computer labs and it resources.

Thanks to my best friend Buthyna Al-Falougi for her advice during my research.

Finally, special thanks to my parents for lights my life and tried to solve my problems. In addition, I thank them for helped me to finish this thesis.

www.manaraa.com

Table of Contents

Abstract .... a

Arabic Summary........ c

Acknowledgment .... d

Table of Contents........ e

List of Figures........ g

List of Tables........ i

Abbreviations........ j

Nomenclatures........ k

Chapter One Introduction........ 1

1.1. Image Processing Applications........ 1

1.2. Methods of Introducing Parallelism........ 2

1.3. Classes of Computer Systems 4

1.4. Computer Networks and Distributed Processing 9

1.5. LAN-Based Distributed Systems 11

1.6. Data Communication Libraries for LAN-Based Systems.... 12

1.7. Methodologies for Distributed Programming........ 13

1.8. Statement of the Problem........ 15

1.9.Organization of the thesis........ 16

Chapter Two Literatures Review 18

2.1 Parallel and Distributed Models for Image Processing Applications 18

2.2 .Parallel and Distributed Models Running on LAN-Based Systems 22

Chapter Three The Proposed Distributed Image Processing (DIP) Model 28

3.1 Image Processing Application 29

3.1.1. Convolution Function........ 30

www.manaraa.com

3.1.2 .Edge Detection 31
3.1.3.Noise Reduction 35
3.1.4.Noise in Digital Images 36
3.2 .Parallel Programming Methodologies 40
3.2.1 .Processor Farm Parallelism 40
3.3. The Proposed Distributes Image Processing (DIP) Model . 42
3.4 .Data Communication Library 44
3.4.1 Parallel Virtual Machine (PVM) 45
3.4.2 .Java Parallel Virtual Machine (JPVM) 45
3.5.Implementation of the DIP Model 47
3.6 .Performance Measures 48

Chapter Four Results and Discussions.......................................52

4.1. Investigate the Effect of the Convolution Function Kernel Size 53
4.2. Performance Evaluation 55
4.3. Performance Comparison... 62

Chapter Five Conclusions and Recommendations for Future Work..65

5.1. Conclusions... 65
5.2.Recommendations for Future Work 67

References 68

www.manaraa.com

List of Figures

Figure	.... Description	Page
1.1	MIMD memory-based sub-classification scheme: (a) Shared memory (tightly-coupled) architecture. (b) Distributed memory (loosely-coupled) architecture.	8
1.2	Classes of computer networks according to devices separation distance.	10
3.1	Applying the convolution function.	28
3.2	Sobel edge detection. (a) Original image. (b) Sobel edge detection.	30
3.3	Soble operators (G_x: vertical direction and G_y: horizontal direction).	30
3.4	Median filter.	32
3.5	Impulse noise. (a) Original image. (b) Add impulse noise to image	35
3.6	Reduction impulse noise. (a) Image corrupted by impulse noise. (b) Noise reduction using median filter with 3×3 sizes of neighborhoods of pixels	36
3.7	System architecture and the data flow of the proposed DIP model.	39
3.8	Algorithm of the DIP model.	40
3.9	JPVM architecture.	43
4.1	Comparison images. (a) Lena image with distorted wit impulsive noise. (b) Edge detected with 3x3 kernel size. (c) Edge detected with 5x5 kernel size.	48
4.2	Serial computation time vs. number of images for 3x3 and 5x5 convolution function kernel sizes.	50

www.manaraa.com

4.3	Variation of *S* with *n* for various values of *m* and 3x3 kernel size.	53
4.4	Variation of *E* with *n* for various values of *m* and 3x3 kernel size.	53
4.5	Variation of S with n for various values of m and 5x5 kernel size.	56
4.6	Variation of E with n for various values of m and 5x5 kernel size.	56
4.7	Variation of *S* with *n* for various values of *m* and kernel size.	57
4.8	Variation of *E* with *n* for various values of *m* and kernel size.	58

www.manaraa.com

List of Tables

Table	Description	Page
4.1	Serial computation time (Ts) and image processing rate (□) using 3x3 and 5x5 convolution function kernel sizes.	49
4.2	Comparison of the performance of the DIP model running on a LAN-based system of various number PCs (*n*≤5) and images (m≤500). (Results for 3x3 convolution function computation)	51
4.3	Comparison of the performance of the DIP model running on a LAN-based system of various number PCs (*n*≤5) and images (m≤500). (Results for 5x5 convolution function computation)	54

www.manaraa.com

Abbreviations

ATGP	Automatic Target Generation Process
DEDIP	Development Environment for Distributed Image Processing
DIP	Distributed Image Processing
DIPORSI	Distributed Processing Of Remotely Sensed Imagery
DLP	Data-Level Parallelism
FPGA	Field Programmable Gate Array
IEA	Iterative Error Analysis
IRS	Indian Remote Sensing
JPVM	Java Parallel Virtual Machine
JVM	Java Virtual Machine
MPI	Message Passing Interface
NUMA	Non Uniform Memory Access
PB	Processing Buffer
PEs	Processing Elements
P-IEA	Parallel Iterative Error Analysis
P-LSU	Parallel Linear Spectral Unmixing
P-MORPH	Parallel Morphological target detection algorithm
P-PPI	Parallel Pixel Purity Index
P-UFCLS	Parallel Unsupervised Fully Constrained Least Squares
PVM	Parallel Virtual Machine
RPC	Remote Procedure Call
SIMD	Single-Instruction Multiple-Data
SMT	Simultaneous Multithreading
UFCLS	Unsupervised Fully-Constrained Least Squares
VLSI	Very Large Scale Integrated

www.manaraa.com

Nomenclatures

Symbol	Description
S	Speedup factor
E	Parallelization efficiency
n	Number of active processors within the network
□	Image processing rate
G_x	Gradient in the vertical direction
G_y	Gradient in the horizontal direction
$s(i,j)$	True image
□(i,j)	The noise
$g(i,j)$	Image with additive noise
SNR	Signal-to-Noise Ratio
T_s	Time required to perform the computation on a single PC (single processor)
T_p	Time required to perform the computation on all active PCs (active processors)
T_{comp}	Actual computation CPU time
T_{comm}	Communication time
T_{over}	Overheads time
R	The ratio between the communication and computation times
m	Number of images processed
T	The total job time

للاستشارات المنارة
www.manaraa.com

Chapter One
Introduction

1.1. Image Processing Applications

Digital image processing is an ever expanding area with applications reaching out our everyday life such as medicine, space exploration, surveillance, authentication, automated industry inspection, security, and many more areas. Such applications involve different processes like image enhancement, edge detection, object detection, noise removal, color quantization, etc [Rao 06, Civ 04, and Mai 06].

The image processing application that is considered in this thesis performs edge detection in distorted or noisy images. In particular, it uses one of the most efficient and reliable edge detection algorithms, namely, Sobel algorithm [Kel 05, Rou 06]. However, due to the presence of noise, the performance of the edge detection algorithm is degraded as the noise level increases. To enhance the performance of the edge detection algorithm for processing distorted images, a pre-processing noise removal algorithm is performed for image enhancement. Median filter, which is based on the time-consuming convolution algorithm, is the most widely used filter for noise reduction or removal in distorted images. A detail description of the above image processing algorithms will be presented in Chapter 3.

Image processing applications are characterized as resource (processing time and memory) demanding applications, and, for many practical and strategic applications, needs to be speeded up [Kel 05]. Despite the fact that implementing such applications on general-purpose scalar computers is easier, but, in addition to be slow, it has other drawbacks such as memory restriction, single feed-in/feed-out (Input/output (I/O) port) and other peripheral devices limitations. The optimum and most satisfactory solutions to these problems can be achieved through using parallel or distributed computing systems [Kel 05, Bra 01, and Man 06].

www.manaraa.com

Parallel Computing is the simultaneous use of multiple computing resources to solve a computational problem. To conduct parallel computing, the computing resources can include a single computer with multiple processors (Parallel Processing), or an arbitrary number of computers connected by a network or a combination of both (Distributed Processing). The key benefit of parallel and distributed computing is to solve large and complex problems fast. Other benefits include: taking advantage of non-local resources to overcome memory constraints of a single computer, cost savings by using multiple "cheap" computing resources.

Since, this thesis is concerned with the implementation of image processing applications on distributed system architectures; in particular, a LAN-based distributed processing system, this chapter is devoted to explain how parallelism can be introduced into computer systems, provide an introduction to the different computer architectures and classifications, and the parallel methodologies that have developed to efficiently utilize the computing power of such advanced computing systems.

1.2. Methods of Introducing Parallelism

Parallelism in various forms appeared in computers produced during the 1960s, and proved to be an efficient approach. Nevertheless, greater parallelism needs to be introduced into the design of computing systems because improvement in circuit speed alone cannot produce the required performance. The parallelism was limited by the cost of logic units, so that the computer was substantially serial with only moderate capability to support parallel operation. As the cost of components decreased drastically in the last two decades, computer design has become more and more complex to achieve higher computation speed. By the end of 1960s and the beginning of 1970s several projects were undertaking for the development of truly parallel computers [Gra 02].

www.manaraa.com

The principal way of introducing parallelism into the architecture of computers can be summarized as follows [Gra 02, Ste 06]:

(1) Pipelining: The application of assembly-line techniques to improve the performance of an arithmetic or control unit.

(2) Functional: Providing several independent units for performing different functions, such as: logic, addition or multiplication, and allowing these to operate simultaneously on different data.

(3) Array: Providing an array of identical processing elements (PE) under common control, all performing the same operation simultaneously (i.e., lockstep mode) but on different data stored in the private memories.

(4) Multiprocessing: The provision of several processors, each obeying its own instructions on different data, either stored locally or in a common memory.

Of course, individual designs may combine some or all of these parallel features. For example processor array may have pipelined arithmetic units as its PEs, and one functional unit in a multi-unit computer might be a processor array.

History shows that parallelism has been used to improve the effectiveness of computers since the earliest designs, and that it has been applied at several distinct levels which might be classified as [Hoc 88]:

(1) Job level (between jobs or between phases of a job).

(2) Program level (between parts of a program or within DO LOOPs).

(3) Instruction level (between phases of instruction execution).

(4) Arithmetic and bit level (between elements of a vector or matrix or within arithmetic logic circuits.

www.manaraa.com

The main requirement of computer architectures in allowing parallelism at the job level is to provide a correctly balanced set of replicating resources, which comes under the general classification of functional parallelism, applied overall the computer installation. In this respect, it is important for the level of activity to be monitored well in all parts of the installation, so that bottle-necks can be identified, and resources added or removed as circumstances demand.

The types of parallelism that arise during the execution of a program also need to be carefully considered. Within such a program there may be sections of code that are quite independent of each other and could be executed in parallel on different processors in a multiprocessor environment (e.g., a set of linked processors). Some sections of independent code could be recognized from a logical analysis of the source code, but others will be data dependent and therefore not known until the program is executed.

In another case, different executions of a loop may be independent of each other, even through different routes are taken through the conditional statements contained in the loop. In this case, each microprocessor can be given the full code, and as many passes through the loop can be performed in parallel as there are microprocessors. This situation has important applications in many areas of science and engineering.

1.3. Classes of Computer Systems

There is wide variety of different computer systems, in particular multiple processing systems, have developed through the years. In order to provide a general framework for understanding these different types it is important to be able to order them into some kinds of taxonomy. Ideally, this classification scheme should present a methodology for the decomposition of any processing system, such that all the differences and similarities between configurations are indicated. It would be advantageous if such taxonomy could be developed to bring out the diversity in different system designs. The difficulty is choosing the minimum amount of information characterizing the computer system that should be incorporated into the classification scheme.

www.manaraa.com

Many schemes have been proposed for characterizing the various parallel and quasi-parallel computing systems in existence. Computer systems may be classified according to one of the following principles:

(1) Hardware-based classification scheme

(2) Instruction and data streams-based classification scheme (Flynn taxonomy)

In what follows a brief description is given for each of the above classification schemes. However, further details can be found in related literatures, such as: [Bau 02, Kel 05, and Rao 06].

(1) Hardware-based classification scheme

According to this classification scheme, i.e., the architecture of the hardware; three principle types of computer architecture emerged, these are [Kel 05]:

i. Pipeline (vector) computers.

ii. Array (synchronous) processors.

iii. Multiprocessor (asynchronous) systems.

Discussion of the above types of computer architecture is beyond the scope of this thesis.

(2) Instruction and data streams-based classification scheme

Computer systems may be usefully further classified according to how the machine conveys its instructions to the data being processed. This scheme was first proposed by Flynn [Fly 72]. In the approach taken by Flynn "instruction stream" and "data stream" are divided into two types, namely, single or multiple. The term stream is used to denote a sequence of items (instruction or data) as they are executed, or operated upon, by single processor. An instruction stream is a sequence of instructions executed by the machine, whereas the data stream is a sequence of data, including input or partial or temporarily results called for by the instruction stream.

www.manaraa.com

Four classes of processing systems can be identified, according to whether the instruction or data streams are single or multiple, these are:

(i) SISD (Single Instruction stream, Single Data stream)

(ii) SIMD (Single Instruction stream, Multiple Data stream)

(iii) MISD (Multiple Instruction stream, Single Data stream)

(iv) MIMD (Multiple Instruction stream, Multiple Data stream)

In what follows a brief description is given for each of the above classes. However, further details can be found in related literatures, such as: [Kel 05, Caa 05, and Man 06].

(i) SISD (Single Instruction stream, Single Data stream)

This is the conventional (i.e., serial) von Neumann computer [Wac 05] in which there is one stream of instructions (therefore, in practice, all one instruction processing unit) and each are arithmetic instruction initiates one arithmetic operation, leading to a single data stream of logically related arguments and results. It is irrelevant whether pipelining is used to speed up the processing of instructions or the arithmetic.

(ii) SIMD (Single Instruction stream, Multiple Data stream)

The SIMD category is further subdivided into vector (pipelined) and parallel (array or synchronous processors). For example, a vector processor is defined as SIMD processor, since a single vector instruction will operate on a vector (or vectors) of data to yield a result vector. Vector architecture is characterized by arithmetic units which are designed like automobile assembly; the units are segmented to perform smaller tasks, each of which may take a relatively small amount of time to complete [Kel 05].

Although, the overall time for the totals task (e.g., multiplication) may exceed the time for a conventional (scalar) arithmetic unit to perform the same function, the segmented arithmetic unit can accept vectors of operands, which stream

www.manaraa.com

through the unit in a lockstep fashion. This makes the overall operation, for example, processing a vector of 64 elements, faster as compared to a loop of 64 performed by a scalar machine.

The synchronous array processors are classified as SIMD, because all CPUs operate in lockstep mode, obeying a single instruction, taken from the Master Control Unit (MCU), with perhaps different data. Thus, we need to differentiate between SIMD vector and SIMD parallel. This distinction is important not only from an architectural standpoint, but also from a functional standpoint, since these two classes have important similarities, which might be exploitable by algorithms developer.

(iii) MISD (Multiple Instruction stream, Single Data stream)

Instead of parallelism in the data stream, it is conceivably possible to have parallelism in the instruction stream. This provided by a class of computers with MISD architecture. In this case, each operand operated upon simultaneously by several instructions. This mode of operation is generally unrealistic for parallel computers, therefore, at the present time, there is no practical machine having this type of architecture [Wac 05].

(iv) MIMD (Multiple Instruction stream, Multiple Data stream)

This is a general-purpose high performance computing systems, which uses multiple processing units to execute multiple instructions on multiple data, both independently and concurrently. Concurrency is a high level, or global, form of parallelism that denotes the independent operation of a collection of simultaneous computing activities, rather than the lockstep connection that exits in SIMD systems.

Concurrency is essentially an interactive parallelism that allows the asynchronous operation of processors in a system. Example of such architecture are provided by the data flow machine, the N-Cube processor, transputer-based Meiko Computing Surface, the ESPIRIT supernode machine, CRAY-XMP, IBM-3090, etc.

www.manaraa.com

MIMD computer systems are different in that they consist of several interconnected processing units, memory, and I/O units. This leads to a sub-classification scheme for MIMD processors, which is based on memory structure. In this respect, as shown in Figure (1.1), we have two classes, these are:

(a) Shared memory (tightly-coupled) architecture, in which the memory is shared by all processors, where all memory is equally accessible to all processors.

(b) Distributed memory (loosely-coupled) architecture, in which each processor has its own private memory, which other processors cannot access directly.

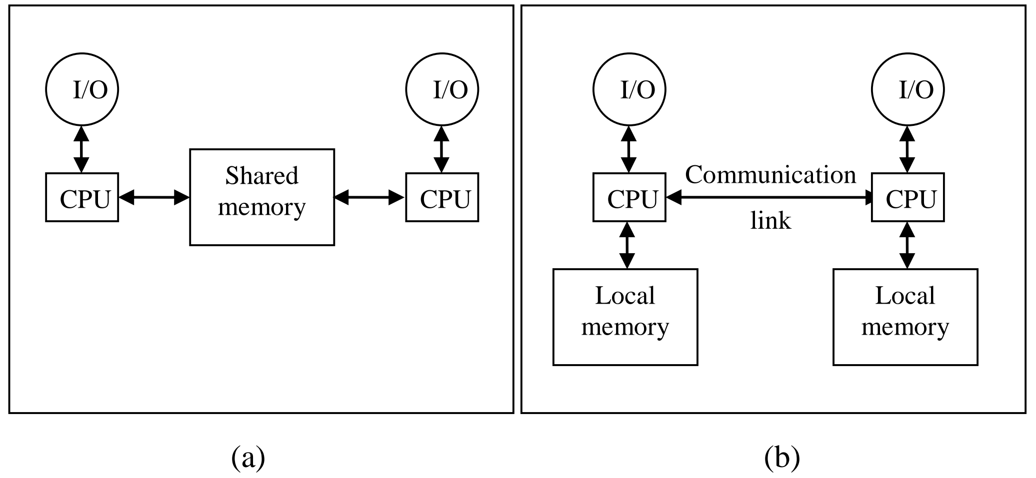


Figure (1.1). MIMD memory-based sub-classification scheme: (a) Shared memory (tightly-coupled) architecture. (b) Distributed memory (loosely-coupled) architecture.

Shared memory processors tend to be more expensive due to the need to have an expensive bus or complex switching network to allow all processors access to all memory. The shared memory approach provides fast interprocessor communication but is limited by the speed at which the shared memory can operate as the number of processors increases. With current technology, cost-effective shared memory systems tends to saturate at a relatively small number of processors and the addition of further processing units has limited effect on overall system performance.

www.manaraa.com

Distributed memory systems do not suffer from an asymptotically approached limited to their ultimate performance. However, one must then supply communication channels between processors, since it is likely that processors will need to communicate data during parallel computation. In such systems, interprocessor communication is based on “message-passing” via dedicated communication channels. It is termed message-passing since the way in which processor access the memory in processor B is via a message to B, asking B to send the appropriate data.

Thus, distributed memory and message passing are nearly synchronous terms, at least in current designs. This is turning out to be a crucial issue, due to the relatively slow interprocessor communication speeds for a distribute memory processor, compared to shared memory processor systems. Also, such systems need to provide an additional channel for each added processor, so that the system total communication performance is increased along with its processing performance. However, there is no inherent limit to the ultimate performance achievable with such computer systems. Furthermore, this approach makes it feasible to use a very large number of simple and comparatively cheap to manufacture processing units.

1.4. Computer Networks and Distributed Processing

Computer networks are defined as a number of independent digital processing devices (e.g., computers, microprocessor-based devices, personal digital adapters (PDAs), mobile phones, or any digital devices with compatible communication capabilities) that are connected together using wire or wireless data communication links [Tan 03]. Typically, each device, which also refers to them as a node, has its own central processing unit (CPU), memory, and I/O unit.

From parallel or distributed architectural point of view, and according to the discussion in Section 1.3, computer networks can be classified as distributed memory MIMD computer systems. There are many criteria that are used to classify computer networks. One widely used criterion for classifying computer

www.manaraa.com

networks is their scale (distance), i.e., their physical size. Distance is important as a classification metric because it imposes different signal propagation media, modulation techniques, protocols to be used at different scales. Accordingly, computer networks can be classified as [Tan 03]:

(1) Personal Area Network (PAN)

(2) Local Area Network (LAN)

(3) Metropolitan Area Network (MAN)

(4) Wide Area Network (WAN)

(5) International Network (Internet)

Figure (1.2) shows the distance associated with each of the above classes. Discussion of the different classification criteria and the different distance-based classes of computer networks are beyond the scope of this thesis. In this work, we concern with using a LAN as a distributed memory MIMD computer system to speedup image processing computations, and evaluate the performance of such system.

Interprocessor distance	Processors located in the same	Example
1 m	Square meter	Personal Area Network (PAN)
10 m	Room	Local Area Network (LAN)
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan Area Network (MAN)
100 km	Country	Wide Area Network (WAN)
1000 km	Continent	
10,000 km	Planet	The Internet

www.manaraa.com

Figure (1.2). Classes of computer networks according to devices separation distance [Tan 03].

1.5. LAN-Based Distributed Systems

During the last four decades, there has been an impressive gain in computer performance, in terms of speed and memory. Not only due to advances in technology, but also due to innovations in computer architectures, so that a number of high performance supercomputing systems or parallel computers is emerged. In practice, a number of parallel computers have been designed, developed, and commissioned, such as: the CRAY series, the Cyber machines, the Convex machines, US-connection machine, AMT DAP series, transputer-based Meiko Computing Surface, etc. However, running such machines are very costly, in terms of initial capital cost, running cost, hardware and software maintenance, etc.

Other efficient parallel computer systems are the application specific hardware implementation, which offer much greater speed than a software implementation and some types of parallel computers. With advances in the VLSI (Very Large Scale Integrated) technology, hardware implementation has become an attractive alternative. Implementing complex computation tasks on hardware and by exploiting parallelism and pipelining in algorithms yield significant reduction in execution times. But, using such types of parallel computers or dedicated hardware are not always cost-effective, and can not be provided for all our needs.

The use of heterogeneous collections of computing systems interconnected by one or more networks as a single logical computational resource has become a wide-spread approach to high-performance parallel computing. As an example of such cost-effective computing systems is the LAN-based computer system. The LAN-based parallel computer systems allow individual applications to harness the aggregate power of the increasingly powerful, well-networked, heterogeneous, and often largely under-utilized collections of resources available to many users [Fer 98].

www.manaraa.com

1.6. Data Communication Libraries for LAN-Based Systems

In order to be able to efficiently and effectively use LAN-based systems to perform parallel (distributed) computations, it is important to have an adequate and a reliable distributed program development toolset, which can support the programming of multiprocessors application using familiar development environments and standard languages.

Numerous software systems that have been developed to support some form of network parallel computing, the majority of them are based on a small set of popular packages that provide an explicit message-passing for distributed memory MIMD computer systems, such as the Parallel Virtual Machine (PVM), and the Message Passing Interface (MPI). These software systems support simple, portable library interfaces for typical high-performance computing languages such as C and FORTRAN [Lee 99, Fer 98].

The PVM provides the programmer with library routines to perform task creation, data marshalling, and asynchronous message passing. In addition, the PVM provides tools for specifying and managing a collection of hosts on which applications will be executed.

More recently, a new toolset based on Java language is developed; therefore, it is called Java Parallel Virtual Machine (JPVM). This is because Java provides a number of features that appear to be promising tools for addressing some of the inherent problems associated with network parallel programming. For example, Java provides a portable uniform interface to threads. Using threads instead of traditional heavyweight processes has been found to be an avenue for increasing latency tolerance and allowing finer-grained computations to achieve good performance in distributed memory parallel processing environments. Java supports a high degree of code portability and a uniform API for operating system services such as network communications [Fer 98].

www.manaraa.com

The JPVM library supports an interface similar to the C and FORTRAN interfaces provided by the PVM system, but with syntax and semantics enhancements afforded by Java and better matched to Java programming styles. The similarity between JPVM and the widely used PVM system supports a quick learning curve for experienced PVM programmers, thus making the JPVM system an accessible, low-investment target for migrating parallel applications to the Java platform. At the same time, the JPVM offers novel features not found in standard PVM such as thread safety, multiple communication end-points per task, and default-case direct message routing. The JPVM is implemented entirely in Java, and is thus highly portable among platforms supporting some version of the Java Virtual Machine (JVM). This feature opens up the possibility of utilizing resources commonly excluded from network parallel computing systems such as Macintosh and Windows-NT based systems [Fer 98].

1.7. Methodologies for Distributed Programming

Distributed programming is the design, implementation, and tuning of distributed computer programs to take advantage of distributed computing systems. Distributed programming focuses on partitioning the overall problem into separate tasks (processes and data), allocating tasks to processors and synchronizing the tasks to get meaningful results. Distributed programming can only be applied efficiently to problems that are inherently distributable, mostly without data dependence.

There are two major approaches to distributed programming: implicit, where the system (the compiler or some other program) partitions the problem and allocates tasks to processors automatically; or explicit, where the programmer must themselves suggest and implement the partitioning method for the problem [Sil 99].

For most applications, there are three common broad models, which may be considered in modelling physical systems on distributed memory MIMD computer systems [Sil 99]. The strategies are:

www.manaraa.com

(1) Arithmetic or algebraic model

(2) Geometric model

(3) Event or processor-farm model

(1) Arithmetic or algebraic model

Here the whole algorithm is split into a number of sections, each of which is assigned to one processor, but data relating the whole system flows through each processor like a production line. Thus, elaborate communication is required in transferring the data from one processor to another.

(2) Geometric model

In this model, each processor executes more or less the same program but here the data is distributed in a manner which requires extensive communication between the processors, for example, each processor might be used to simulate one part or more of a large system of similar objects interacting with each other.

(3) Event or processor-farm model

This may be considered as the simplest model in which each processor executes the same program independently from all other processors, each operates on a different part of the total data. Therefore, this model is mostly suitable for applications where the same process has to be applied to a number of independent data sets.

The processor-farm model is very simple since it allows exactly the same serial program to be implemented, assuming that enough memory is available to accommodate the whole program on each processor. In addition, the farming strategy is the most efficient model which can be used in concurrently running image processing applications. This is because, by using this strategy, no interprocessor communication is required other than that involved in forwarding data/results to and from the processors, once for all at the beginning and the end of the computation.

www.manaraa.com

1.8. Statement of the Problem

It has been well recognized that distributed computations are the most efficient solution to speedup the resource (processing time and memory) demanding image processing applications. LAN-based computer systems, which can be classified as distributed MIMD (loosely-coupled) computer systems, are considered as the most cost-effective systems. However, the performance of such systems depends on the parallel methodology that is used in implementing the application on the distributed machine.

The main objective of this work is to develop and evaluate the performance of a distributed image processing (DIP) model that utilizes a LAN-Based computer system. The DIP model is implemented to speedup the extensive image processing computations of noise reduction (median filters based on convolution function) and edge detection (Sobel algorithm) in distorted, continuously feed-in, images.

The objectives of this work can be summarized as follows:

(1) Develop a serial research level code for edge detection in distorted images using Sobel algorithm. In order to increase the edge detection accuracy of the algorithm, the input images are pre-processed for noise reduction or removal (image enhancement) using median filter, which is based on convolution function of different sizes (e.g., 3x3 and 5x5).

(2) Develop a DIP model, which is based on the processor-farm methodology, in which each processor executes the same program independently from all other processors, each operates on a different part of the total data.

(3) Develop a distributed or parallel version of the code to accommodate the processor-farm parallel methodology to run the code efficiently on a LAN-based system using the JPVM as parallel environment.

www.manaraa.com

(4) Evaluate speedup factor, parallelization efficiency, and image processing rate achieved by the DIP model over LAN-based system consisting for the pre-described image processing application.

(5) Investigate the effect of number of parameters, such as: number of PCs forming the system, number of images allocated to each PC, size of image, size of convolution function (3x3, 5x5), on the system performance.

(6) Demonstrate how to estimate, for certain problem size, the optimum number of processors that can be used.

1.9.Organization of the thesis

This chapter presents an introduction to the main topics, objectives, and outcome of this thesis. Chapter 2 presents a literature review that summarizes the most recent and related work. It is presented in two sections. Section 2.1 reviews a number of parallel and distributed models for image processing applications running on a variety of parallel and distributed system architectures. Section 2.2 reviews a number of parallel and distributed models running on LAN-based computer systems.

Chapter 3 provides a detail description of the proposed distributed image processing (DIP) model, the parallel programming methodologies in use, and the criteria that are used in developing this model. This chapter also describes in details the image processing application considered in this thesis, which includes the edge detection algorithm, namely, Sobel algorithm, and the noise reduction or removal techniques, namely, the median filter that is based on the convolution function.

Chapter 4 is devoted to present some results to evaluate the performance that can be achieved over a LAN-based system consisting of *n* PCs connected through an Ethernet 10/100 Mbps switch. The performance is measured in

www.manaraa.com

terms of the speedup factor (*S*), parallelization efficiency (*E*), the image processing rate (). The results obtained are discussed and presented in tables and/or graphs as appropriate. Finally, in Chapter 5, conclusions are drawn and recommendations for future work are pointed-out.

www.manaraa.com

Chapter Two
Literatures Review

Image processing applications are time and storage demanding applications, therefore, the implementation of image processing applications on parallel and distributed computer architectures has been an area of extensive research for the last three decades. Consequently, there are a number of techniques that have been developed to speedup such time-consuming computations. Most models are developed to run efficiently on advanced, dedicated, and expensive distributed computer architectures. However, more recently, with the enormous advancement in computer networks technology and protocols, many projects have been directed towards the implementation of more general-purpose cost-effective systems, namely, the LAN-Based systems for image processing applications.

In this chapter, we review some of the most recent and related work. This chapter is divided into two sections. Section 2.1 reviews number of parallel and distributed models for image processing applications on different types of parallel computer architectures. Section 2.2 is dedicated for parallel and distributed models that run on LAN-Based systems.

2.1 Parallel and Distributed Models for Image Processing Applications

L. Baumstark and L. Wills [Bau 02] presented a technique for extracting the two-dimensional spatial data dependencies from C image filtering source code. A key insight gained by looking at the image filtering programs is that extracting these spatial data dependencies is the critical and most difficult step; often, the core filtering computation that is applied to each neighborhood of pixels can be directly transferred over to the data parallel code unchanged. Based on this insight, their strategy was to first focus on identifying two-dimensional data reference patterns in the source code and later apply different analysis techniques (as needed) to the core filtering computation.

www.manaraa.com

Baumstark and Wills developed a reverse engineering technique to image filtering code from a commercial library originally written for the Texas Instruments TMS320C62xx family of digital signal processors. The technique was applied to common image filtering algorithms. The results obtained from this technique were validated by retargeting to a MATLAB program and matching the results against those of the original source.

H. Fatemi et. al. [Fat 04] presented and evaluated a method for introducing parallelism into an image processing application. The method is based on algorithmic skeletons for low, medium and high level image processing operations. They provided an easy-to-use parallel programming interface. Fatemi et. al. approach identified number of skeletons for parallel processing of low-level, intermediate-level and high-level image processing operations. Each skeleton can be executed on a set of processors. From this set of processors, a host processor is selected to split and distribute the image to the other processors. The other processors from the set receive a part of the image and the image operation which should be applied to it. Then the computation takes place and the result is sent back to the host processor. The programmer of the application should only select the skeleton from the library and gives the appropriate operation as a parameter.

To evaluate their approach, face recognition was implemented twice on a highly parallel processing platform, namely, the IMAP-board, once via skeletons, once directly and highly optimized. It was demonstrated that the skeleton approach is extremely convenient from a programmer's point of view, while the performance penalty of using skeletons is well below 10% in their case study.

W. Caarls et. al. [Caa 05] thought that developing embedded parallel image processing applications is usually a very hardware-dependent process, requiring deep knowledge of the processors used. In addition, image processing application consists of number of operations surrounded by control flow constructs, and it's important to run these operations concurrently. For this, they designed asynchronous Remote Procedure Call (RPC) system to exploit

www.manaraa.com

low-level image processing operation task-level parallelism to be used for algorithmic skeletons. The system was programmed in C language, divided into number of image processing operations, and applied these using function calls.

Caarls et. al. implemented a double threshold edge detection algorithm on a prototype architecture consisting of XETAL 16 MHz 320-PE SIMD processor and a TriMedia 180 MHz 5-issue VLIW processor. The result showed that the overhead of running the RPC system is around 8%, but decreasing processing time about 42%. Their result also showed that the system can achieve a significant speedup by using SIMD processor for low-level vision processing.

H. Kelash et. al. [Kel 05] presented parallel processing using multi-agent system which can be structured into application interface that allows to call particular operators or to pass image processing operation for parallelization. In their system, each agent has a very simple behavior which allows it to take a decision such as find out an edge, or region, etc., according to its position in the image and to the information enclosed in it. The system provides an environment for developing and processing image operations within distributed system. Data parallelism was implanted in this system, where all Processing Elements (PEs) receive commands from a central control processor. The system uses the CxC language, and applies Sobel and Laplace operators using different data which can be parallelized using array controller of processors where one processor associated with one pixel. They compared between their multi-agent system and the sequential execution using MATLAB. They found that the speedup factor is increasing when using multi-agent system as the size of images increases.

D. V. Rao et. al. [Rao 06] addressed the implementation of image processing algorithms such as: image filtering, image smoothing and edge detection on Field Programmable Gate Array (FPGA) using Handle-C language which is a C-based language that can provide direct implementation of hardware from the C-based language description of the system. The design was implemented on RC1000-PP Xilinx Vertex-E FPGA based hardware. The results from this design used operations for the image processing algorithms on a 256x256 size

www.manaraa.com

grayscale of Lena image show that the speed of this FPGA solution for the image processing algorithms was approximately 15 times faster than the software implementation in C language.

F. Schurz and D. Fey [Sch 07] presented a parallel processor architecture based on small Processing Elements (PEs) in a Field Programmable Gate Array (FPGA). Their architecture is able to detect and process multiple separated objects simultaneously in image which is divided into partitions and handled one by one to keep the whole design small. The architecture is using SIMD approach, which means that the same operations are carried out in parallel on each image pixel. The PEs in this design are connected through a NEWS network and controlled by a central unit. Their design is programmable using assembler language. This approach designed to be small and cheap and fast possibility for industrial image processing. The results for this design, in a VGA resolution approximately one and half million clocks, were used and 66 images can be processed at 100 MHz, which leads to a performance of 20 MPixel/s.

F. Baldacci and P. Desbarats [Bal 08] presented a parallel algorithm for 3D split and merge segmentation using topological and structuring with an Oriented Boundary Graph image processing. The researchers used multiprocessor systems and Non Uniform Memory Access (NUMA) architecture. The algorithm was tested in two machines. First machine was equipped with two Intel Xeon Quad core at 2,33 GHz, and the other was equipped with eight AMD Opteron Dual core at 1,8 GHz with NUMA architecture. They used two medical images in test: one image with size 256x256x256 voxels and the other with 512x512x475 ovxels size. The goal of the approach was to reduce the split and merge operations computation time. The results studied the execution time and showed that the NUMA architecture was two time slower than the other one, and using sixteen threads was slower than using eight threads.

www.manaraa.com

2.2 .Parallel and Distributed Models Running on LAN-Based Systems

A. Bevilacqua [Bev 99] introduced a model to obtained efficient load balancing for data parallel applications based on dynamic data assignment running on a heterogeneous cluster of workstations. The model was referred to the working-manager model. The aim of the model was to maximize the performance of the loosely coupled parallel systems. It is essential to minimize the idle time of each process and ensure the balancing of processes workload.

The cluster used consists of four workstations, connected to a LAN by a 100Mbit Ethernet, except for workstation 3, the amounts was 10Mbit adapter. The workstations hardware consists of the following:

- Workstation 1: SMP system: 2 PII 400MHz, 512 MB.
- Workstation 2: SMP system: 2 PPro 200MHz, 128 MB.
- Workstation 3: AMD K6-3D, 300MHz, 64 MB.
- Workstation 4: DEC AXP 4/200, 200MHz, 256 MB.

The operating system used Linux 2.0 for all workstations except for AXP that comes with its native OSF/1, and PVM is the communication library. The gcc and the C compiler were used in the model. The results showed that the efficiency was over 90%.

J. A. Gallud et. al. [Gal 99] presented a workbench called Distributed Processing Of Remotely Sensed Imagery (DIPORSI). It was developed to provide a framework for the distributed processing of Landsat images using a cluster of NT workstations connected by Ethernet network using the Message Passing Interface (MPI) standard.

The distributed machine in their model is composed of the 8 P II 333 MHz with 32 MB of RAM running windows NT Workstation v4.0, and the nodes were linked using a 10 Mbps Ethernet. The time in the distributed algorithm was

www.manaraa.com

compared with the time in the sequential algorithm. The results showed that the reduction of the execution time in distributed algorithm over 400% for a moderate number of nodes. The results also showed that a near linear speedup for large image size can be achieved.

H. S. Bhatt et. al. [Bha 00] developed an environment over a network of VAX/AMS and UNIX for distributed image processing. They presented a WebDedip, which is redesigned and generalization of Development Environment for Distributed Image Processing (DEDIP) to make it more user friendly and truly heterogeneous, using Java and web technology. The model uses three-tier architecture instead of master-slave one. The WebDedip has three tier architecture: GUI, Dedip Server and agents.

The functionality and efficiency of the WebDedip was tested using Microsoft NT as host and IRIS workstations as a slave. IIS 4 was used as a web server and the front-end GUI was tested on two most popular browsers IE and Netscape. The model was used by 15 scientists for development and operationalization of 10 distributed image processing applications for Indian Remote Sensing (IRS) satellite. The efficiency was as high as 90-95%. The page and applet loading time over the network was excluded and the communication delay over the network was the additional delay.

C. Nicolescu and P. Jonker [Nic 02] presented a data and task parallel low-level image processing environment for distributed memory system. They designed an approach of adding data and task parallelism to an image processing library using algorithmic skeletons and the Image Application Task Graph (IATG). In their approach, the authors allowed the application to be implemented in a C programming environment and allowed the possibility to use and implement different scheduling algorithms for obtaining the minimum execution time.

Nicolescu and Jonker presented a data parallel paradigm with the host/node approach for image processing operations where the host processor is selected for splitting and distributing the data to the other nodes and the host also

www.manaraa.com

processes a part of the image then each node processes its received part of the image and then the host gathers the image back together. The authors use a distributed system which consists of a cluster of Pentium Pro/200 MHz PCs with 64Mb RAM running Linux, and connected through Myrinet in a three-dimensional (3D)-mesh topology with dimension order routing.

The code was written using C and MPI message passing library and the multi-baseline stereo vision algorithm is an example used in their system. They compared the speedup for different image sizes in data parallel approach and the speedup of the same application using the data and task parallel approach also for different image sizes. The speedup in data and task parallel approach was more efficient than the speedup in data parallel approach.

Z. Qiu et. al. [Qiu 02] developed fast parallel stereo matching parallel algorithm on home-based software DSM JIAJIA. A cluster of eight Pentium II PCs connected by a 100 Mbps switched Ethernet are using in there design. The stereo images were divided into eight parts. Each PC carried out the matching task of one parts of stereo image. The results showed that when two PCs were used the speedup ratio is 1.8. When four PCs were used, the speedup ratio is 3.7. When eight PCs were used, the speedup ratio is 7.5. The speedup ratio is near the ideal linearity speedup ratio. The speedup of finding corresponding points reaches 3200 pair/second, when eight PCs were used.

J. O'Connell and P. Caccetta [Con 05] presented an algorithm used for time series classification of remotely sensed image data which is spatial/temporal algorithm. Their approach used homogeneous and heterogeneous clusters of computers for reducing computational time using the MPI standard library.

The parallel algorithm distributes each line of the input probability images to a number of slave nodes with I/O performed by one master node. Slave nodes then perform the necessary LS processing tasks and send the output back to the master. The parallel algorithm implemented on two clusters, an ad hoc cluster and the dedicated cluster. The ad hoc cluster used 13 office Wintel

www.manaraa.com

machines. All machines were Pentium 4 between 1.6 GHz and 3.6 GHz and of signal dual and quad CPU connected via 100 Mbit Ethernet. The MPICH implementation was used in this cluster. The results showed that the efficiency in an ad hoc cluster at least 67% in homogenous CPUs, but the efficiency in the dedicated cluster was about 86.2% (speedup 7.76) in 9 CPU, and 85.43% (speedup 41.86) in 49 CPU.

A. Clematis et. al. [Cle 06] presented an approach for high performance legacy code in a grid-oriented environment. In particular, they presented PIMA(GE)^2 Parallel IMAGE processing GEnova server obtained a legacy code parallel library. The parallel server was implemented by using CORBA and integrated in Grid architecture. The goal of the approach was reuse of a parallel image processing library in a heterogeneous environment obtained a high level of flexibility to developed client server image processing applications.

The approach implemented using a C++/MPI-2 parallel library to be used in distributed environment. The application used was the detection of linear structure in an image. They authors used a Linux cluster with eight nodes interconnected by a Gigabit switched Ethernet, and each node processor was a 2.66 GHz Pentium, 512 Mbytes of RAM and two EIDE disks interface in RAID 0. They compared between library functions and the PIMA(GE)^2, and the results showed that the variation of the speedup was around 1.3%.

A. K. Manjunathachari and K. SatyaPrasad [Man 06] designed approach to solve the convolution filter by using Simultaneous Multi-Threading (SMT), Processing Buffer (PB), and simulated in a standard LAN environment. Their approach presented a method to the bifurcation of image processing application into three fundamental layers (resource layer, linking layer and application layer) which are isolated based on processor requirements and their functionality. The parallelism was enhanced by adding the concepts of SMT over the processor for redundancy the transition delay in parallel computing image processing application. Results showed that for a large number of processing units, speedup is close to linear, and also speedup characteristics were identical when the same number of templates was used in the matching process.

www.manaraa.com

The approach used two different implementation methods for parallel image convolution. The first method was the direct convolution method which has less communication load than the other method, which was 2D Fast Fourier Transform (FFT) in a Fourier domain. Direct convolution method's scalability slightly decreased as kernel size got smaller but hardly affected by image size. The other method's scalability decreased as image size got smaller and never affected by kernel size.

A. Plaza et. al. [Pla 06] presented a parallel exploitation-based algorithm for onboard data hyperspectral data compression. In their model, three different parallel computing platforms were used for demonstration purposes: a Beowulf cluster made up of 256 processors at NASA's Goddard Space Flight Center, a heterogeneous network of 16 distributed workstations at University of Maryland, and a Xilinx Virtex-II FPGA.

In their model, they implemented many parallel algorithms such as: parallel unsupervised fully constrained least squares (P-UFCLS) algorithm, parallel iterative error analysis (P-IEA) algorithm, parallel pixel purity index (P-PPI) algorithm, parallel N-FINDR algorithm and parallel algorithm for data compression P-FINDR/P-LSU (Linear Spectral Unmixing) compression algorithm which was implemented using FPGA hardware.

The results demonstrated that massively parallel Beowulf clusters and low-cost heterogeneous networks of workstations offer an unprecedented opportunity to explore methodologies in data mining that looked to be too computationally intensive due to the immense volumes of information in remote sensing databases. To address the real-time computational requirements introduced by many applications, the authors had also developed an FPGA-based algorithm for onboard, hyperspectral data compression.

A. Paz et. al. [Paz 08] developed several parallel algorithms for target detection in hyper-spectral imagery. They developed four algorithms for target and anomaly detection in hyper-spectral images, these algorithms are: the

www.manaraa.com

Automatic Target Generation Process (ATGP), an Unsupervised Fully-Constrained Least Squares (UFCLS) algorithm, an Iterative Error Analysis (IEA) algorithm, and RX algorithm which developed by Reed and Xiaoli for anomaly detection. The problem in these algorithms were computational very expensive. The authors solved the computational problem by developed four computationally efficient parallel implementations, a parallel ATGP (P-ATGP) algorithm, a parallel UFCLS (P-UFCLS) algorithm, a parallel anomaly detector (P-RXD) and a parallel MORPHological target detection algorithm (P-MORPH). In all algorithms they used a data-driven partition strategy tested on a hyper-spectral image scene collected by the AVIRIS instrument.

The full data in the experiment consists of 2133x512 pixels, 224 spectral bands and total size about 900 Mbytes. The authors used a single processor of a Beowulf cluster with 256 processors called Thunderhead and available at NASA's Goddard Space Flight Center. The results showed that the computation time of the parallel algorithms was more efficient of the computation time in sequential algorithms.

www.manaraa.com

Chapter Three
The Proposed Distributed Image Processing (DIP) Model

In the last few decades there has been an impressive gain in computer performance, due not only to advances in hardware, but also to innovations in computer architecture (i.e., how the computer is designed and organized to perform its computational tasks) [Gra 02, and Ste 06]. The later reason leads to the emergent of extremely fast but expensive computing systems, which are recognized as supercomputers or parallel computers. In addition, as it was discussed in Chapter 1 that a powerful cost-effective parallel (distributed) computing system can be established by utilizing a number of Personal Computers (PCs) connected in a Local Area Network (LAN). This is of course subject to having an efficient message passing data communication library.

The performance of a LAN-based computer system depends on a number of factors, these include:

(1) Number of PCs used to perform the computational task concurrently.

(2) Speed of each individual PC within the network.

(3) Speed of the communication channels.

(4) Efficiency of the message passing library.

(5) Parallel programming model that is used in porting the computational task to the parallel system.

Image processing applications (e.g., edge detection in noisy images) are characterized as time consuming and memory demanding applications [Kel 05]. Fortunately, for many image processing applications, a LAN-based system can provide an efficient and cost-effective solution.

This chapter presents a description of a distributed image processing (DIP) model to speedup image processing computations by efficiently utilizing the relatively high computational power of a LAN-based system. The system uses

www.manaraa.com

the professional Java Parallel Virtual Machine (JPVM) for message passing between processors. The DIP model is based on the simple processor farm methodology that was briefly introduced Chapter 1.

The rest of chapter is organized as follows. The image processing application considered in this thesis is described in Section 3.1. The parallel programming methodology, in particular the concept, issues, and features of the event or processor-farm model are discussed in Section 3.2. Section 3.3, presents a detail description of the proposed DIP model. The main characteristics of data communication libraries in use for message passing in LAN-based systems, namely the PVM and the JPVM libraries are given in Section 3.4. Section 3.5 outlines the implementation of the DIP model using the JPVM library. Finally, in Section 3.6, the parameters (e.g., speedup factor (*S*), parallelization efficiency (*E*), and image processing rate (□)) that are used to evaluate the performance of the DIP model running on a LAN-based system are defined.

3.1 Image Processing Application

Edge detection in noisy images is a typical image processing application. It is one of the most important steps in image processing, analysis, and pattern recognition system [Civ 04]. Number of techniques have been developed for accurate edge detection, such as Canny algorithm, Sobel algorithm, etc. The main challenge for accurate and efficient edge detection is the presence of noise in images. In order to enhance the performance of such edge detection techniques, it is important, first, to reduce or ultimately remove noise from images before carrying on the edge detection process.

This thesis is concerned with the development of image processing application that performs edge detection in images distorted with impulsive noise (noisy images). Both noise reduction and edge detection algorithms used in this work are based on a simple mathematical function, it is the convolution function. Therefore, in this section, before proceeding with the description of the image processing application, a brief introduction is given for the convolution function.

www.manaraa.com

3.1.1. Convolution Function

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. It involves a multiplication of two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. It is used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values.

In an image processing context, one of the input arrays is normally just a grey level image. The second array is usually much smaller, and is also two dimensional (although it may be just a single pixel thick), and is known as the kernel. The kernel array could be in different size to get different results; the kernel may be 3×3 or 5×5 or 7×7 size of neighborhood. If the image has *M* rows and *N* columns, and the kernel has *m* rows and *n* columns then the convolution function is written as [Fis 94]:

$$H(i,j)=\sum_{k=1}^{m}\sum_{l=1}^{n}G\left(i+k-1,j+l-1\right)K\left(k,l\right) \qquad (3.1)$$

Where *i* runs from 1 to *M*-*m*+1 and *j* runs from 1 to *N*-*n*+1. The function moves the kernel *K* through the image *G* pixel by pixel, at each point the overlapping pixels in the image and kernel arrays are multiplied and then summed to get new value for the pixel.

Note that many implementations of convolution produce a larger output image than this because they relax the constraint that the kernel can only be moved to positions where it fits entirely within the image. Instead, these implementations typically slide the kernel to all positions where just the top left corner of the kernel is within the image. Therefore the kernel overlaps the image on the bottom and right edges. One advantage of this approach is that the output image is the same size as the input image. Unfortunately, in order to calculate the output pixel values for the bottom and right edges of the image, it is necessary to invent input pixel values for places where the kernel extends off the end of the image.

www.manaraa.com

Typically, pixel values of zero are chosen for regions outside the true image, but this can often distort the output image at these places. Therefore in general if you are using a convolution implementation that does this, it is better to clip the image to remove these spurious regions. Removing *n*-1 pixels from the right hand side and *m*-1 pixels from the bottom will fix things.

Convolution filter is a very complex operation that requires huge computation power. To calculate a pixel for a given kernel or mask of 3x3 there are 9 multiplications per image pixel, if the input image is 1024×1024 and the kernel is 3×3 the convolution filter need about 9 million multiplications to apply this filter in image, for this convolution filter is very slow algorithm and take long time to execution. Figure (3.1) shows how to apply the convolution function for 3×3 mask on an input image with *M* height and *N* width.

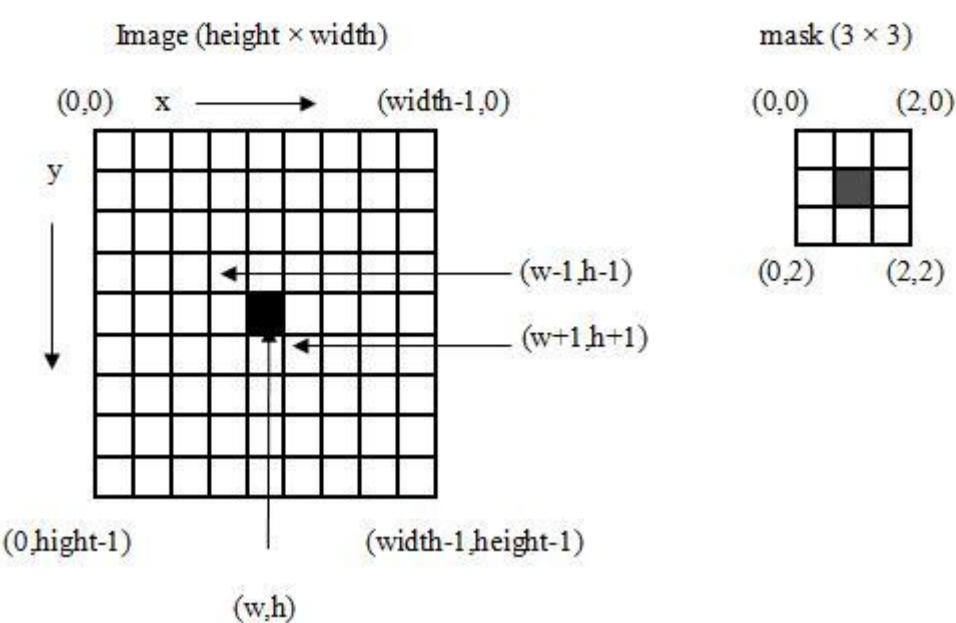


Figure (3.1) – Applying the convolution function.

3.1.2 .Edge Detection

Edge detection is the most important process in image processing, which is used in analysis and pattern recognition systems. Its importance arises from

www.manaraa.com

the fact that edges often provide an indication of the physical extent of objects within the image. The result of edge detection is edge map that contains important information about the objects in image [Civ 04].

Edge detection techniques

There are different techniques that can be used efficiently for edge detection. These techniques can be grouped into two main categories; these are [Mai 06, Civ 04]:

- Gradient edge detection: detects the edges by looking for the maximum and minimum in the first derivative of the image. This technique sometimes known as search-based methods.
- Laplacian edge detection: Known as zero-crossing based methods. This method searches for zero-crossings in the second derivative of the image in order to find edges.

In what follows, a brief description is given for the most widely used techniques, namely, the Canny and Sobel edge detection techniques as an example of gradient edge detection techniques and the Laplacian edge detection technique as an example indicated by its name.

(1) Canny edge detection technique

The Canny edge detection is kind of gradient edge detection. It could be considered as a standard method and it is used in many researches, because it provides very sharp and thin edges. Canny edge detection works in a multi-stage process, uses linear filtering with a Gaussian kernel to smooth noise and then computes the edge strength and direction for each pixel in the smoothed image [Roa 06].

(2) Sobel edge detection technique

The Sobel operator is one of the most commonly used edge detectors. Sobel operator performs a 2-D spatial gradient measurement on an image.

www.manaraa.com

The operator consists of a pair of *nxn* convolution kernels, one estimating the gradient in the columns and the other estimating the gradient in the rows. It has one disadvantage which is a slower method than other edge detection operators because its use convolution function which is time consuming.

In this work, the Sobel edge detection technique is used for edge detection in noisy images. Figure (3.2) shows the results obtained from applying the Sobel edge detection techniques for the Standard Lena image.

(a)

(b)

Figure (3.2) – Sobel edge detection. (a) Original image. (b) Sobel edge detection.

The Sobel operator is one of the most commonly used edge detectors [Wil 99]. It performs a two-dimensional (2-D) spatial gradient measurement on an image. The operator consists of a pair of 3×3 convolution kernels, one estimating the gradient in the columns (vertical) and the other estimating the gradient in the rows (horizontal) as show in Figure (3.3). The Sobel operator has one disadvantage which is a slower method than other edge detection operators because it uses convolution function which is time consuming [Fis 03].

المنارة للاستشارات
www.manaraa.com

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

Gx

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

Gy

Figure (3.3): Soble operators (G_x: vertical direction and G_y: horizontal direction).

The two operators can be combined together to find the absolute magnitude of the gradient in edge at each point and the orientation of that gradient. The gradient magnitude is given by [Fis 03]:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3.2}$$

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{3.3}$$

Where G_x is the gradient in the vertical direction, G_y is the gradient in the horizontal direction, $|G|$ is the absolute magnitude of the gradient, and θ is the orientation of the gradient.

(3) Laplacian edge detection technique

The Laplace local image operator is one of the simplest edge detection algorithms in the field of image processing. An edge is made visible by using a neighbor's pixel value to suppress its own one [Kel 05].

www.manaraa.com

3.1.3.Noise Reduction

Two dimensions convolution filter is usually used in edge detection and noise reduction applications. There are many filters used convolution function to remove the noise in image, such as, mean filter, median filter and Gaussian filter. The most common filters that are used for noise reduction are [Met 00]:

(1) Mean filter

(2) Median filter.

They both use convolution function for noise reduction in images, like Gaussian noise and impulse noise.

(1) Mean filter

Mean filter or average filter is the simplest linear spatial filter that is used for noise reduction in images. This is a low pass filter, which removes high spatial frequencies from an image and is also good at reducing Gaussian noise present in an image. Mean filter works by replacing each pixel value in an image with the mean or average value of the neighbors of the pixel and the pixel itself. Mean filter use convolution function with usually 3×3 kernel which the size of the neighborhoods. The mean filters are good at removing Gaussian noise but this type of filter blur the edges in the image [Met 00].

(2) Median filter

Median filter is a non-linear spatial filter. The median filter is calculated by sorting all the pixel values from the surrounding neighborhood into numerical order and then setting the center pixel to the middle pixel value. A median filter is a very effective in removing impulse noise in images and it also does a better job than the mean filter at preserving edges within an image. Median filter is very widely used in image and video processing applications [Met 00].

www.manaraa.com

The median filter has two main advantages over the mean filter [Fis 94]:

i. The median is a more robust average than the mean and so a single very unrepresentative pixel in a neighborhood will not affect the median value significantly.

ii. The median value must actually be the value of one of the pixels in the neighborhood; the median filter does not create new unrealistic pixel values when the filter straddles an edge. For this reason the median filter is much better at preserving sharp edges than the mean filter.

The main problem of the median filter is its high computational cost, because sorting *p* pixels the time complexity is O(*p* log *p*) [Roa 06]. However, in this work the median filter with convolution operator will be used for noise reduction in images. Figure (3.4) shows the example of median filter with mask of 3×3 of size of neighborhood pixels.

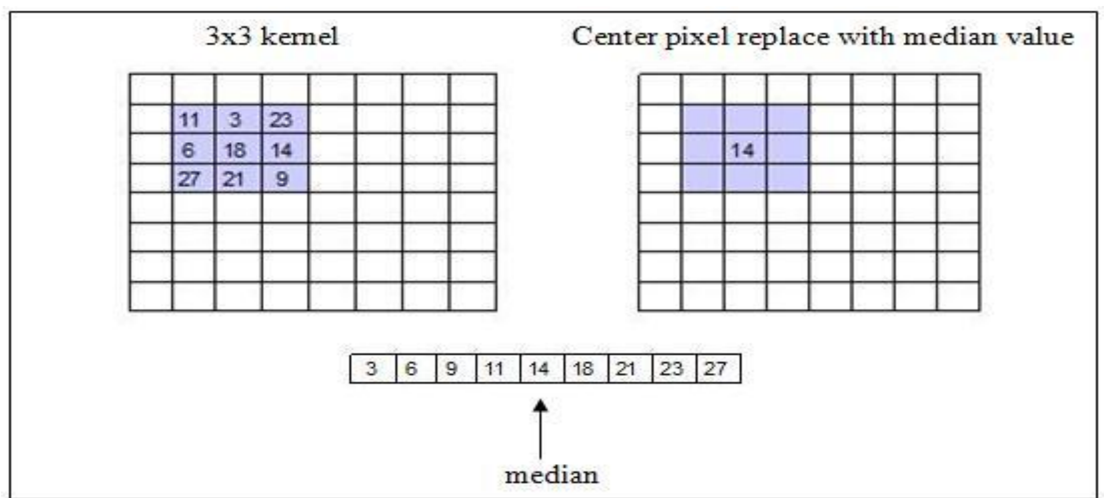


Figure (3.4) - Median filter.

3.1.4.Noise in Digital Images

Real world signals usually contain departures from the ideal signal that would be produced by the model of the signal production process. Such departures are referred to as noise. Noise arises as a result of unmodeled or unmodelable processes during the production and capture of the real signal. It is not part of

www.manaraa.com

the ideal signal and may be caused by a wide range of sources, e.g. variations in the detector sensitivity, environmental variations, the discrete nature of radiation, transmission or quantization errors, etc. It is also possible to treat irrelevant scene details as if they are image noise (e.g. surface reflectance textures). The characteristics of noise depend on its source or on the noise production operator [Mai 06].

Many image processing packages contain operators to artificially add noise to an image. Deliberately corrupting an image with noise allows us to test the resistance of an image processing operator to noise and assess the performance of various noise filters [Gon 02].

Noise can generally be grouped into two classes:

- Image independent noise.
- Image dependent noise.

Image independent noise can often be described by an additive noise model, where the recorded image *g(i,j)* is the sum of the true image *s(i,j)* and the noise □*(i,j)*:

$$g(i,j) = s(i,j) \pm \sigma(i,j) \qquad (3.4)$$

The noise □*(i,j)* is often zero-mean and described by its variance σ_n^2. The impact of the noise on the image is often described by the Signal-to-Noise Ratio (SNR):

$$SNR = \frac{\sigma_s}{\sigma_n} = \sqrt{\frac{\sigma_f^2}{\sigma_n^2} - 1} \qquad (3.5)$$

Where σ_s^2 and σ_f^2 are the variances of the true and the recorded images, respectively.

In many cases, additive noise is evenly distributed over the frequency domain (i.e. white noise), whereas an image contains mostly low frequency information.

www.manaraa.com

Hence, the noise is dominant for high frequencies and its effects can be reduced using some kind of low-pass filter. This can be done either with a frequency filter or with a spatial filter. Often a spatial filter is preferable, as it is computationally less expensive than a frequency filter.

In the second case of image data-dependent noise (e.g., arising when monochromatic radiation is scattered from a surface whose roughness is of the order of a wavelength, causing wave interference which results in image speckle), it is possible to model noise with a multiplicative or non-linear model. These models are mathematically more complicated; hence, if possible, the noise is assumed to be data independent.

Noise detector

One kind of noise which occurs in all recorded images to a certain extent is *detector noise*. This kind of noise is due to the discrete nature of radiation, *i.e.* the fact that each imaging system is recording an image by counting photons. Allowing some assumptions (which are valid for many applications) this noise can be modelled with an independent, additive model, where the noise □*(i,j)* has a zero-mean Gaussian distribution described by its standard deviation (σ_n^2), or variance. This means that each pixel in the noisy image is the sum of the true pixel value and a random Gaussian distributed noise value

Types of noise

There are many common types of noise in image, such as [Gon 02]:

(1) Gaussian noise.

(2) Impulse noise.

(1) Gaussian noise

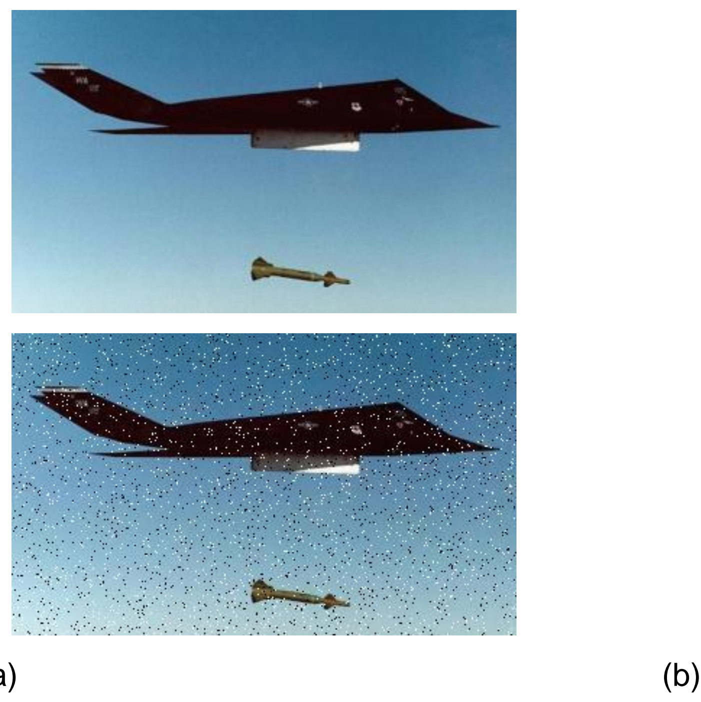
Gaussian noise is also known as additive noise or Gaussian additive noise. It is usually introduced during image acquisition. An amount of noise is added to every part of the image. Each pixel in the image will be changed from its original

www.manaraa.com

value. Additive Gaussian noise is characterized by adding to each image pixel a value from a zero-mean Gaussian distribution, this means that each pixel in the noisy image is the sum of the true pixel value and a random, Gaussian distributed noise value [Mai 06]. The Gaussian noise can be removed using mean filter usually with 3x3 mask, but mean filter blur the edges and details in images.

(2) Impulse noise

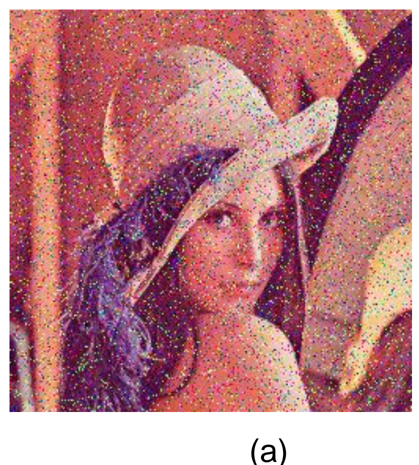
Impulse noise (salt and pepper noise) is a special type of noise caused by errors in data transmission, acquisition, in processing in images. The corrupted pixels are either set to the maximum values, which show as a snow in image, or the single pixels set to zero value, giving the image a salt and pepper like appearance [Mai 06]. Impulse noise also known as random noise or independent noise defining characteristic is that the color of a noisy pixel bears no relation to the color of surrounding pixels. Figure (3.5) shows the effect of impulse noise in color images. Nonlinear filter such as median filter is the most common filter used to reduce the impulse noise in images as shown in Figure (3.6).



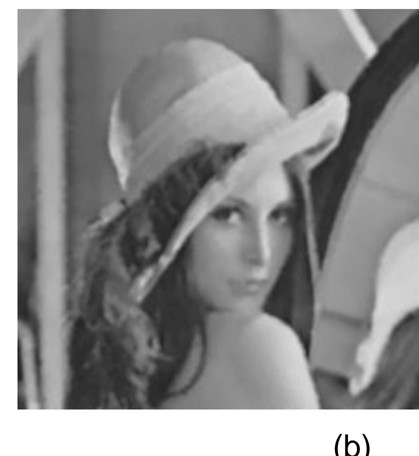
(a) (b)

Figure (3.5) – Impulse noise. (a) Original image. (b) Add impulse noise to image [Mat 04].

www.manaraa.com



(a)



(b)

Figure (3.6) –Reduction impulse noise. (a) Image corrupted by impulse noise. (b) Noise reduction using median filter with 3×3 sizes of neighborhoods of pixels.

3.2 .Parallel Programming Methodologies

Chapter 1 discussed three different explicit distributed (i.e., parallel) programming methodologies, theses are:

(4) Arithmetic or algebraic model

(5) Geometric model

(6) Event or processor-farm model

This work utilizes the third model, the event or processor-farm model, to develop an efficient distributed image processing model. The concept, issues, and features of this model are discussed below.

3.2.1 .Processor Farm Parallelism

One paradigm that is commonly used on multiprocessors is the processor-farm paradigm, in which each processor executes almost the same program independently from all other processors, each operates on a different part of

www.manaraa.com

the total data [Wag 97]. The processor-farm (also called task-farming or master-slave) paradigm consists of two entities: master and multiple slaves. The master is responsible for decomposing the problem into small tasks (and distributes these tasks among a farm of slave processes), as well as for gathering the partial results, which is run independently, in order to produce the final result of the computation. The slave processes execute in a very simple cycle: get a message with the task, process the task, and send the result to the master. Therefore, the communication takes place only between the master and the slaves, and by using this strategy, no interprocessor communication is required other than that involved in forwarding data/results to and from the processors, once for all at the beginning and the end of the computation [Hey 00].

The idea of a process farm is that a master process can compute the sequential portions of the code independently, and then spawn slave processes when it is time to do the parallel sections of code. This has the benefit that the parallelizable and sequential portions of code can be completely separated [Sac 03].

Accordingly, this paradigm can achieve high computational speedups and an interesting degree of scalability. However, for a large number of processors the centralized control of the master process can become a bottleneck to the applications. It is, however, possible to enhance the scalability of the paradigm by extending the single master to a set of masters, each of them controlling a different group of process slaves.

For applications, such as image processing, where the same computations have to be applied to a number of independent data sets, the processor-farm model is becoming as the most suitable model. This is because it allows exactly the same serial program to be implemented, assuming that enough memory is available to accommodate the whole program, on each processor.

www.manaraa.com

3.3. The Proposed Distributes Image Processing (DIP) Model

The image processing application described in Section 3.1 can be used efficiently for edge detection in noisy images. But the algorithms used in this applications (median filter for noise reduction and Sobel algorithm for edge detection) are considered as time consuming application. This is because both the median filter and Sobel algorithm are based on the time-consuming convolution function for noise reduction and edge detection. This is especially true when the kernel size in convolution function is increased. To overcome this issue, this thesis is devoted to utilize the potential computing power of LAN-based systems. Consequently, to exploit the computing power of such systems, an efficient distribution or parallelization model is needed.

This section presents a description of a distributed image processing model. It is referred to as the DIP model. It is proposed to speedup the time consuming image processing computation described above, and also to run efficiently on a LAN-based computing system that utilizes the JPVM as a parallel environment and data communication library for message passing.

The parallel methodology that is going to be used in transferring (parallelizing) the serial computation is based on the processor-farm strategy. In which the modified (parallelized) version of the code is developed in two versions, one version is developed to run as a master (server), and the other version is to run as a slave (client). More than one slave is usually loaded and run concurrently on different processor performing its calculation on different data (images), i.e., in MIMD architecture. This form of paradigm involves no inter-processor communication and slaves are only allowed to communicate with the master. The relationship between the master-slave is summarized as follows:

(i) The master reads-in the input data (images) and perform any require preliminary computations.

www.manaraa.com

(ii) Send the data to the appropriate slave.

(iii) After completion of the computations, the slave sends the output results back to the master processor.

(iv) The master processor performs any post-processing computations and then presents the final outputs.

It is clear from the above relationship that the master processor is idle while waiting for the slaves to complete their computations. Therefore, in order to utilize the master processor to perform some useful computations, instead of being idle while waiting, it is used to run as a slave. However, since the master-processor starts performing its computation after sending all data to all slaves (after all slaves start), then it is expected that all of them will finish first, and they have to wait until the master complete its computations before sending their results back to the master. In order to avoid this conflict, the size of the task assigned to the master should be less, so that it can finish before the slaves complete their computations, and be ready to receive their results.

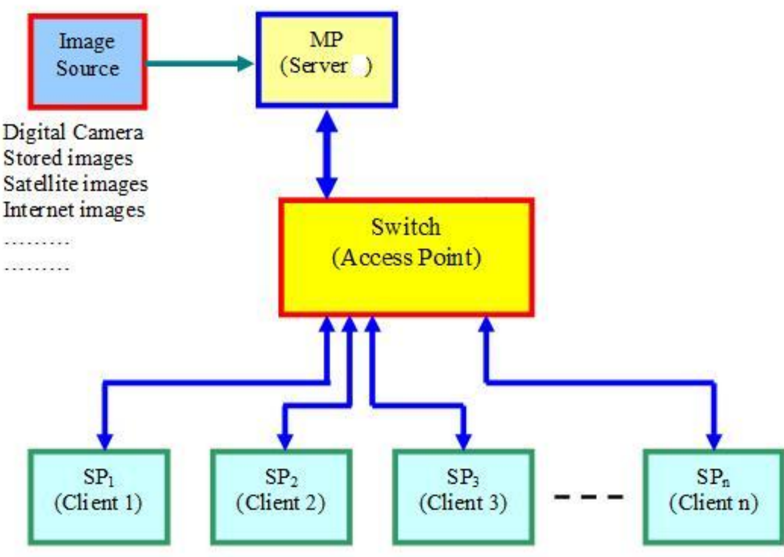


Figure (3.7) - System architecture and the data flow of the proposed DIP model.

www.manaraa.com

The DIP model can efficiently process many images at a time. These images may be obtained from on-line (real-time) or offline image sources. On-line image sources include capturing devices, such as: digital camera, satellite images, etc. Off-line image processing means those images which are captured, stored to be processed in a later time, such as internet images.

DIP uses JPVM environment as message passing tools between PCs in the network, and the algorithm in DIP model implemented in java language.

Algorithm for Master	Algorithm for slave
Step 1: Initialize LAN-based System.	**Step 1**: Initialize LAN-based System.
Step 2: Read images data.	**Step 2**: Receive images from master.
Step 3: Initial timer.	**Step 3**: Post-processes for images.
Step 4: Select number of slaves.	**Step 4**: Send images to master.
Step 5: Calculate slave job.	**End**
Step 6: Send images to slaves.	
Step 7: Post-processes for images.	
Step 8: Receives images from slaves.	
Step 9: Stop timer.	
Step 10: Post processes for images.	
End	

Figure (3.8)- Algorithm of the DIP model.

3.4 .Data Communication Library

In LAN-based distributed system, data can only be exchanged among PCs using a message passing methodology. There are many message passing libraries in use in distributed LAN-based distributed systems, such as: Message-Passing Interface (MPI), Parallel Virtual Machine (PVM) and Java Parallel Virtual Machine (JPVM) [Lee 99].

www.manaraa.com

3.4.1 Parallel Virtual Machine (PVM)

Parallel Virtual Machine (PVM) is a software system that permits a collection of many heterogeneous computers that networked together to be one large computer working in parallel mode [Yal 98]. The PVM is designed to link computing resources together and provided users with a transparent efficient parallel platform for running their computer applications. The PVM transparently and independently handles all message routing data conversion and task scheduling across a network of incompatible computer architectures. Therefore, it is used in many sites all over the world to solve important problems in scientific, engineering, industrial and in medical applications.

The PVM system can be used to run different computers in parallel mode (concurrently), and it is designed to have many important features and capabilities, such as:

(1) Reduce the cost to solve problems.

(2) Reduce the contention for resources.

(3) More effective implementations of an application.

(4) Make the parallel programming in a heterogeneous collection of processors straightforward.

3.4.2 .Java Parallel Virtual Machine (JPVM)

The Java language and its libraries and environment provide a powerful and flexible platform for programming computer clusters. Java tools enable experimentation in both management aspects as well as performance aspects of cluster systems [Haw 99].

The JPVM is a PVM like library of object classes implemented in and for use with the Java programming language. The library supports an interface similar to C and FORTRAN interfaces provided by the PVM system, but with syntax and semantics enhancements afforded by Java and better matched to Java programming styles.

www.manaraa.com

The JPVM is a combination of both ease of programming inherited from Java and high performance through parallelism inherited from the PVM. The JPVM library is software used for message passing in distributed memory MIMD LAN-Based parallel computing system. The JPVM has many features not found in standard PVM, such as [Pes 04]:

(1) JPVM is thread safety; it can control multiple Java threads inside a single JPVM task.

(2) Standard PVM has single communication end-points for every task, but the JPVM can create a new task within a process every time, so it has multiple communication end-points for each task.

(3) JPVM code can be maintained much simpler than the PVM across heterogeneous machine.

(4) JPVM has default-case direct message routing.

For as mention features of Java language and JPVM tools, the DIP model use the JPVM as a parallel environment.

As in the PVM, the programmer decomposes the problem to be solved into a set of cooperating sequential task implementations. These sequential tasks execute on a collection of available processors and invoke special library routines to control the creation of additional tasks and to pass messages among tasks. In JPVM, task implementations are coded in Java, and support for task creation and message passing is provided by the JPVM library.

The architecture of the JPVM is similar to architecture of the PVM, which is consisting of the daemon, the console and the interface library functions. The JPVM library routines require run-time support during execution in the form of a set of JPVM daemon processes running on the available collection of processors. The console can start in any processors in the network. The JPVM console can be used to list the hosts available to the system and the JPVM tasks running in the system.

www.manaraa.com

Tasks in the JPVM environments are process-based; however the communications are using Transfer Control Protocol (TCP) sockets through the network. Figure (3.9) outlines the JPVM architecture.

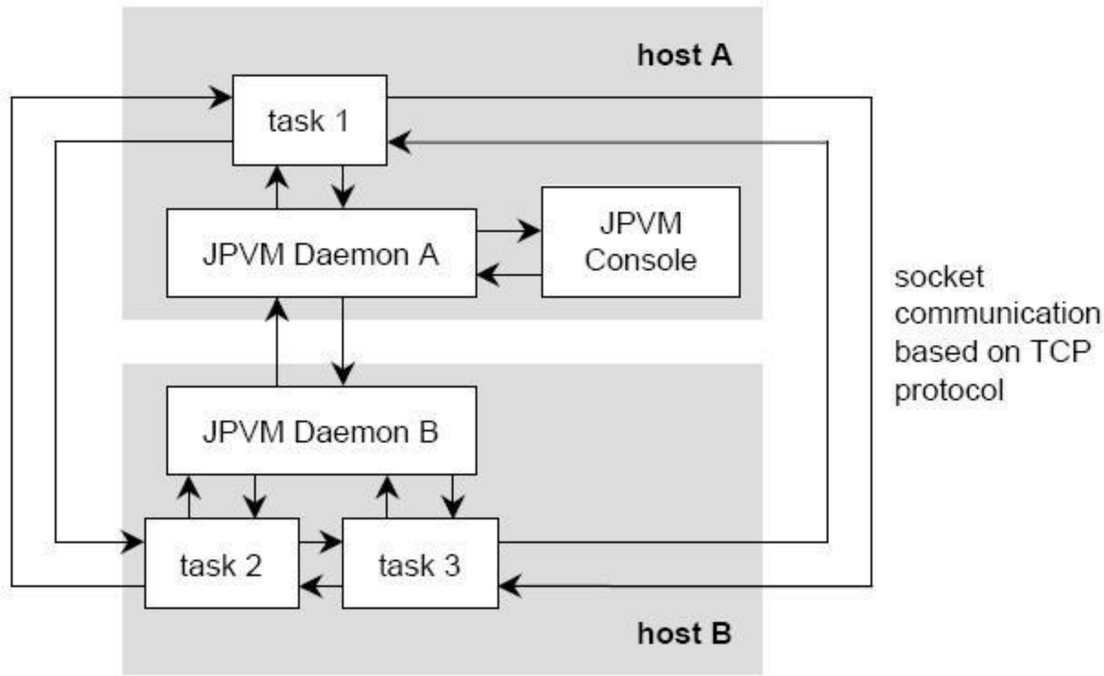


Figure (3.9): JPVM architecture [Lee 99].

3.5.Implementation of the DIP Model

First the JPVM platform must starting by run *jpvmDaemon.java* program in all computers in the LAN-based system, then run *jpvmConsole.java* in one computer as a master computer. The master controls the message passing techniques in the network. There are two programs running in the DIP model, the first program is the master (server) and the other program is the slave (client), which is duplicated on all slave processors. Both of programs have the same noise reduction and edges detection algorithms.

The master PC starts to capture or input sequence of noisy images from devices or files, these images may have similar or different sizes. Task creation start in master program by using *jpvm.pvm_spawn( )* method, which has number of slaves and the java class program for slave.

www.manaraa.com

When the master has number of images it is start to distribute these images between slaves by send the same number of images for each slave in the LAN, the distribution of images depends on the total number of images and number of slaves.

The master does not send all images at the same time; it is sends images one by one to each slave using a *for-loop* to prevent slaves from being idle while waiting until first slaves receive their images, the images are storing in buffers. Then each slave reads one image from the buffer and start processing. When a slave finishes processing the image it read, then it reads another image from buffer and so on until it processes all images sent by the master.

At this time when the master finishes send all images to slaves it start processing number of images to save time and to be not idle and waiting receive from slaves. This technique utilizes load balancing for computer in LAN system. The number of images for master must be less than number of images sends to each slave, because the master has high communication time than slaves. Also number of processing images in master depends on number of all images and number of slaves in LAN system.

When slaves finish processing all images, they start send processed images back to the master. The master starts receiving all images from slaves and then the master output all images.

3.6 .Performance Measures

In order to measure the performance of a parallel algorithm, two main factors are considered, these are:

(1) Speedup factor (*S*)

(2) Parallelization Efficiency (*E*)

In what follows an introduction is given for each of them.

www.manaraa.com

(1) Speedup factor (S)

In general, the speedup factor is defined as the ratio between the time required to perform a particular computation on a sequential mode machine (T_s) and the time required to perform an equivalent computation on a parallel mode machine (T_p). Thus, the speedup factor is expressed as [Wil 99]:

$$S = \frac{T_S}{T_p}$$

(3.6)

However, for a LAN-based distributed computing system, T_s represents the time required to perform the computation on a single PC (single processor), and T_p is the time required to perform the computation on all active PCs (active processors) that includes both the master and all servers that are participating in the computations.

Ideally, the maximum speedup that can be achieved is equal to the number of active PCs within the LAN network. However, there are several factors that limit and prevent the speedup from reaching its maximum value, such as:

i. Load balancing when not all processors perform useful computation all the time, and some of the processor may be left simply idle for a period of time during the computation.

ii. Software overhead due to the extra computation may be required in the parallel version of the code not appearing in the sequential version, for example, to recomputed constants locally.

iii. Communication time for data and messages exchange among the processors.

(2) Parallelization efficiency (E)

Another factor of interest is the parallelization efficiency, which is defined as the ratio between the speedup factor that is achieved and the maximum possible

www.manaraa.com

speedup factor. The parallelization efficiency may be expressed as [Wil 99]:

$$E = \frac{S}{n} \times 100 \quad (3.7)$$

The n is the number of active processors within the network (master plus the number of running slaves).

Another way to define E as the actual computation CPU time (T_{comp}) divided by the total computation and communication times (T_p) which represents the sum of the computation CPU time (T_{comp}), communication time (T_{comm}), and other timing overheads (T_{over}). Accordingly, the parallelization efficiency can be given as:

$$E = \frac{T_{comp}}{T_{comp} + T_{comm} + T_{over}} \times 100 \quad (3.8)$$

Since, the T_{over}, including setup time, is very small compared to T_{comp} and T_{comm}, then T_{over} can be neglected and E is expressed as:

$$E = \frac{T_{comp}}{T_{comp} + T_{comm}} \times 100 \quad (3.9)$$

It is also can be expressed as:

$$E = \frac{1}{1 + R} \times 100 \quad (3.10)$$

Where R is the ratio between the communication and computation times. It is clear from the above two equations that E depends on the amount of time that is spent on communication or on the ratio R between the communication and computation times. The maximum efficiency can be achieved when T_{comm} (i.e., R) approaches zero.

In this thesis, we introduce another parameter which is the image processing rate (□). It is defined as the number of images that can be processed by the systems per unit time (say, sec). It can be expressed as:

www.manaraa.com

$$\lambda = \frac{m}{T} \tag{3.11}$$

Where

□ is the image processing rate (Image/sec),

m is the number of images processed (Images),

T is the total job time (sec).

For a serial computations T is take to be equal to T_s, while for a parallel or distributed computations T is taken to be equal to T_p, where T_p is the sum of both T_{comp} and T_{comm}.

www.manaraa.com

Chapter Four
Results and Discussions

This chapter presents the performance evaluation of the distributed image processing (DIP) model described in Chapter 3. The model is implemented to run on a standard LAN-based computer system. The LAN composes from a number of Personal Computers (PCs) interconnected through an Ethernet 10/100 Mbps switch. The PCs used are Acer verition GT series, Intel (R), Pentium 4 processor with 2.8 GHz speed. The operating system is Windows XP.

Two versions of the image processing application are developed, one is a serial version to run on a single PC for estimating the serial computation time (T_s) (also referred to as T_1), and a parallel version of code using the DIP model for estimating the parallel computation time (T_p) (also referred to as T_n, where n is the number of PCs on which the computations are concurrently performed). Both programs are written in Java language and the Java Parallel Virtual Machine (JPVM) is used as data communication library for message passing between the processors.

The image processing analysis considered is performing edge detection in noisy images. Sobel algorithm is used for efficient edge detection, and in order to enhance the performance of the algorithm for estimating edges in noisy images, a noise reduction techniques based on median filter is used to reduce or ultimately remove noise for the image before proceeding with edge detection. Both the Sobel algorithm and the median filter are based on convolution function. The accuracy of the edge detection process depends on the size of the convolution function. In this work, a convolution function of 3x3 and 5x5 kernel sizes are considered.

The performance of the DIP model is evaluated in terms of a number of parameters, which are defined in Chapter 3, these are:

www.manaraa.com

(1) Speedup factor (*S*)

(2) Parallelization efficiency (*E*)

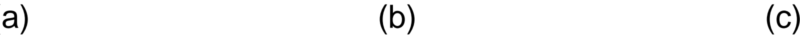
(3) Image processing rate (□) (Image/sec)

Section 4.1 investigates the effect of the convolution function kernel size on the total computation time, which enable use to estimate the image average processing time. In Section 4.2, the performance of the DIP model when implemented on a LAN-based system to perform image processing analysis, using 3x3 and 5x5 convolution function kernel sizes, on various numbers of noisy images of 256x256 size. The performance of the DIP model for these two different kernel sizes is compared in Section 4.3.

4.1. Investigate the Effect of the Convolution Function Kernel Size

It has been discussed in Chapter 3 that the image processing application this thesis concerned with is edge detection in images distorted with impulsive noise. The edge detection algorithm used is the astonishing Sobel algorithm. In order to enhance the performance of this algorithm for effectively estimating edges in noisy images, each image is preprocessed for noise reduction using median filter. Both the edge detection algorithm and the noise reduction filter use a convolution function. The size of the convolution function affects the performance of the application in terms of accuracy and processing (computation) time. A more accurate solution can always be obtained with higher kernel size (see Figure (4.1)), but this is at the cost of higher computation time.

www.manaraa.com



(a) (b) (c)

Figure (4.1): Comparison images.

(a) Lena image with distorted wit impulsive noise.

(b) Edge detected with 3x3 kernel size.

(c) Edge detected with 5x5 kernel size.

This section investigates the effect of the size of convolution function on the computation time (T_{comp}) for various numbers of noisy images (m) of size 256x256. The results obtained are tabulated in Table (4.1) and plotted in Figure (4.2). The results obtained illustrate the following:

i. The T_{comp} increases linearly with m.

ii. The average T_{comp} is about 0.12 sec/image for 3x3 kernel size, and about 0.36 sec/image for 5x5 kernel size.

iii. The T_{comp} for 5x5 kernel size is about 3 times the time for 3x3 kernel size.

iv. The image processing time is around 8 image/sec for 3x3 kernel size and around 2.7 image/sec for 5x5 kernel size, because the 5x5 kernel is more time consuming. It is calculated by dividing the total number of images processed by total computation time as given by Equation (3.11). But, at the same time it is more accurate.

Table (4.1) Serial computation time (Ts) and image processing rate (□) using 3x3 and 5x5 convolution function kernel sizes.				
No. of images	Computation time (T_s) (sec)		Image processing rate (□) (Image/sec)	
	3x3	5x5	3x3	5x5
100	12.50	36.63	8.00	2.73
200	24.91	73.06	8.03	2.74
300	37.20	109.83	8.07	2.73
400	49.66	146.02	8.05	2.74
500	62.03	182.50	8.06	2.74
600	74.82	219.90	8.02	2.73
700	86.88	255.77	8.06	2.74
800	99.45	292.05	8.04	2.74

www.manaraa.com

900	111.21	328.63	8.09	2.74
1000	123.73	365.27	8.08	2.74
1100	136.02	401.28	8.09	2.74
1200	149.80	437.72	8.01	2.74
1300	162.66	473.92	7.99	2.74
1400	174.78	509.27	8.01	2.75
1500	186.01	545.20	8.06	2.75

- Number of operations in 3x3 kernel size convolution function are 589824.
- Number of operations in 5x5 kernel size convolution function are 1638400.

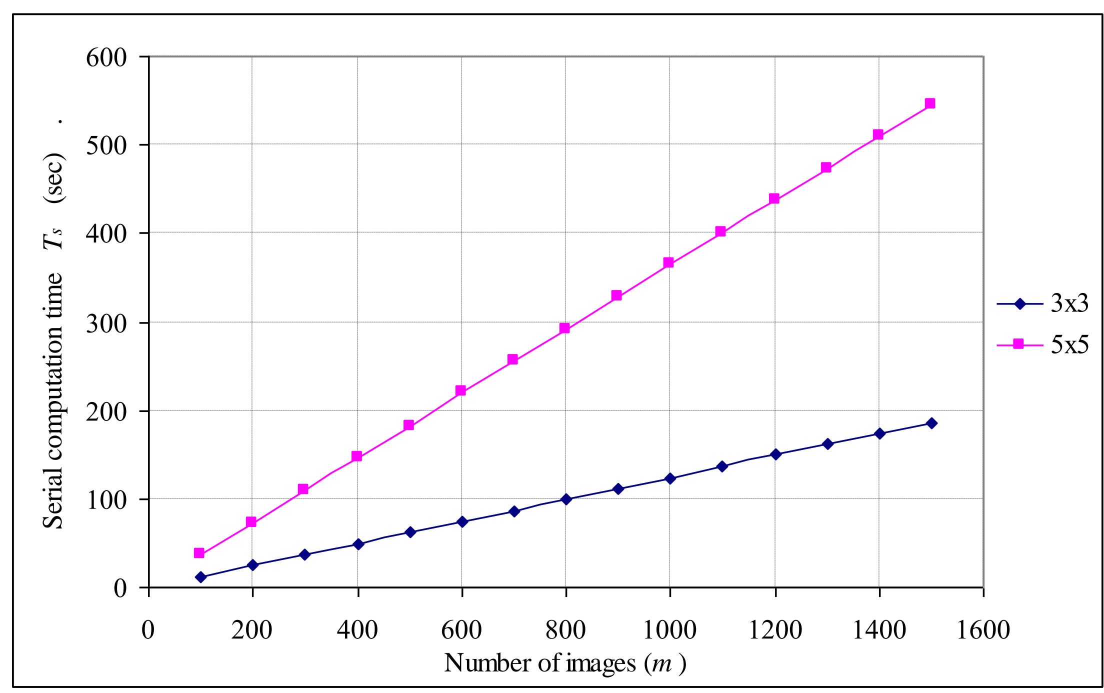


Figure (4.2). Serial computation time vs. number of images for 3x3 and 5x5 convolution function kernel sizes.

4.2. Performance Evaluation

This section evaluates the performance of the DIP model running on the LAN-based computer systems described above. The performance is evaluated in terms of *S*, *E*, and □ for various numbers of collaborative PCs (*n*), and various *m* feed into the system. The accuracy of the serial and the parallel versions of

www.manaraa.com

code are validated against each other, and they demonstrate an excellent agreement between them. Furthermore, in order to investigate the effect of T_{comp} and T_{comm} (communication time) on the performance of the model, the image processing computations are performed for a convolution function of various kernel size, these are: 3x3 and 5x5.

(1) Results for 3x3 convolution function kernel size

For a convolution function of 3x3 kernel size, the DIP model is used to parallelize the image processing application for edge detection in noisy images using the algorithms described in Figure (3.8), to run on the LAN-based system. The computation is performed for various *m*, in practice, up to 1500 images were processed, but the results demonstrate little variation, therefore, we present results for up to 500 images.

The results obtained for the serial T_{comp} (in this case it is also referred to as T_s); which is equivalent to the T_{comp} on a single PC (T_1), parallel T_{comp} (in this case it is also referred to as T_p); which is equivalent to the T_{comp} on *n* PCs (T_n) ($n \geq 2$), *S*, *E*, and □, are presented in Table (4.2).

Table (4.2)
Comparison of the performance of the DIP model running on a LAN-based system of various number PCs ($n \leq 5$) and images ($m \leq 500$).
(Results for 3x3 convolution function computation)

No. of images	No. of processors (PCs) (*n*)				
	1	2	3	4	5
CPU time (sec)					
	T_s	T_p			
	T_1	T_2	T_3	T_4	T_5
100	12.50	9.31	8.62	8.30	7.67
200	24.91	17.88	17.20	15.76	15.06
300	37.20	29.88	25.71	23.79	22.30
400	49.66	41.58	35.00	31.62	29.89
500	62.03	52.92	43.17	38.91	36.88
Speedup factor (*S*)					
100	Serial Computation (*S*=1)	1.34	1.45	1.51	1.63
200		1.39	1.45	1.58	1.65
300		1.24	1.45	1.56	1.67

www.manaraa.com

400		1.19	1.42	1.57	1.66
500		1.17	1.44	1.59	1.68
Parallelization efficiency (*E*) (%)					
100	Serial Computation (*E*=100)	67.1	48.4	37.7	32.6
200		69.7	48.3	39.5	33.1
300		62.2	48.2	39.1	33.4
400		59.7	47.3	39.3	33.2
500		58.6	47.9	39.9	33.6
Image processing rate (□) (Image/sec)					
100	8.00	10.74	11.60	12.05	13.03
200	8.03	11.19	11.63	12.69	13.28
300	8.07	10.04	11.67	12.61	13.45
400	8.05	9.62	11.43	12.65	13.38
500	8.06	9.45	11.58	12.85	13.56

According to the results presented in Table (4.2), the following points can be identified:

i. For equivalent computations (i.e., for the same number of images), E is decreasing as n increases, and almost it steadily decreases between 70% for 2 PCs to 30% for 5 PCs.

   This is because as given in Equations (3.9) and (3.10), when the number of PCs increases, the number of images allocated for each PC is decreased and consequently the T_{comp} is also decreased. Furthermore, as n increases, the T_{comm} is increased, which means R is increased, where R is defined as the ratio between T_{comm} and T_{comp}.

   The T_{comm} is increasing as n increases, because the total number of images exchanged across the networks is also increased. For example, if m=500 images, for n=2 PCs, the number of images exchanges is 250 images. While for n=3, 4, and 5, the numbers of images exchanged are 334, 375, 400 images, respectively.

ii. For serial computation (1 PC), as discussed in Section 4.1, the total T_{comp} increases steadily as m increases, with average T_{comp} of 0.12 sec/image. For parallel computation (n≥2), we can recognize two types of behavior. For n=2 and n=3, S and E are decreasing as m

www.manaraa.com

iii. increases. For example, for n=2, E decreases from 67.1% (m=100 images) to 58.6% (500 images). On average, we can estimate that it is reduced by ≈2% for each extra 100 images. However, for n=3, it is reduced by ≈0.1% for each extra 100 images. For a certain value of n (n>3), S and E are almost remained unchanged with m, however, only slight increase is recognized.

iv. A speedup of approximately 1.6 can be achieved for 5 PCs running concurrently with a parallelization of efficiency of just above 30%. This is mainly because the processing time per image is very low, and although we increase the number of images, but this adds communication overheads, so that R almost remains unchanged and consequently the parallelization efficiency.

v. For a particular m, it is clear that when n increases, T_p is decreasing, and consequently □ increases as it is given by Eqn. (3.11) as, where □=m/T_p.

Figures (4.3) and (4.4) show the variation of S and E with n for various values of m.

www.manaraa.com

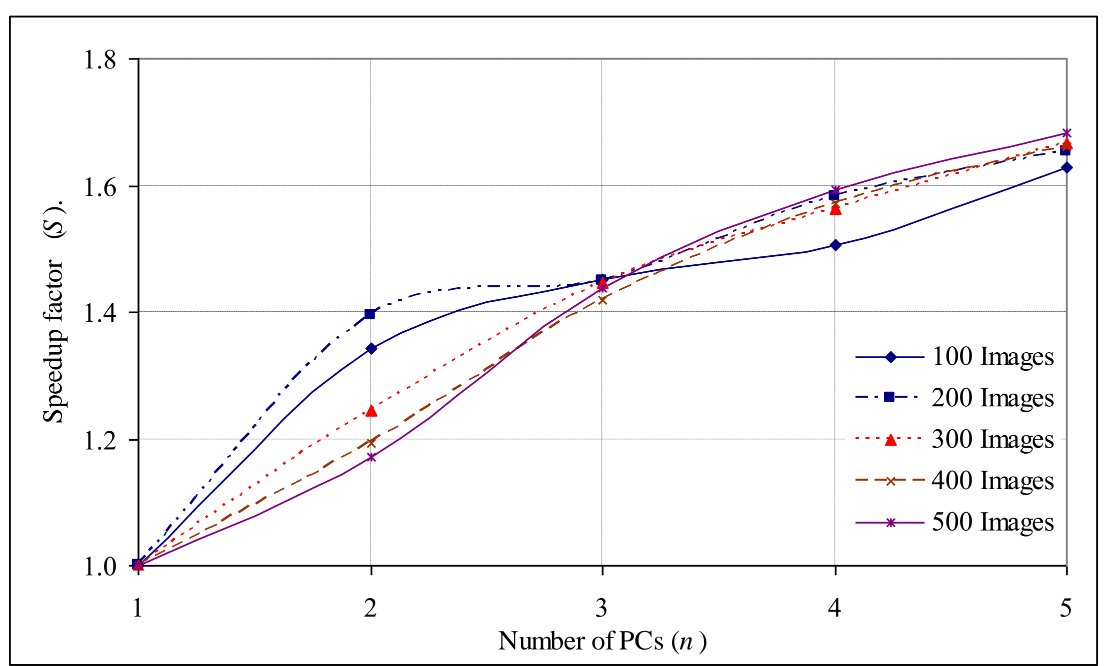


Figure (4.3). Variation of *S* with *n* for various values of *m* and 3x3 kernel size.

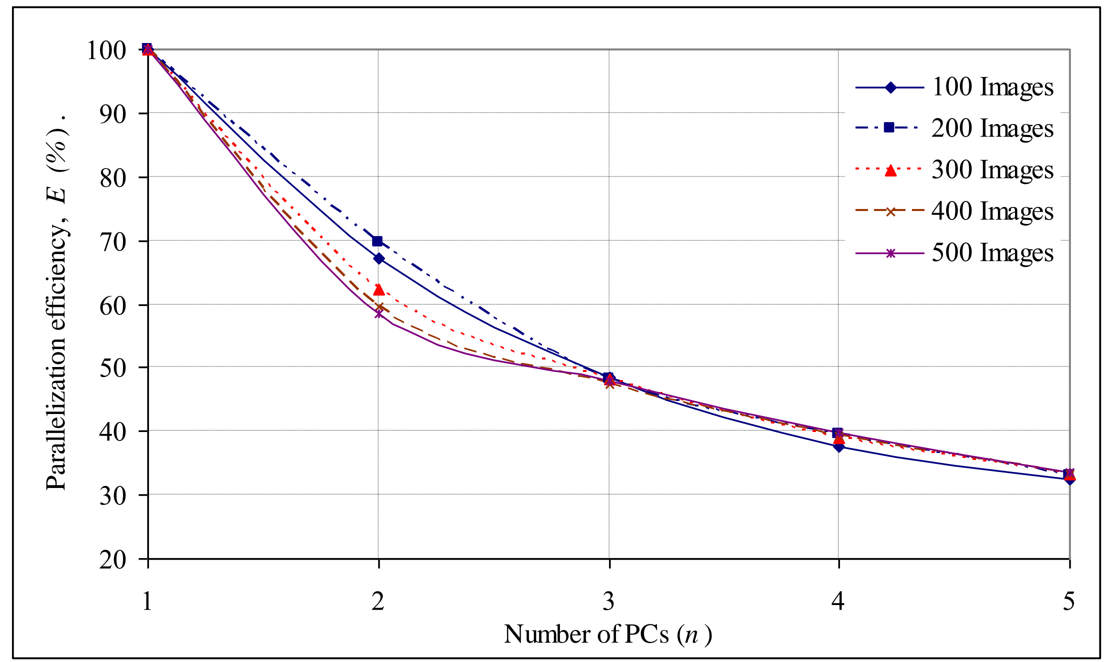


Figure (4.4). Variation of *E* with *n* for various values of *m* and 3x3 kernel size.

(2) Results for 5x5 convolution function kernel size

The following set of runs is equivalent to the above set, except it uses a convolution function of 5x5 kernel size, so that T_{comp} is actually increased without affecting T_{comm}. As it had been shown in Section 4.1 that the

www.manaraa.com

computation time is 0.36 sec/image for 5x5 kernel size. Once again, the computation is performed for various *m*, in practice, up to 1500 images were processed, but the results demonstrate little variation, therefore, we present results for up to 500 images.

Table (4.3)
Comparison of the performance of the DIP model running on a LAN-based system of various number PCs ($n \leq 5$) and images ($m \leq 500$).
(Results for 5x5 convolution function computation)

No. of images	No. of processors (PCs) (n)				
	1	2	3	4	5
CPU time (sec)					
	T_s	T_p			
	T_1	T_2	T_3	T_4	T_5
100	36.6	21.7	16.6	14.2	12.4
200	73.1	42.4	31.9	27.3	24.2
300	109.8	63.1	47.6	40.7	36.2
400	146.0	83.3	63.6	52.7	48.1
500	182.5	104.9	79.3	66.0	59.5
Speedup factor (S)					
100	Serial Computation (S=1)	1.69	2.20	2.58	2.95
200		1.72	2.29	2.68	3.02
300		1.74	2.31	2.70	3.03
400		1.75	2.30	2.77	3.04
500		1.74	2.30	2.77	3.07
Parallelization efficiency (E) (%)					
100	Serial Computation (E=100)	84.3	73.5	64.4	59.0
200		86.2	76.4	66.9	60.4
300		87.0	76.9	67.4	60.7
400		87.6	76.5	69.3	60.7
500		87.0	76.7	69.1	61.3
Image processing rate (□) (Image/sec)					
100	2.73	4.61	6.02	7.04	8.06
200	2.74	4.72	6.27	7.33	8.26
300	2.73	4.75	6.30	7.37	8.29
400	2.74	4.80	6.29	7.59	8.32
500	2.74	4.77	6.31	7.58	8.40

The results obtained for T_s, T_p, S, E, and □, are tabulated in Table (4.3). The following points can be identified:

www.manaraa.com

i. Using 5x5 convolution function provides better accuracy for edge estimation than 3x3 convolution function.

ii. Once again, for the same number of images, *E* is steadily decreasing as *n* increases, but with lower rate when compared with 3x3 convolution function. For example, for *m*=100 images, it decreases from ≈84% for 2 PCs to ≈60% for 5 PCs. This demonstrates that the performance is highly improved when the relative T_{comp} is increased, while T_{comm} is remained unaltered.

   Despite the fact that T_{comp} is relatively high (3 times higher), but as discussed above when *n* increases, the T_{comm} increases to become the dominant factor, so that the efficiency is reduced. However, this time, for a particular number of PCs and images, T_{comp} increases, while T_{comm} remains unchanged giving lower value for *R* and as a result relatively higher *E*.

vi. Unlike 3x3 convolution function computation; because T_{comp} is dominant, for any value of *n*, *E* is increasing with *m*. However, the increasing rate is decreasing as *n* increases, and when *n* is further increased, T_{comm} may become the dominant factor, so that *E* will start decreasing.

   However, this scalability problem is the main drawback of the DIP model or the processor-farm methodology. The optimum number of PCs that can be used within the LAN-based system depends on the value of *R*.

vii. A speedup factor of approximately 3 can be achieved for 5 PCs running concurrently with a parallelization of efficiency of just above 60%. This can be considered as an excellent performance and subject to increase as the relative T_{comp} increases.

Further numerical examples can be deduced from Table (4.3). Figures (4.5) and (4.6) show the variation of *S* and *E* against *n* for various values of *m* for 5x5 convolution function computation.

www.manaraa.com

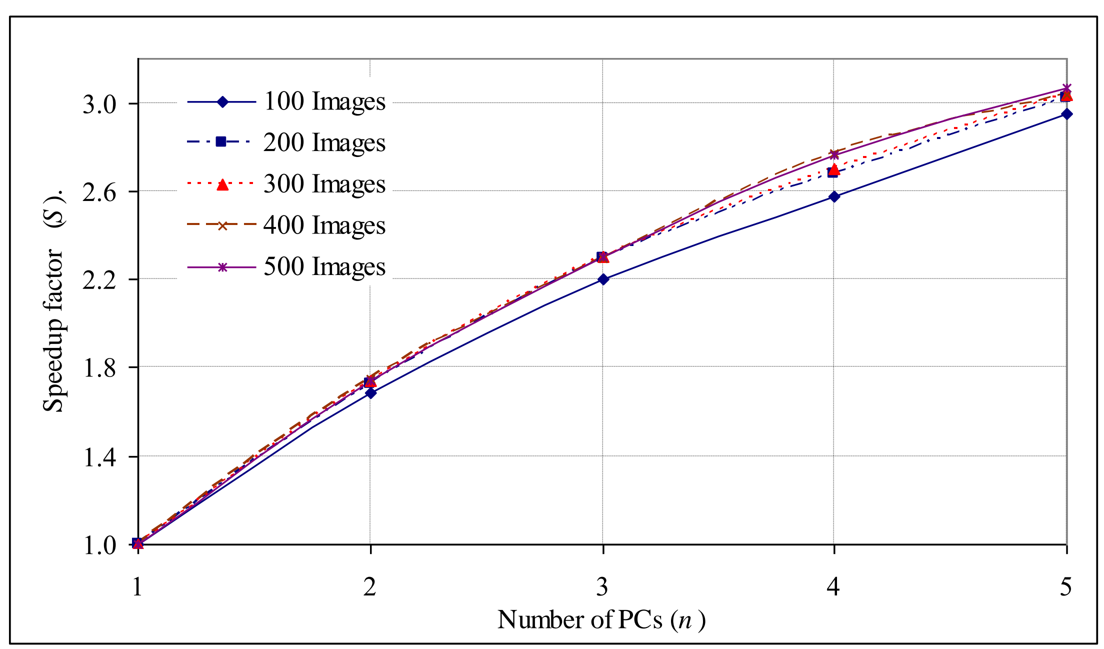


Figure (4.5). Variation of *S* with *n* for various values of *m* and 5x5 kernel size.

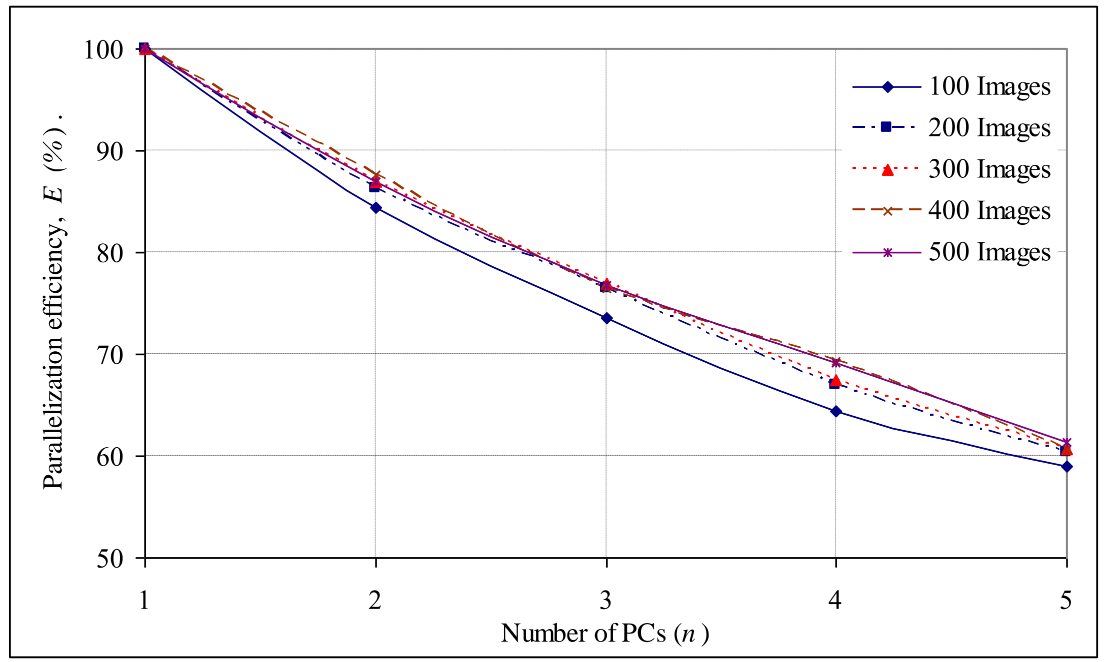


Figure (4.6). Variation of *E* with *n* for various values of *m* and 5x5 kernel size.

4.3. Performance Comparison

In this section, we illustrate crucial conclusions regarding the performance and effectiveness of the DIP model for image processing applications running on LAN-based systems.

www.manaraa.com

A LAN-based system of *n* PCs may provides an excellent performance in terms of *S* and *E*, if the image processing time is high as compared to the image exchange time, regardless of the number of images processed. It was made obvious above that when the average image processing time increases by 3 times (from 0.12 to 0.35 Image/sec), for 5 PCs LAN-based system, *S* increases ≈1.6 to ≈3.0, and *E* is nearly doubled as it increases from 30% to 60%, regardless of the number of images. This means that a higher *S* and *E* can be achieved if the image processing time becomes even higher and higher. Figures (4.7) and (4.8) show the variation of *S* and *E* with *n* for processing 300 and 500 images with 3x3 and 5x5 convolution function kernel sizes. They illustrate that 5x5 kernel size provides a higher performance values in terms of *S* and *E*.

It can be clearly seen from the above discussion that the DIP model demonstrates an excellent performance when implemented to run on the cost-effective LAN-based distributed processing systems for image processing applications, and the performance and effectiveness of the model are improving with increasing the image processing time. This encourages us to do more accurate analysis on each image so that comprehensive image processing applications can be designed with less cost.

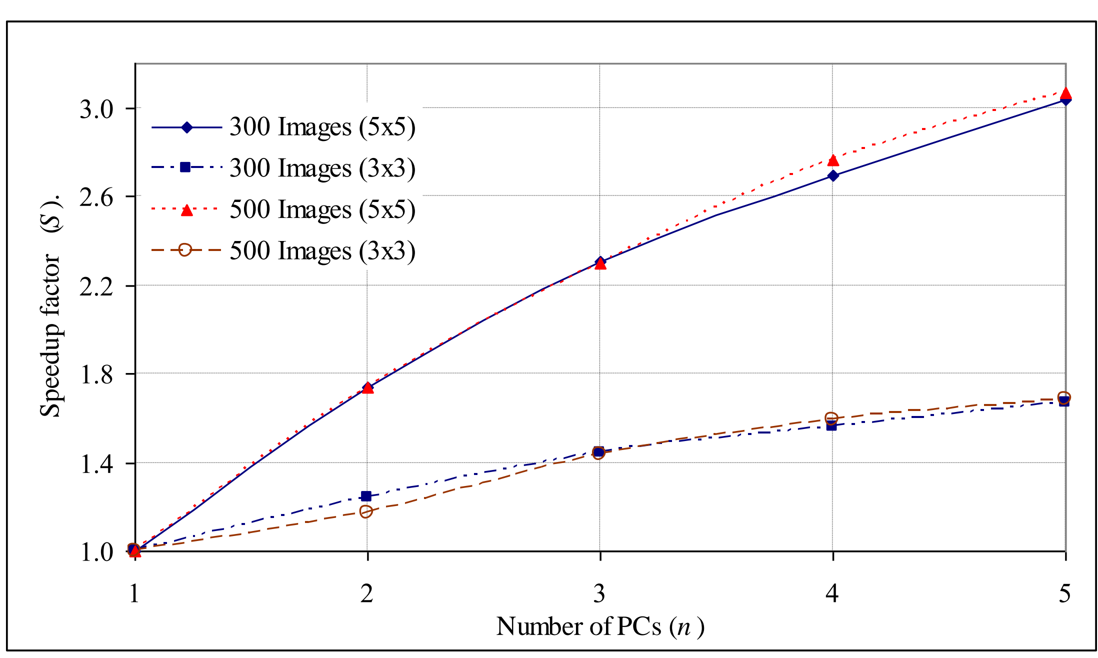


Figure (4.7). Variation of *S* with *n* for various values of *m* and kernel size.

www.manaraa.com

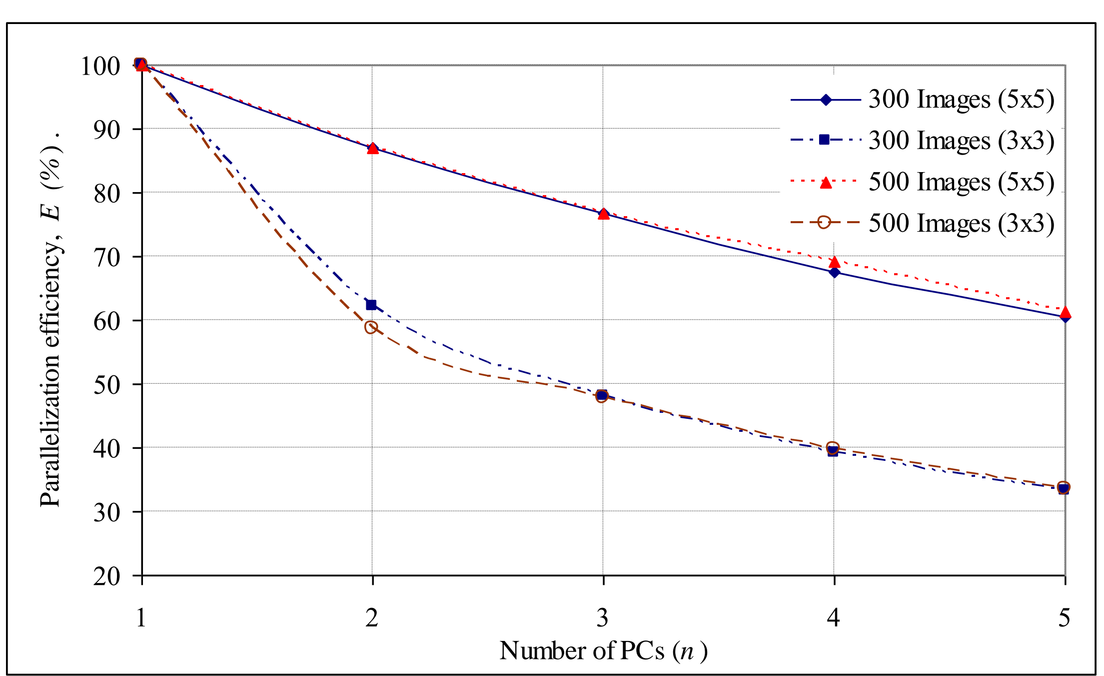


Figure (4.8). Variation of E with n for various values of m and kernel size.

www.manaraa.com

Chapter Five
Conclusions and Recommendations for Future Work

5.1. Conclusions

The main conclusions of this work can be summarizes as follows:

(1) The image processing algorithms demonstrated a satisfactory accuracy, especially with the implementation of the median filter algorithm for noise reduction or removal (image enhancement). But, such image processing application was found to be computationally time consuming application.

(2) A standard Ethernet LAN-based computing system can be successfully used as a cost-effective, efficient, and reliable computing platform to speedup such computations, subject to the development of an efficient parallel (distributed) implementation model.

(3) The proposed distributed image processing (DIP) model that is based on the well-known processor-farm parallel methodology, demonstrated an excellent performance (in terms of efficiently speeding up image processing computations), and it performs even better when the computation resources increase.

(4) The Java Parallel Virtual Machine (JPVM) library is proved to be an efficient and a reliable tool of data communications (message passing) between PCs within a LAN.

(5) The main characteristics of the DIP model are:

a. It is simple and easy to implant and configured according to the application needs. As in this model the same version of the code is running on all slaves, with little variation required for the master.

b. It involves no interprocessor communications during the computation process, apart from that requires for exchanging data/results at the start/end of the computations.

www.manaraa.com

c. It can provide an exceptional load balance across the network.

d. It can handle various numbers of PCs within the LAN without any variation in the running code.

e. It can be easily modified to run on various LAN technologies or data communication libraries for message passing.

(6) The speedup factor (*S*), the parallelization efficiency (*E*), and Image processing rate (□) depend on the following main parameters:

f. The actual number of PCs (*n*) in uses performing computations concurrently.

g. Image computation (processing) time (T_{comp}), which depends on the size of the processed image and the details of the image processing application.

h. Image communication time (T_{comm}), which is a function of the LAN technology and the size of the image.

In addition to the above general conclusions, a number of numerical conclusions can be identified. For example, for the image processing application, parallelization model (DIP model), message passing library (JPVM), size of the image, and the LAN-based system, described in this thesis, the following numerical wrapping up performance values can recognized:

(1) Regarding our application, changing the size of the convolution function from 3x3 to 5x5 increased the image processing time by triple from 0.12 sec/image to 0.36 sec/image.

(2) *S* is directly proportional to *n* ($n \leq 5$). For example, for image processing time of 0.12 sec/image, *S* increased from ≈1.3 to ≈1.6 when *n* increased from 2 to 5 PCs. While for 0.36 sec image processing time, *S* increased from ≈1.7 to ≈3.0 for the same range of *n*.

www.manaraa.com

(3) E is inversely proportional to n ($n \leq 5$). For example, for image processing time of 0.12 sec/image, E decreased from ≈67% to ≈30% when n increased from 2 to 5 PCs. While for 0.36 sec image processing time, E decreased from ≈85% to ≈60% for the same range of n.

(4) This means that tripling the image processing time leads to double speedup and efficiency.

(5) The number of images processed (m) has insignificant effect on the performance.

5.2.Recommendations for Future Work

The main recommendations for future work may include:

(1) In order to provide a comprehensive performance evaluation of LAN-based computing systems implementing the processor-farm-based DIP model, it is important to evaluate the performance under the following conditions:

a. More time consuming image processing applications.

b. Different image size, e.g., 512x512.

c. More number of PCs connected to the LAN.

d. Different network technologies, protocols, and topologies.

(2) Developed DIP models based on the other parallel programming methodologies (algorithmic model or geometric model), evaluate the performance achieved, and for equivalent computations compare the performance of implementing all three methodologies on a similar LAN-based system.

www.manaraa.com

References

[Bal 08] F. Baldacci and P. Desbarats, “**Parallel 3D Split and Merge Segmentation with Oriented Boundary Graph**”, WSCG 2008 Advanced Conference Program, 2008, available at http://wscg.zcu.cz /WSCG2008 /wscg_program .htm, last visited 2008.

[Bau 02] Lewis Baumstark and Linda Wills, "**Exposing Data-Level Parallelism in Sequential Image Processing Algorithms**", Proceedings of the 9th Working Conference on Reverse Engineering (WCRE-02), pp. 245-254, 2002.

[Bev 99] Alessandro Bevilacqua, “A Dynamic Load Balancing Method on A Heterogeneous Cluster of Workstations”, **Journal Informatica**, Vol. 23, pp. 49-56, 1999.

[Bha 00] Haresh S. Bhatt, V. H. Patel and A. K. Aggarwal, “Web Enable Client-Server Model for Development Environment of Distributed Image Processing”, Proceedings of the **First IEEE/ACM International Workshop on Grid Computing**, Vol. 1971, pp. 135–145, 2000.

[Caa 05] Wouter Caarls, Pieter Jonker and Henk Corporaal, "**Skeletons and Asynchronous RPC for Embedded Data and Task Parallel Image Processing**", MVA2005 IAPR, Conference on Machine Vision Application, Tsukuba Science City, pp. 16-18, 2005.

[Bra 01] Thomas Bräunl, "Tutorial in Data Parallel Image Processing", Australian **Journal of Intelligent Information Processing Systems (AJIIPS)**, Vol. 6, No. 3, pp. 164-174, 2001.

www.manaraa.com

[Civ 04] P. Civicioglu and M. Alci, "Edge Detection of Highly Distorted Image Suffering from Impulsive Noise", **International Journal of Electronics and Communications**, Vol. 58, pp. 413-419, 2004.

[Con 05] Jared O'Connell and Peter Caccetta, "**A Parallel Implementation of an Image Processing Algorithm**", CSIRO Mathematics & Information Sciences Private, APAC conference and exhibition on advanced computing, No. 2123, 2005.

[Cle 06] A. Clematis, D. D'Agostino and A. Galizia, "A Parallel IMAGE Processing Server for Distributed Applications", **John von Neumann Institute for Computing, NIC Series**, Vol. 33, pp. 607-614, 2006.

[Fat 04] Hamed Fatemi, Henk Corporaal, Twan Basten, Pieter Jonker and Richard Kleihorst, "**Implementing Face Recognition Using a Parallel Image Processing Environment Based on Algorithmic Skeletons**", Proceedings of the 10th Annual Conference of the Advanced School for Computing and Imaging, pp. 351-357, 2004.

[Fer 98] Adam Ferrari, "JPVM: Network Parallel Computing in Java", **Concurrency- Practice and Experience**, Vol. 10, No. 11-13, pp. 985-992, 1998.

[Fis 94] Bob Fisher, Simon Perkins, Ashley Walker and Erik Wolfart, "**Convolution**", Department of Artificial Intelligence, University of Edinburgh, available at http://www.cee.hw.ac.uk/hipr/html/hipr_top.html , 1994, last visit 2008.

www.manaraa.com

[Fis 03] R. Fisher, S. Perkins, A. Walker and E. Wolfart, “**Sobel Edge Detector**”, available at http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm, 2003, last visit 2008.

[Fly 72] M. J. Flynn, "Some Computer Organizations and Their Effectiveness", **IEEE Transactions on Computing C-21**, No. 9, pp 948-960, 1972.

[Gal 99] J. A. Gallud, J. M. Garcia and J. Garcia-Consuegra, “**Cluster Computing Using MPI and Windows NT to Solve the Processing of Remotely Sensed Imagery**”, Publisher Springer Berlin / Heidelberg , Vol. 1697, pp. 675, 1999.

[Gon 02] Rafael C. Gonzalez and Richard E. Woods, **Digital Image Processing**, Prentice-Hall Inc., 2nd Edition, 2002.

[Gra 02] Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar, **Introduction to Parallel Computing**, PEARSON, Addison Wesley, second edition, 2002.

[Haw 99] K. A. Hawick, H. A. James, J. A. Mathew and P. D. Coddington, “**Java Tools and Technologies for Cluster Computing**”, Distributed & High Performance Computing Group, Department of Computer Science, University of Adelaide, Australia SA 5005, Technical Note DHPC-077, 1999.

[Hey 00] Elisa Heymann, Miquel A. Senar, Emilio Luque and Miron Livny, “Adaptive Scheduling for Master-Worker Applications on the Computational Grid”, Proceedings of the **First IEEE/ACM International Workshop on Grid Computing,** Vol. 1971, pp. 214 – 227.

www.manaraa.com

[Hoc 88] R. Hockney and C. Jesshope, **Parallel Computers 2**, Adam Hilger, Ltd. , Bristol, United Kingdom, 1988.

[Kel 05] H. Kelash, M. Zaki Gamal El_Dein, N.Kamel, "**Agent Distribution Based Systems for Parallel Image Processing**", GVIP 05 Conference, CICC, Cairo, Egypt, 2005.

[Lee 99] Bu-Sung Lee, Yan Gu, Wentong Cai and Alfred Heng, "Performance Evaluation of JPVM", **Journal Parallel Processing Letters**, Vol. 9, No. 3, pp. 401- 410, 1999.

[Mai 06] Raman Maini and J. S. Sohal, "Performance Evaluation of Prewitt Edge Detector for Noisy Images", **GVIP Journal**, Vol. 6, Issue 3, 2006.

[Man 06] A. K. Manjunathachari and K. SatyaPrasad, "Implementation of Image Processing Operations Using Simultaneous Multithreading and Buffer Processing", **GVIP Journal**, Volume 6, Issue 3, 2006.

[Mat 04] James Matthews, "**An Introduction to Noise Processing**", Generation5, available at. http://www.generation5.org/content/2004/noiseIntro.asp, 2004.

[Met 00] Glenn Metherall, "**Local Segmentation of Images**", School of Computer Science and Software Engineering, Monash University, Thesis, available at. http://www.csse.monash.edu.au/hons/projects/2000/Glenn.Metherall, 2000, last visited 2008.

[Nic 02] Cristina Nicolescu and Pieter Jonker, "A Data and Task Parallel Image Processing Environment", **Elsevier Science B.V.,** Parallel Computing, Vol. 28, pp. 945-965, 2002.

www.manaraa.com

[Paz 08] Abel Paz, Antonio Plaza and Soraya Blazquez, " **Parallel Implementation of Target and Anomaly Detection Algorithm for Hyperdpectral Imagery**", IEEE International Geoscience & Remote Sensing Symposium Conference, 2008.

[Pes 04] Dan A. Pescaru and Muguras D. Mocofan, "**An Easy-to-use Distributed Framework for Image Processing**", FACTA UNIVERSITATIS (Nis), Series: Electronics and Energetics, Vol. 17, Issue No. 3, pp. 453- 464, 2004.

[Pla 06] Antonio Plaza, David Valencia, Javier Plaza, Juan Sanchez-Testal, Sergio Munoz and Soraya Blazquez, "**Parallel Implementation of Hyperspectral Image Processing Algorithms**", IEEE International Geoscience and Remote Sensing Symposium, 2006.

[Qiu 02] Zhenge Qiu, ZengBo. Qian, Zhihui Gong and Qing Xu, "**Fast Parallel Image Matching Algorithm on Cluster**", ISPRS, Symposium on Geospatial Theory, Processing and Applications, 2002.

[Rao 06] Daggu Venkateshwar Rao, Shruti Patil, Naveen Anne Babu, and V. Muthukumar, "Implementation and Evaluation of Image Processing Algorithms on Reconfigurable Architecture Using C-Based Hardware Descriptive Languages", **International Journal of Theoretical and Applied Computer Sciences**, Vol. 1, No. 1, pp. 9–34, 2006.

[Rou 05] Mohamed Roushdy, "Comparative Study of Edge Detection Algorithms Applying on the Grayscale Noisy Image Using Morphological Filter", **GVIP Journal**, Volume 6, Issue 4, 2006.

www.manaraa.com

[Sac 03] Nathan Sachs and Jeffrey McGough, “**A Hybrid Process Farm/Work Pool Implementation in a Distributed Environment Using MPI**”, MICS 2003 Proceedings The 36th Annual, Midwest Instruction and Computing Symposium, Mathematics and Computer Science, 2003.

[Sch 07] Frank Schurz and Dietmar Fey, "**A Programmable Parallel Processor Architecture in FPGAs for Image Processing Sensors**", Integrated Design and Process Technology, IDPT-2007, pp. 30-35, 2007.

[Sil 99] Luis Moura E. Silva and Rajkumar Buyya. **Parallel programming models and paradigms.** In R-jkumar Buyya, editor, High Performance Cluster Computing, Vol. 2, Programming and Applications, pp. 427. Prentice Hall PTR, Chap. 1, 1999.

[Ste 06] Gheorghe Stefan, "**Integral Parallel Computation**", Proceedings of the Romanian Academy, Series A, Vol. 7, 2006.

[Wac 05] Alf Wachsmann,”**Parallel Computing: Clustering and Shared Memory**”, Stanford Linear Accelerator Center (SLAC), 2005, available at. http://researchcomp.stanford.edu/hpc/archives/HPCparallel.pdf, last visited 2008.

[Tan 03] Andrew S. Tanenbaum, **Computer Networks**, Pearson Education, Inc., publishing as Prentice Hall PTR, 4th Edition, 2003.

[Wag 97] Alan S. Wagner, Halsur V. Sreekantaswamy and Samuel T. Chanson, “Performance Models for the Processor Farm Paradigm”, **IEEE** TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, Vol. 8, No. 5, 1997.

www.manaraa.com

[Wil 99] G. Wilson, **Parallel Programming for Scientists and Engineers**, MIT Press, Cambridge, MA, 1999.

[Yal 98] Narendar Yalamanchilli and William Cohen, "**Communication Performance of Java based Parallel Virtual Machines**", Department of Electrical and Computer Engineering, University of Alabama in Huntsville, USA, 1998.

المنارة للاستشارات

www.manaraa.com

المنارة للاستشارات

www.manaraa.com