University of KwaZulu-Natal

College of Agriculture, Engineering and Science

Fingerprint Identification Using Distributed Computing

by

Nontokozo P. Khanyile- 207512527
Bsc Honours (Computer Science)

In fulfillment of the academic requirements for the degree of Master of Science in Computer Engineering, School of Engineering, University of KwaZulu-Natal

Supervisor:

Professor Jules-Raymond Tapamo

Co-Supervisor:

Mr Erick Dube

September 2012

Fingerprint Identification Using Distributed Computing

Author: Nontokozo P. Khanyile
Student id: 207512527
Email: pkhanyile@csir.co.za

Abstract

Biometric systems such as face, palm and fingerprint recognition are very computationally expensive. The ever growing biometric database sizes have posed a need for faster search algorithms. High resolution images are expensive to process and slow down less powerful extraction algorithms. There is an apparent need to improve both the signal processing and the searching algorithms. Researchers have continually searched for new ways of improving the recognition algorithms in order to keep up with the high pace of the scientific and information security world. Most such developments, however, are architecture- or hardware-specific and do not port well to other platforms.

This research proposes a cheaper and portable alternative. With the use of the Single Program Multiple Data programming architecture, a distributed fingerprint recognition algorithm is developed and executed on a powerful cluster. The first part in the parallelization of the algorithm is distributing the image enhancement algorithm which comprises of a series of computationally intensive image processing operations. Different processing elements work concurrently on different parts of the same image in order to speed up the processing. The second part of parallelization speeds up searching/matching through a parallel search. A database is partitioned as evenly as possible amongst the available processing nodes which work independently to search their respective partitions. Each processor returns a match with the highest similarity score and the template with the highest score among those returned is returned as match given that the score is above a certain threshold. The system performance with respect to response time is then formalized in a form of a performance model which can be used to predict the performance of a distributed system given network parameters and number of processing nodes.

The proposed algorithm introduces a novel approach to memory distribution of block-wise image processing operations and discusses three different ways to process pixels along the partitioning axes of the distributed images. The distribution and parallelization of the recognition algorithm gains up to as much as 12.5 times performance in matching and 10.2 times in enhancement.

Preface

The research discussed in this dissertation was carried out in the College of Agriculture, Engineering and Science of the University of KwaZulu-Natal, Durban from February 2011 until September 2012 by Nontokozo Portia Khanyile under the supervision of Professor Jules-Raymond Tapamo and co-supervision of Mr Erick Dube. As the candidate's supervisor, I, Jules-Raymond Tapamo, approve the dissertation for submission.

Signed: ..... Date: .....

As the candidate's co-supervisor, I, Erick Dube, approve this dissertation for submission.

Signed: ..... Date: .....

I, Nontokozo Portia Khanyile, hereby declare that all the materials incorporated in this dissertation are my own original work, except where acknowledgement is made by name or in the form of a reference. The work contained herein has not been submitted in any form for any degree or diploma to any institution.

Signed: ..... Date: .....

Declaration 1 - Plagiarism

I, Notonkozo Portia Khanyile, declare that

1. The research reported in this dissertation, except where otherwise indicated, is my original research.
2. This dissertation has not been submitted for any degree or examination at any other university.
3. This dissertation does not contain another person's data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.
4. This dissertation does not contain another person's writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:
    a) Their words have been re-written but the general information attributed to them has been referenced
    b) Where their exact words have been used, then their writing has been placed in italics and inside quotation marks, and referenced.
5. This dissertation does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation and in the References section.

Signed: .....

Declaration 2 - Publications

I, Nontokozo Portia Khanyile, declare that the following publications came out of this dissertation

1. N.P Khanyile, J.-R. Tapamo and E. Dube, *A Comparative Study of Fingerprint Thinning Algorithms*, 10th Annual Information Security for South Africa Conference (ISSA'11), August 15-17, 2011;

2. N.P. Khanyile, J.-R. Tapamo and E. Dube, *Distributed Fingerprint Enhancement on a Multicore Cluster*, The 16th International Conference on Image Processing, Computer Vision, & Pattern Recognition (IPCV'12), July, 2012, Vol 2, pp 890-897; [invited for journal publication]

3. N.P. Khanyile, J.-R. Tapamo and E. Dube, *Performance Prediction Model for Distributed Applications on Multicore Clusters*, Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering 2012, WCE 2012, July, Vol 2, pp 1119-1124;

4. N.P. Khanyile, J.-R. Tapamo and E. Dube, *An Analytic Model for Predicting the Performance of Distributed Applications on Multicore Clusters*, IAENG International Journal of Computer Science, Vol 39, Issue 3, pp 312-320;

I declare that all the work contained in these publications was done solely by me, and has been reproduced in this dissertation.

Signed: .....

Glossary

1. **Minutiae** - Tinny features on a fingerprint pattern caused by discontinuity or forking of the ridges
2. **Bifurcation** - Type of minutiae characterized by a single ridge that forks into two branches
3. **Termination/Ending** - Type of minutiae characterized by a sudden ending/temination of a ridge
4. **AFIS** - Automatic Fingerprint Identification System used to identify individuals automatically using their fingerprint information
5. **Subject** - A human individual who submits a trait to a recognition system
6. **Trait** - Distinguishing characteristic used for biometric recognition
7. **Latent (fingerprint)** - Fingerprint impression left behind on surface contact. Usually used in crime scene investigation
8. **Auxiliary angle** - An angle that the base of the fingerprint image makes with the line joining the core and the center of the fingerprint foreground at the bottom of the fingerprint area
9. **Gradient** - A change in the slope of a straight line
10. **Conjugal slope (C-slope)** - The gradient of the line joining the core and the delta
11. **Graph** - A collection of nodes and edges, where edges represent some form of relation between the nodes
12. **Isomorphism** - Similarity of shape of two graphs
13. **Classification error** - Incorrectly classifying a fingerprint to a ridge class it does not belong to

14. **False match** - Incorrectly confirming a match between two fingerprints which are not the same

15. **False rejection** - Incorrectly rejecting a match between two fingerprints that are the same

16. **Winnowing** - A way of reducing the search space of a template database by partition the database according to some predetermined classes

17. **Rotation** - Alignment of a fingerprint in relation to the vertical axis through the center of the background (acute angle if print rotated anticlockwise, reflex if rotated clockwise)

18. **Translation** - Position of a fingerprint image in relation to the background (whether the print is positioned on the center or shifted to left, right, top or bottom)

Acknowledgements

I would like to thank the Center for High Performance Computing (CHPC) for granting me access to their clusters. This work would not have been successful without their resources. Secondly, I would like to thank my supervisors Prof. J.-R. Tapamo and Mr. E. Dube for their support, patience and endless insight. They made a valuable impact on my work. I would also like to acknowledge the Authentication Research Group of the CSIR Modelling and Digital Sciences, Information Security (MDSIS) for their support and for some of their code that I used. Lastly I want to thank my loving family and friends who provided me with emotional support and endless motivation to keep going and give it my best. I love you guys.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The US National Science and Technology Council (NSTC) "Biometrics Glossary" defines biometrics as *"a general term used to describe a characteristic or a process. As a characteristic [the term biometrics refers to] a measurable biological (anatomical and physiological) and behavioral characteristic that can be used for automated recognition. As a process [biometrics refers to] automated methods of recognizing an individual based on measurable biological (anatomical and physiological) and behavioral characteristics"* [1]. In essence, the term biometrics refers to the automatic recognition of biological and behavioral traits. While biological traits are acquired by or before birth, behavioral traits are learned and acquired overtime. Biological traits include fingerprints, palm prints, iris, vascular patterns, hand geometry, face, retina, DNA and ear shape. Behavioral traits include speech, gait, signature and keystroke. Biometrics have been in use for over thousands of years, before computers were invented. Evidence of biometric use dates back to as early as 8000 years ago where fingerprints were used to authenticate documents in Assyria and Babylonia. Hand ridge patterns from 3000 years ago were discovered in Nova Scotia marking (signing) cave paintings. Fingerprints were used in Persia in the 14th century to authenticate government documents. In the 1880s, Alphonse Bertillon created the Bertillonage for forensic criminal identification. The system measured the length and breadth of the head, the length of the middle finger, left foot, and forearm from elbow to middle fingertip. Eye colour and length of the little finger were also recorded [1]. Since then, there has been major breakthroughs in the history of biometrics. Figure 1.1 shows a timeline of the history of biometrics development since the 1880s.

Today, biometrics has a large user market across the world, mainly with law enforcement and government agencies. In the current age, the security offered by biometrics has become a necessity, with the high rates of terrorist attacks like the 9/11 attacks in the United States and the global pandemic of identity theft. According to data from Marcia Y. Jung, International Biometric Groups "Biometric Market and Industry overview", Africa contributes a share of 6.1% to the global biometric market [1]. Figure 1.2 shows the global biometric market share as depicted by Jung.

The common challenge facing most biometric systems today is the processing overhead that is associated with growing databases and increasing image sizes and resolutions. The

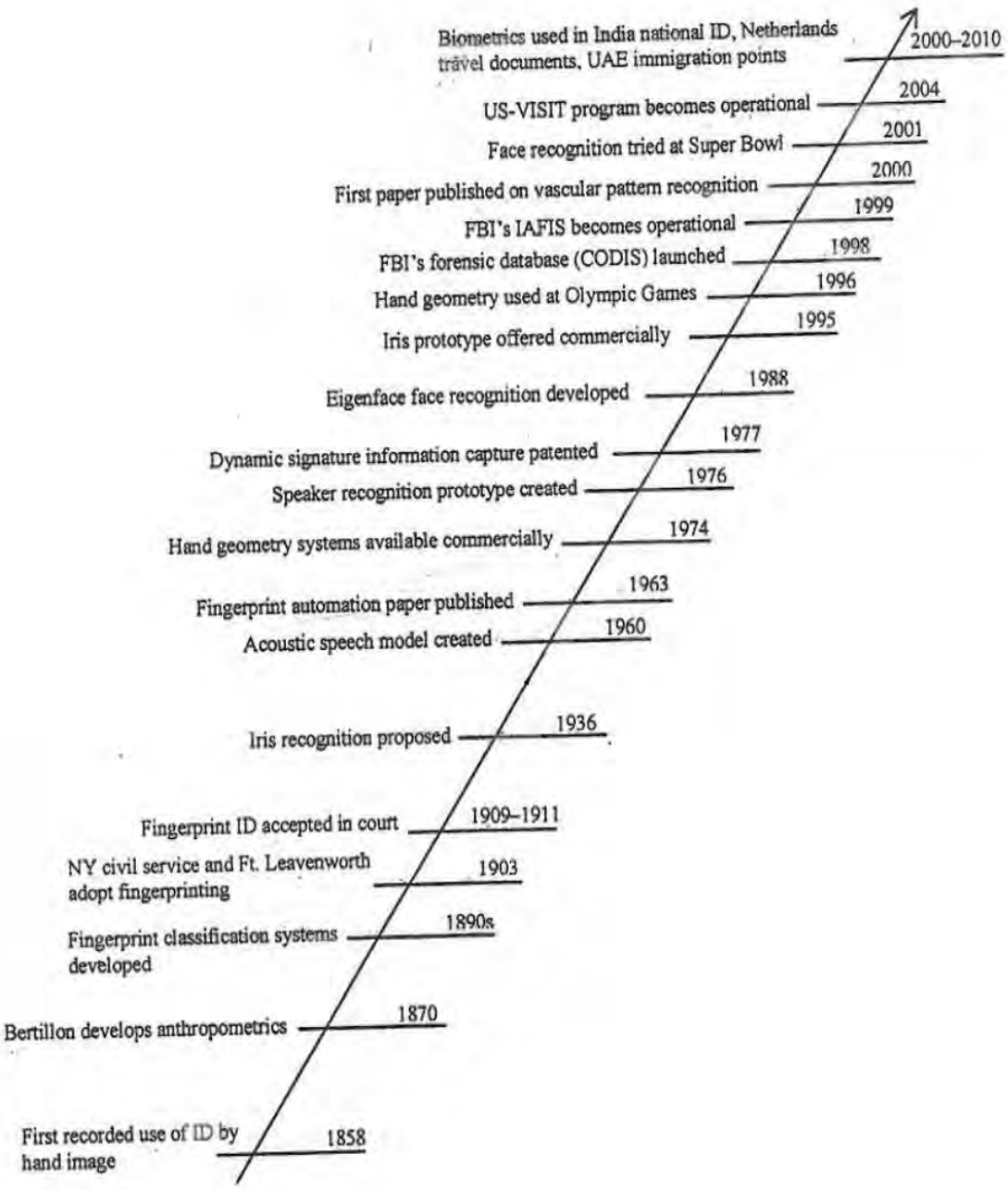

Figure 1.1: Biometric history timeline (taken from [1])

processing of large contents of data is expensive. Most current biometric solutions deployed globally use CPU-based systems [10]. CPU-based systems are considered general purpose machines designed for all types of applications. Therefore, they are sequential processing devices. Instructions are executed by Arithmetic Logic Units (ALUs)[1]. Some processors may contain more than one ALU, in which case multiple instructions can be executed in parallel, however, modern processors are limited in the number of ALUs they posses; most of today's CPUs do not exceed 4 ALUs [10] making their processing capability limited. Research has identified this as a serious problem and as a result, a considerable amount of research has been put into developing faster and more efficient biometric systems [10-20]. Most such developments however, are architecture- or hardware-specific and do not port well to other platforms. This research proposes a cheaper and portable alternative through the utilization of mixed-mode shared memory and distributed memory parallel processing that makes use of multicore clusters.

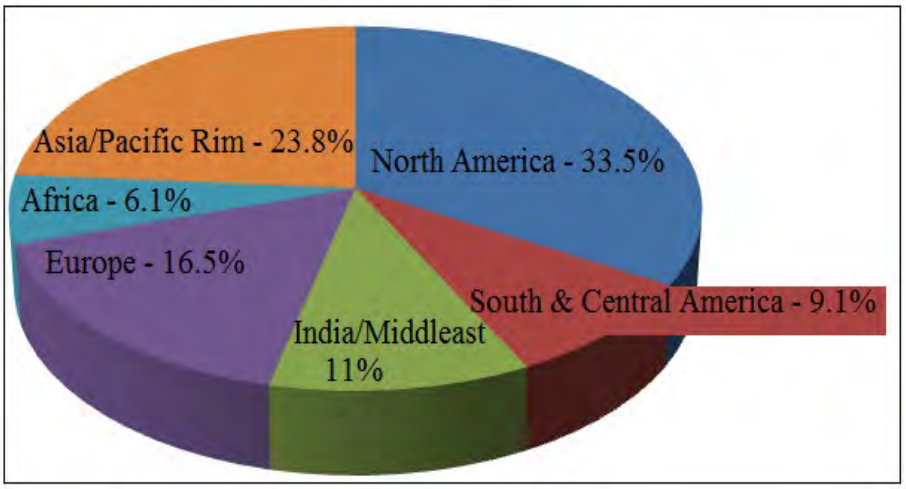

Figure 1.2: Approximate Division of Global Biometric Market by Region (taken from [1])

1.1 Background

Traditional biometric software systems are based on serial computation. These algorithms work by executing a serial stream of instructions. No two instructions may execute at the same time. As one might deduce, this style of programming produces slow software. There has been efforts in literature to parallelize biometric algorithms in order to improve their performance [10–17].

[1] ALU - a digital circuit that performs arithmetic and logical operations

1.1.1 Parallel and Distributed Processing

Parallel programming is a form of programming in which many computations are carried out simultaneously. A large problem is divided into smaller subproblems which are solved concurrently. Distributed programming divides a task into several subtasks which are executed concurrently by different processing nodes. A distributed system is made up of a collection of autonomous computers that communicate through a network. The main difference between parallel and distributed systems lies in the memory usage. Parallel systems are usually referred to as shared memory systems. All processors [2] access shared memory for their I/O operations. The shared memory can be used to pass information between processors. Distributed systems on the other hand use local memory. Each processor reads and writes to its own private memory. Information is passed between processors using a technique known as message passing.

Parallel computers are classified according to the level at which their hardware supports parallelism. Multicore systems are stand alone shared-memory machines which comprise of multiple processing elements for parallel processing. Clusters, Massively Parallel Processors (MPPs) and grids are distributed-memory systems which use multiple computers to complete a task [18].

1.1.2 Message Passing Interface

Message passing is a technique used by distributed systems to share information between the processing nodes. A few message passing paradigms exist including Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). This research uses MPI for its message passing. MPI is a standardized and portable message passing paradigm which is used by most industries. It allows for efficient communication between processors by avoiding memory-to-memory copying, allowing overlaps in computation and communication [19]. MPI provides hardware abstraction, hence code written in MPI is portable and can be run on heterogeneous systems. Processors can only read and write to their local memory. Communication between processes, although crucial, is an expensive operation, as such it must be kept at a minimum [19]. This research uses OpenMPI 1.4.2 which is an open source MPI-2 implementation. Table 1.1 gives some strengths and weaknesses of MPI.

1.1.3 OpenMP

OpenMP is an open specification for multiprocessing. It is a shared memory programming model which is normally used for fine grain parallelization. It is not appropriate for distributed memory environment because it has no message passing capability. OpenMP is recommended when the goal is to achieve modest parallelism on a shared memory computer. The parallelization is explicit allowing the programmer control over parallelization. Table 1.2 shows strengths and weaknesses of OpenMP.

[2] NOTE: Processor in the context of shared memory (fine-grain) parallelization refers to a single core/ALU, while processor in the context of distributed (coarse-grain) parallelization refers to an autonomous system that is part of an overall cluster

Table 1.1: Strengths and weaknesses of MPI [9]

Strengths	Weaknesses
* MPI code runs on both distributed and shared memory architectures	* Decomposition, development and debugging of applications can be a considerable overhead
* Code written in MPI is portable	* Communication can often cause large overhead, which needs to be minimized
* Coarse grain parallelism	* The granularity has to be large, fine grain granularity can create a large quantity of communication
* Large number of vendor optimized MPI libraries exist	* Global operations can be very expensive
* Each process has its own local memory	* Dynamic load balancing is often difficult
* Allows static task scheduling	* Making a transfer between serial and parallel code can be difficult because of significant changes that are required
* Data placement problems are rarely observed	
* Explicit parallelism often provides better performance	
* Communication & computation can be overlapped	

Table 1.2: Strengths and weaknesses of OpenMP [9]

Strengths	Weaknesses
* Fairly portable	* Code only runs on shared memory machines
* Permits both course grain and fine grain parallelism	* Placement policy of data often causes problems
* Each thread sees same global memory, but has its own private memory	* Overhead can become an issue when the size of the parallel loop is too small
* Higher level of abstraction (higher than that of MPI)	* Threads are executed in a non-deterministic order
* OpenMP applications are relatively easy to implement	
* Makes better use of shared-memory architectures	
* Allows run time scheduling	
* Transferring between serial & parallel code is relatively easy	

1.2 Research Problem Statement

Biometric recognition performance is measured using two interrelated factors: the accuracy and the efficiency [20]. Accuracy rates are determined by factors like the size of template database and the types of algorithms used.

The rapid increase in the use of biometrics, along with the advancement on scanning devices has led to slow performing biometric systems. Research in the field of biometrics relies heavily on the effective processing of tera bytes of digital data. For systems to be competent and keep up with overwhelming growth in technology, research on power-efficient and cost-effective computational techniques and platforms is necessary if systems are to meet the demands associated with biometric processing. The quality of a system is highly dependent on the amount of data marshalled to support it [13]. Despite the rapid growth of biometric large-scale databases, the research community is still too far focused only on the accuracy of systems within small databases, neglecting the real issue of scalability and speed on large-scale applications [21]. On the other hand, large high resolution images are expensive to process. Subjects rarely present their traits the same way, as such, there usually is a need to normalize and enhance the input data so that it is noise free and falls within a required region. The signal/image enhancement used involves a series of operations which can be computationally intensive and take time to complete.

The ever growing database sizes cause major overheads during searching. Searching is one of the biggest challenges facing large scale biometric systems. For instance, with the Federated Bureau of Investigation (FBI) Intergraded Automatic Fingerprint Identification System (IAFIS) housing 66 million criminal prints along with 25 million civilian prints, it takes an average of 10 minutes to process a criminal fingerprint [22]. A response time of 10 minutes might be acceptable for forensics, but it is not acceptable for commercial applications. Imagine having to wait for 10 minutes to be granted access to your account.

The need for performance improvement among the biometric community is undeniable. Parallel processing provides a low cost and high performance solution [23]. As a result some algorithms that had been initially designed for sequential systems are being parallelized [16, 17, 24, 25]. Most of these developments however, have low portability due to architecture- or machine-dependency. One way out of the hardware and architecture dependence is through a trend in parallel processing known as High Performance Computing (HPC). HPC provides high computational power which helps process any complex recognition algorithm in order to provide real-time authentication without relying on any particular hardware or architecture. This research investigates a cheaper and portable solution through the utilization of HPC. A mixed-mode distributed and parallel system is proposed that makes use of multicore clusters for performance strength. The algorithm is machine and architecture independent, and can run on heterogenous systems.

1.3 Research Questions

1. *Can the use of a parallel and distributed paradigm vastly improve the performance of biometric identification with respect to time?*

2. *Is it possible to increase identification accuracy through efficient processing?*

3. *Can we then, given the paradigm on Question 1, formalize the performance gain in order to predict system performance of the distributed application before development?*

1.3.1 Thesis Goals

The purpose of this research is to:

1. Parallelize and distribute the fingerprint recognition algorithm and run it on a cluster in order to gain performance. The distribution is targeted at image enhancement and searching, as they are the most time consuming operations. I believe that using a parallel structure paradigm should vastly improve the performance of this modality with respect to computation time. A complex Supercomputer MPI architecture is proposed with multicore processing nodes.

2. From this parallelization/distribution I hope to increase the accuracy of the system by removing winnowing during the searching process. Reason for this is because winnowing introduces tremendous error margins if the classification is not accurate.

3. Lastly, from the distributed system, an attempt is to be made to formalize the performance of distributed applications on multicore clusters. In other words, I want to model performance and ultimately enable performance prediction.

1.4 Delineations and Limitations

1. The main focus of this research is to develop strategies for improving the processing speed of fingerprint recognition through parallel and distributed computation, hence the accuracy of the recognition algorithm is not the *primary* concern of this research, it is only noted as a resulting effect of the optimizations performed during the parallelization process.

2. Only the enhancement and searching algorithms are considered for parallelization and distribution.

3. The algorithms used do not address problems of latent or partial prints.

1.5 Motivation

Generally, the accuracy rates for fingerprint recognition are high. A 2004 study of the US-VISIT [3] fingerprint matching system found the accuracy of the system for two-finger identification to be 95% with a false match rate of 0.08%. In the verification mode, the

[3]US-VISIT - United States Visitor and Immigrant Status Indicator Technology is is a US Homeland Security immigration and border management system which is biometric based

matching rate was 99.5% with a false rate of 0.1% [1]. When it comes to efficiency, however, fingerprint recognition falls amongst the least efficient modalities, together with palm print and face recognition.

The image enhancement and searching subsystems are the most time consuming steps of the fingerprint identification process [26]. Input data is often corrupted by noise and by variations in fingerprint impression conditions such as dry skin. Image enhancement helps remove the effect of these corruptions through a series of image processing operations which make minutiae more visible to facilitate the subsequent feature extraction [27]. The enhancement process consists of the following image processing operations which are often computationally expensive depending on the resolution of the image:

1. Normalization
2. Mask region generation
3. Ridge orientation estimation
4. Ridge frequency estimation
5. Ridge filtering
6. Thinning

Parallelizing and distributing this process provides a significant improvement in the execution time of the algorithm [18]. Searching poses a very big challenge for large scale systems because its efficiency is directly dependent upon the size of the template database. The bigger the database, the longer and less accurate the results will be [28]. While fingerprint verification is fast and has high accuracy, identification is slow and has low accuracy rate compared to verification. The reason for this is that verification mode performs only 1:1 match (i.e. it needs to only verify the claimed identity which the user provides), where as in identification mode, the system requires a vigorous search of the database to perform $1 : N$ matches where N is the size of the database. As database size increases, processing becomes an overhead [26]. Assume a system consists of 100 subjects each with 10 fingerprint captures, cross matching the subject would generate about 17.3 trillion scores. Time taken to process this much information on a general purpose CPU can be quite significant. Research over the years has developed a searching technique known as winnowing. The process of winnowing involves classifying features into different classes based on some degree of similarity [26]. For fingerprint recognition, fingerprint patterns are classified according to their ridge structure type. Ridge patterns can be classified into different classes based on macro features such as arches, loops and whorls [29]. After the classification, templates are grouped according to their classes. This reduces the search space during matching in the following way: after enhancement, a probe is classified according to its ridge type and only the corresponding subset/bin of the database is searched. This reduces the searching time in fractions and reduces false match rate provided the classifications were accurate. The down fall to this technique is that it is only as good as the classification algorithm. In other words,

if the classification is not accurate, winnowing introduces binning errors and produces horrendous results. Winnowing can be avoided through the use of parallel algorithms where multiple processors virtually divide the database and search concurrently different parts of the database [1].

1.6 Contributions

This research makes four main contributions to the field of biometrics, image processing and parallel processing:

1. It presents the distribution process of the entire fingerprint identification procedure (excluding minutiae extraction)
2. It presents a novel approach to memory distribution for block-wise image processing operations in distributed environments
3. Three different ways of dealing with pixels along the partitioning axis of distributed images are presented
4. A new performance model for distributed applications is introduced. This model allows for the prediction of performance of distributed applications running on multicore clusters

1.7 Thesis Overview

This thesis is structured as follows: Chapter one gives the introduction and background. Chapter two discusses related work. Chapter three outlines the serial implementation and its performance. Chapter four discusses the parallel implementation of the fingerprint recognition algorithm. Chapter five discusses application performance of distributed systems. Chapter six gives the experimental results and chapter seven concludes the dissertation and discusses future work.

Chapter 2

Literature Review

This chapter discusses some of the efforts in literature towards the improvement of the recognition algorithms with respect to time.

2.1 Introduction

Biometrics have been in use for over centuries. Despite the maturity and the continuous advancements in this field, it is evident that recognition algorithms still exhibit some limitations which need to be addressed [13]. The rapid advancement and growth in the usage of biometric systems posses a serious problem when it comes to the performance of these systems [1]. The increase in the use of biometrics has drastically increased the sizes of template databases, while the advancement of scanning devices increases the resolution of images thereby increasing the computation requirements. Performance in biometrics is measured by accuracy and efficiency [26]. Accuracy alone is not useful if a system is too slow. For example, DNA is a form of biometric identification with high accuracy, but as an authentication system it is infeasible due to the process required for identification [26]. Conversely, speed alone obviously means nothing if the results are inaccurate.

There has been considerable work in attempting to improve the performance of biometric systems. Research has shown parallel processing as a promising solution to the problem of slow biometric systems. Work on parallelizing the processing of biometric systems includes the use of Field-Programmable Gate Arrays (FPGAs) [10–15], Graphics Processing Units (GPUs) [16], multicore architecture [30], cloud computing [31, 32], mobile agents [33], High Performance Computing (HPC) [34] and using distributed designs [35–37].

2.2 Field-Programmable Gate Arrays

Modalities such as face, palm, finger and speaker recognition are known for not being the most efficient, they demand high computational power [15, 20]. For years researchers have been working towards improving the efficiency and to speed up these systems [10,11,17,38–40]. Performance improvement on biometric systems is targeted at two crucial subsystems:

signal enhancement and searching/matching subsystem. To speed up signal enhancement, recent techniques use FPGAs and GPUs [10, 11, 14, 16, 17].

"FPGAs are complex programmable logic devices that are essentially a 'blank slate' integrated circuit that can be programmed with nearly any parallel logic function" [11]. FPGAs are mostly used to accelerate embedded systems directly using hardware. Although iris recognition is the fastest biometric in the industry [10] because of the smallness of the size of the templates (a European Union technical report cited a 1.7 seconds processing time to search a database of 1 million templates with a 2.2 GHz processor in 2005 [16]), researchers are still skeptical about the performance of this modality on national-sized databases. It is believed that as iris technology becomes widespread and databases grow, matching will require higher speeds [16]. Rakvic et al [10] presented a direct and parallel processing alternative to the sequential iris recognition algorithm using FPGAs. The most time consuming tasks in the modern iris recognition algorithm have been deconstructed and parallelized in order to speed up the resulting system. In particular, portions of iris segmentation, template creation, and template matching were parallelized on an FPGA-based system. Results showed that the implementation on a modest sized FPGA (as opposed to CPU-based system) executed approximately 9.6 times faster for iris segmentation, 324 times for template creation and 19 times for template matching [10].

The fingerprint recognition algorithm comprises of series computationally intensive image processing operations [13, 15, 18, 27]. Improving the efficiency of this modality has opened up a large research area both in academia and industry due to its large user base. Xu et al [11] and Hermanto et al [14] used FPGAs to accelerate the process of skeletonizing a fingerprint image for minutiae extraction, i.e. the thinning process. These research endeavors investigate hardware implementations of the thinning algorithm which exhibits efficient performance. Special architecture was designed to enhance the parallelism of this algorithm. Xu et al's algorithm was implemented on the Xilinx Virtex II Pro developing system with a highly-parallel architecture [11]. While Hermato et al uses the Xilinx Spartan III. The algorithms take as an input a binary fingerprint image, and apply a set of intermediate steps on the input image, to finally output the thinned image. Xu et al managed up to 50 times speedup, while Hermato et al achieved 3 times speedup.

Work by Qin [17] on acceleration of the fingerprint enhancement algorithm uses the Spartan III chip with up 720k bits block RAM and 320k bits distributed RAM running at a speed of 80MHz. The intensive image processing operations of the fingerprint enhancement algorithm were ported to an FPGA device and accelerated by up to 6.79 times from the 6098 ms software implementation to a 898 ms on hardware [17].

Barrenechea et al [41], Yang et al [42], Danese et al [43], Fons et al [13, 15] and Vitabile et al [44] all ported the full fingerprint recognition algorithm to an embedded environment. While work by Barrenechea et al [41] and Yang et al [42] fail to achieve any real-time performance, Vitabile et al's [44] work was able to achieve a recognition processing time of a little under 200 ms on a Virtex-E FPGA device consisting of over 2M system gates. The multiple parallel processors which are instantiated in a large FPGA enable very high acceleration of the execution of the workload by distributing it to the multiple processors. While this solution [44] achieves applaudable performance, the cost associated with such a large FPGA device is not ideal. In order to lower the cost, Fons et al [13, 15] suggest

using a small FPGA which is still able to achieve the same functionality without requiring a large number of systems gates. This is achieved through the use of run-time reconfiguration [13, 15]. Run-time reconfiguration allows for FPGAs to be reconfigured on-the-fly in order to enable the reusage of resources.

Fons et al [15] investigate the applicability of using run-time partially reconfigurable FPGAs on automatic fingerprint recognition. The partial reconfiguration concept enables FPGA devices to adapt their hardware on-the-fly to meet the requirements of the application by reusing resources. The same resources are able to play different roles at different moments which helps maintain low cost as FPGA prices increase with the size of the hardware. The Xilinx Virten FPGA family, in particular Virtex-4, Virtex-5 and Virtex-6 are cited as FPGAs devices that support run-time reconfiguration [15]. This research [15] exploits the run-time reconfiguration computing to implement a software/hardware co-design fingerprint recognition embedded system. The computationally intensive image processing steps are ported to the custom hardware in order to accelerate the processing using custom parallelism and pipeline under a programmable logic device. Tasks which are not considered time critical are left in software [15]. According to their research, the authors [15] believe their reconfiguration controllers to be one of the fastest ever implemented in FPGAs and published. They describe biometric recognition as a "Killer application" for run-time reconfigurable computing in terms of efficiently balancing computational power, functional flexibility and cost. The system was tested on images sized 268×460 pixels and performed 1.04 times better (from 541.100 ms to 521.933 ms) in comparison to the software version executed on a general purpose desktop for enrolment, and 5.35 times better (from 3774.380 ms to 705.025 ms) for authentication. The performance of the same algorithm when ported to an embedded system as-is was given to be 2933.468 ms and 143193.451 ms for enrolment and authentication, respectively. To improve the performance of the embedded system, the partial reconfiguration implementation was developed using a platform with 21504 1-bit flip-flops, 21504 4-input LUTs, 1296kbit RAM blocks and 48 DSP blocks of the Xilinx Virtex-4 running at a bandwidth of 32-bit at 100 MHz. Experiments showed a great improvement up to 5.62 times from the initial 143193.451 ms to 705.025 for enrolment and an improvement of 203.104 times from 2933.468 ms to 521.933 ms for authentication [15].

Progressing further, the authors developed a fully reconfigurable implementation of the fingerprint recognition application for embedded systems this time using a System-on-Programmable-Chip (SoCP) device. The application was executed on a platform composed of a 32-bit ARM922T RISC processor with up to 200 MHz frquencies, and an FPGA with specific VLS1 hardware accelerators running at 48 MHz [13]. The software implementation of the fingerprint recognition algorithm was first ported to the embedded system as-is in order to profile the performance and identify the time-consuming tasks of the algorithm so that they can be accelerated using hardware. Unlike the previous system [15] which was centered around partial reconfiguration, this implementation [13] uses full reconfiguration. This full reconfiguration is achieved in the following way. The application is divided into several FPGA contexts each of which corresponds to full bitstreams that are used to describe the functional content of the FPGA; the Microprossor Unit (MPU) is responsible for transferring the configuration data of the new context to a specific register. Hardware co-processors are then instantiated in one context into the FPGA depending on the size of

the FPGA and the resources needed by each of the co-processors. Depending on the algorithm complexity, the FPGA is reconfigured as many times as required. A new context is loaded each time the FPGA is reconfigured. The algorithm defines 3 contexts. In the first context, which consists of fingerprint acquisition and image enhancement, the FPGA accommodates necessary hardware accelerators to speedup the context tasks. The second context downloads to the FPGA, the hardware accelerators for directional filtering and image binarization. In the last context, the remaining co-processors are utilized for feature extraction, alignment and matching processes on the FPGA [13]. While very useful for reducing the cost of FPGA-based systems, this reconfiguration does, however, come with a penalty; reconfiguration latency. The authors cite a latency of 37.646%, which basically means that for every response time achieved, 37.646% is the overhead introduced by reconfiguration. Experiments showed a 309.41 times performance gain of 955.84 ms from the 295748.05 ms of the software only embedded implementation and an improvement of 3.43 times from the desktop implementation [13].

FPGAs show high potential as the direct access to hardware reduces the power consumption and improves performance, however, they often involve high complexity and high cost [10, 16]. While Fons et al [13, 15] research investigates ways of reducing the cost, it is evident that the overhead encountered is still problematic.

2.3 Graphics Processing Units

GPUs are commonly used to process matrices. They utilize immerse computational power on Video Graphics Adapters (VGAs) to perform computations on matrix structures [16]. This makes them well suited for image enhancement techniques, as images are represented as matrices in computer memory. A utilization of the immense computation power contained within commodity VGAs can be used to effectively attain higher recognition speeds using GPUs [16]. Modern computers can hold up to four video graphics cards which are separately addressable allowing them to work independently of other cards. This physical and logical separation of the GPU gives multiple VGAs inherent parallelism [10].

Broussard et al ported a highly optimized C++ iris template matching algorithm to a video graphics card's GPU using the High Level Shader Language (HLSL) which is part of Microsoft's DirectX 9.0. Their research focused on the field of iris identification to demonstrate the acceleration achievable by using the GPU on the current video graphics [16]. Current iris identification algorithms execute quickly on small database searches. To demonstrate the acceleration gains, template matching in the form of a Hamming distance calculation was performed on large iris template databases. Results showed that a state-of-the-art video card performed template matching 9.5 times faster(24.65 million template comparison per second) than a CPU [16].

Gannon Technologies [45] has developed a fully automatic latent print matching system which makes use of ridge flows rather than minutiae to match fragments of partial prints to templates stored on a database. The algorithm works by transforming an input image into a highly contrast image which is then masked and thinned directly. Matching is then performed using ridge-specific makers to describe and capture the curve shapes of ridges.

Using this type of matching can require up to trillions of calculations. To deal with computation complexity, the system made up of the NVDIA Gemini PCI X3.0 running on a Dell PowerEdge R720 with the FusionIO ioDrive2Dou is used to match the latent prints [45].

GPUs are more flexible than FPGAs in the sense that within the same architecture code can be portable, but this is obviously still limited. FPGAs and GPUs use the Single Instruction Multiple Data (SIMD) architecture. While this architecture can be highly parallelized, it has significant restrictions on the flow of an algorithm. SIMD imposes locksteps on data in order to provide deterministic execution [30]. One other big concern with using FPGAs and GPUs is the hardware/architecture dependency. Such systems have low portability and can not be used on heterogenous systems.

2.4 Multicore

The focus on improving modern computer processing speeds has shifted from increasing CPU clock speeds and moved towards increasing core counts [30]. Unlike CPU speed, increasing the number of core doesn't automatically benefit software. Code modification is necessary to leverage the processing power offered by multicore systems. Multicore systems are shared memory systems, so they work best with shared memory parallel processing. In order to gain the performance improvement, multithreading is necessary. Indrawana et al [30] used a multicore system AMD Triple core running at 2.20GHz with RAM of 4GB to accelerate the feature extraction (generation of image maps, binarization, thinning, minutiae detection and removal of false minutiae) of fingerprint images using data parallelism. Data parallelism (as opposed to task parallelism) produces systems that scale up easily on highly parallel hardware. When an algorithm is data parallel, it contains minimal, if any, shared data which in turn reduces contention and thread-safety issues [30]. Multicore systems are shared memory systems hence it makes sense for Indrawana et al to implement multithreading to parallelize the fingerprint feature extraction algorithm. Each thread is assigned a number of blocks (of data) which it operates on independently of other threads. Tested on three different databases, each with 80 fingerprints, the algorithm ran 57%, 55% and 60% faster on the different databases [30]. While using multicore systems for accelerating biometric systems seems promising, one must bear in mind that general purpose computers have a limited number cores, most being four, this greatly limits the amount of process gain that can be achieved.

2.5 High Performance Computing

One other trend in parallel processing is towards the use of High Performance Computing (HPC). HPC is a parallel processing technology which provides massive processing power of up to thousands of autonomous processing systems connected by networks. This technology is used to solve computational or data-intensive applications which require significant processing power in order to process large amounts of data in a very short period of time [34]. The communication between the nodes of a cluster can be carried out by applications based on the message passing paradigm which uses the explicit programming method,

in which the interaction between processes, data allocation and workloads must be specified by the programmer. Frequently used options for the implementation of parallel applications based on this model are message passing libraries like: PVM, MPI and MPI-2 [23].

The data overload associated biometrics can negatively affect researchers as current state-of-the-art requires researcher to have a heroic level of expertise in systems software to perform large scale experiments [46]. To address this problem Bui et al [46] designed and implemented BXGrid, a data repository and workflow abstraction for biometric research. The system is composed of a relational database, an active storage cluster and a campus[4] computing grid. BXGrid is a collaboration between a systems research group and a biometrics research group at the University of Notre Dame. The computing grid is made up of 500 CPU Condor pool, where each node is also equipped with a chirp fileserver to export each local disk [46]. Megherbi et al [24] proposed a high performance distributed memory & computing algorithm for face recognition via conformal mapping. Much work has been done in the past using object representations that are not unique e.g. curve fitting representation. Megherbi et al's [24] research introduced a new innovative technique for face representation and recognition using HPC. MPI was used to perform face recognition in a distributed environment. Some facial features are grouped together and then processed in parallel in a distributed-network of workstations via MPI over TCP/IP. A master-slave paradigm was used to implement the parallel and distributed algorithm and is based on a distributed-memory approach [24]. Sasaki et al [47] proposed a compact cluster computer, called Ubiquitous Computing Cluster (UCC), which provides a cost-effective prototyping environment for design and test of ubiquitous applications. UCC supports a variety of development tools and libraries for parallel programming including MPI. A fingerprint verification system was implemented as an application example of UCC. The purpose of the case study was to find an efficient implementation of the verification algorithm for typical embedded applications using a combination of phase-based matching algorithm and embedded parallel processing [47, 48]. The verification algorithm was implemented on an eight node cluster system using two UCCs. The algorithm is summaries as follows: first, the master node computes the phase component (capture input image, calculate 2D DFT of the input image and calculate the phase spectrum) of the image while the rest idle and wait for completion of the computation; second, the master node then broadcasts the phase component to all other processing nodes; thirdly, after receiving the phase component, each node computes the cross-phase spectrum, the BLPOC function and the matching score between probe and registered template; fourthly, each node sends back its match score to the master node which then finds the highest and returns it as a match [47]. The parallel implementation of the verification algorithm using distributed processing achieved a speedup factor of up to 6 times for 8 nodes. Although this implementation manages to achieve scalable results, the researchers had poor design choices. Performing master-only operations on parallel systems incurs high overhead due to idle processors [49]. The performance can be improved through better optimization, such as overlapping the communication with computation thereby avoiding idle processors. A parallel AFIS was developed using a cluster by Din et al [34]. The authors [34] perform distributed matching, where a fingerprint database

[4]University of Notre Dame, Department of Computer Science and Engineering

is divided evenly among three computation nodes. Results showed a significant improvement on the searching time as the database grows. At 400000 records, the parallel algorithm gains up to 300% speedup [34].

Some researchers [31,50] have used cloud computing for biometrics. While cloud computing offers great computational power, its security is questionable [50,51], which becomes an issue because the enrolment information of biometric systems is sensitive in nature. Security and privacy is imperative to biometric systems because they hold sensitive personal information.

2.6 Work on search optimization

Storage can be classified into two categories: centralized and decentralized storage. Centralized storage experiences bottlenecks in network speed and capacity, and the increase in storage space increases the probability of false matches exponentially. Distributed (decentralized) storage's redundant design on the other hand increases the speed by maintaining relatively small parts of the database, but introduces new security risks. Multiple servers offer multiple points of possible attacks during data transmission.

Searching is directly linked to the matching process. Matching usually requires an entire database to be searched regardless of whether or not a match has been found [26]. This increases accuracy and reduces false matches by returning a candidate list rather than the first individual over the matching threshold. Searching the entire database does however, mean that for larger databases, searching becomes computationally intensive. To improve searching time most serial algorithms employ some form of classification technique which partitions the database into several classes thereby reducing the search space [26,52]. While parallel searching can be achieved by using GPUs [16], multicore systems and HCP [34]. Performing winnowing techniques is one possible way used to alleviate the computation complexity of searching large databases [21, 29, 53]. Winnowing techniques partition a database into partitions based on some predetermined categories. The second way is using parallel or distributed computing to split up the data into virtual partitions and assign different processors to each partition so that the database can be searched in parallel [16, 26]. Serial search algorithms require template data to be classified, indexed, and database partitioned in some manner [21]. The Henry classification system [54] is a well known and established classification system used to classify fingerprint patterns into different ridge classes such loops, whirls and arches. Serial search algorithm [29, 53, 55] classify the fingerprint images into the Henry system classes and use them to index and partition the database with cited false rejection rate errors ranging from 20% for Hong & Jain [53], 12.4% for Jain et al [29] and 10% for Ratha et al [55]. Mhatre et al [21] perform coarse level classification on template in order to bin templates and prune the database before performing the exhaustive matching on a multimodal system. The database is partitioned using the k-means clustering algorithm. A probe is assigned to a certain partition on the template database and that partition is searched for potential matches. In order to reduce false rejection errors associated with inaccurate binning, the algorithm searches C closest bins. In contrast to Henry classification based systems, Mhatre et al's approach partitions the database to N bins which

need to be specified for the algorithm rather than depending on the number of fingerprint macro feature classes. The goal of winnowing is to reduce the penetration rate of a search algorithm which in turn reduce false accept rates [26]. One major complication associated with winnowing techniques is the strong dependency on classification. If an image is assigned an inaccurate bin, then the matcher spends time searching for a matching template on the wrong partition. One of two things could happen: 1. A wrong match will be returned or 2. A non-match will be returned. Either way, the system error rates increases. Parallel searching on the other hand eliminates the need for bins and ultimately classification. With enough processing power, a database is virtually partitioned into bins equal to the number of processing elements available. And unlike winnowing, ALL partitions are searched concurrently, and each processor returns a match with the highest score from their respective partition, and finally the match with the highest score is selected.

2.7 Conclusion

Although much has been done in the literature, there is still much lacking. Most work towards the improvement of recognition systems seems to be directed towards hardware. While embedded systems are becoming more and more common, it is important for the software approach to be given more attention as well. Hence this research focuses of the improvement of software-based biometrics.

Chapter 3

Serial Implementation

In this chapter, a serial fingerprint recognition algorithm has been implemented using mostly parts of the popular Hong et al recognition algorithm. The algorithm was implemented and tested on the CASIA database.

3.1 Introduction

Fingerprint recognition is the most used biometric technique in the commercial industry and crime forensics [26]. Fingerprints are a highly universal and unique trait. A fingerprint pattern is made up of ridges and valleys. The chaotic way in which these ridge and valley structures are formed makes them unique to each individual, including identical twins. Latent prints are easily and unintentionally left behind on surface contact which makes them well adapted for crime scene forensics. The fingerprint recognition process can be fully automated by Automatic Fingerprint Identification Systems (AFIS). The process includes the following steps:

1. Sensing and Acquisition: A fingerprint is scanned by a scanning device and the image is acquired by the image processing subsystem
2. Enhancement: The acquired fingerprint goes through a series of image processing techniques in order to remove noise and clarify ridges
3. Feature Extraction: Minutiae is extracted from the enhanced image
4. Searching/Matching: Checking the similarity between extracted minutiae and the minutiae of templates in the database
5. Decision: Accept or Reject probe

Figure 3.1 shows the chain of processes that make up automatic fingerprint identification.

This research focuses on the enhancement and matching (searching) subsystems. Sensing devices often produce noisy/corrupted images. Image enhancement helps to remove the effect of these corruptions and make minutiae more visible to facilitate the subsequent

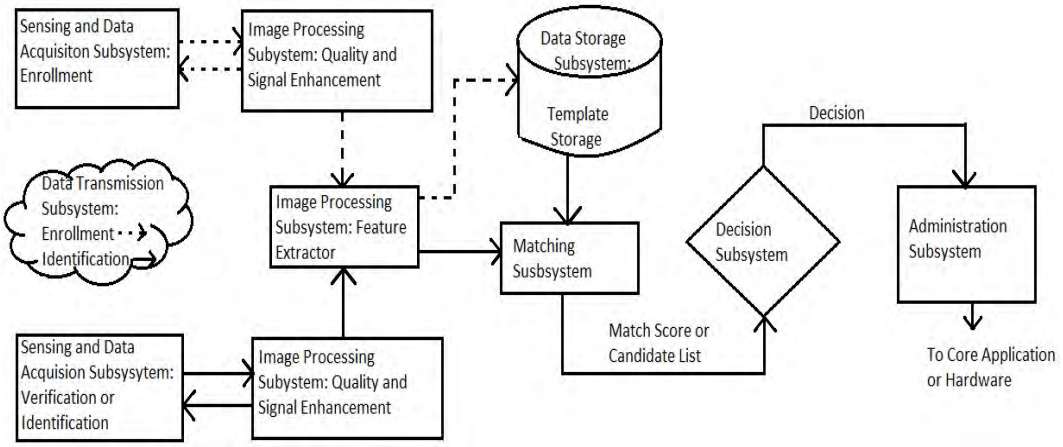

Figure 3.1: Fingerprint identification process (taken from [1])

feature extraction. The enhancement procedure takes in an input image and divides it into two categories - recoverable and unrecoverable regions. The recoverable regions are well-defined or slightly corrupted but visible and can be recovered by using the neighbouring regions to predict their true structure. Unrecoverable regions are corrupted to an extent that the ridge structure is not visible and neighbouring regions do not provide sufficient information to predict their structure [27]. Fingerprint enhancement improves the quality of recoverable regions and removes the unrecoverable regions. The main stages in fingerprint enhancement include normalization, mask region generation, ridge orientation estimation, ridge frequency estimation and ridge filtering [27, 39, 56–59]. Parts of the popular Hong et al [27] algorithm are used for the serial enhancement algorithm presented here.

3.2 Normalization

Normalization is a global operation that adjusts data to fit a certain acceptable region. There often are variations in the way subjects present their traits caused by factors such as dryness of skin, uneven pressure on the scanner, which makes the acquired images to vary. Normalization standardizes the intensity of each pixel to lie within a required range. This process reduces the effects of variations that occur during acquisition and reduces chances of false rejections. For a gray-scale image I, defined as:

$$I = \{(i, j, x_{ij}) | 0 \leq i \leq N-1 \wedge 0 \leq j \leq M-1 \wedge 0 \leq x_{ij} \leq g-1\} \tag{3.1}$$

where (i, j, x_{ij}) is a pixel of the image I, with (i, j) being the position of the pixel and x_{ij} being its gray value. More often the pixel (i, j, x_{ij}) is represented by (i, j), which is adopted in this dissertation. Denote M and VAR as the estimated mean and variance of I, respectively. A normalized image of I can then be expressed as follows:

$$G(i,j) = \begin{cases} M_0 + \sqrt{\frac{VAR_0(I(i,j)-M)^2}{VAR}} & \text{if } I(i,j) > M \\ M_0 - \sqrt{\frac{VAR_0(I(i,j)-M)^2}{VAR}} & \text{otherwise} \end{cases} \tag{3.2}$$

where M_0 and VAR_0 are the desired mean and variance values, respectively.

3.3 Ridge Orientation Estimation

Fingerprint patterns are regarded as oriented texture patterns [27, 57]. An orientation image is made up of directional vectors estimated from a normalized image which represent the orientation of local ridges [56]. A considerable amount of techniques exist which can be used to estimate the local orientation of an image [27, 39, 56–58, 60–62]. The method of averaging square gradients of the gradient covariance matrix seems to be the most widely used approach [27,56,60,61,63,64]. This research uses this approach also adopted by Hong et al to compute gradients of the normalized image. The gradients are then used in the least mean square orientation estimation algorithm.

A normalized image is divided into $W \times W$ non-overlapping blocks then gradients $\sigma_x(i,j)$ and $\sigma_y(i,j)$ are computed for each block using a Sobel or Marr-Hildreth operator. For each block, an orientation vector is derived by averaging all vectors orthogonal to the x and y gradients. The algorithm is summarized as follows:

1. Divide G into $w \times w$ blocks

2. Compute gradients $\sigma_x(i,j)$ and $\sigma_y(i,j)$ at each pixel using Sobel operator. Each pixel of the window $w \times w$ is convolved using horizontal and vertical Sobel kernels s_x and s_y, respectively in order to determine the direction of the maximum intensity change.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

3. Estimate local orientation of each block centered at pixel (i,j) using the following:

$$\gamma_x(i,j) = \sum_{u=i-w/2}^{i+w/2} \sum_{v=j-w/2}^{j+w/2} 2\sigma_x(u,v)\sigma_y(u,v), \tag{3.3}$$

$$\gamma_y(i,j) = \sum_{u=i-w/2}^{i+w/2} \sum_{v=j-w/2}^{j+w/2} (\sigma_x^2(u,v)\sigma_y^2(u,v)), \tag{3.4}$$

$$\theta(i,j) = \frac{1}{2}\tan^{-1}\frac{\gamma_y(i,j)}{\gamma_x(i,j)}, \tag{3.5}$$

where $\theta(i,j)$ is the least square estimate of the local ridge orientation at the block centered at pixel (i,j).

4. To remove noise, a low pass filter is used. In order to use the filter, the orientation image needs to be continuous. The following transformation of the orientation image into a continuous vector field is applied:

$$\Phi_x(i,j) = \cos(2\sigma(i,j)), \tag{3.6}$$

$$\Phi_y(i,j) = \sin(2\sigma(i,j)), \tag{3.7}$$

where Φ_x and Φ_y are the x and y components of the vector field, respectively. The vector field is then used for low-pass filtering as follows:

$$\Phi_x^{'}(i,j) = \sum_{u=-w_\Phi/2}^{w_\Phi/2} sum_{v=-w_\Phi/2}^{w_\Phi/2} W(u,v)\Phi_x(i-uw, j-vw) \tag{3.8}$$

and

$$\Phi_y^{'}(i,j) = \sum_{u=-w_\Phi/2}^{w_\Phi/2} \sum_{v=-w_\Phi/2}^{w_\Phi/2} W(u,v)\Phi_y(i-uw, j-vw) \tag{3.9}$$

where W is a 2D low-pass filter with unit integral and $w_\Phi \times w_\Phi$ is the size of the filter.

5. The local ridge orientation of pixel (i,j) is then computed using

$$O(i,j) = \frac{1}{2}\tan^{-1}\frac{\Phi_y^{'}(i,j)}{\Phi_y^{'}(i,j)} \tag{3.10}$$

The orientation estimation is a prerequisite step for fingerprint filtering, as Gabor filtering relies on accurate local orientation to work correctly.

3.4 Frequency Estimation

Frequency estimation is a block-wise operation which determines the local frequency of ridges along a direction normal to the local ridge orientation [27]. The frequency estimation procedure requires a normalized image which is divided into $W \times W$ blocks. A frequency estimation algorithm used by Hong et al [27] computes oriented windows of size $L \times W$ for each block. These oriented windows are used to compute x-signatures, defined in [56] as the projection of gray-level values from the oriented window to the ridge orientation along an orthogonal direction. The x-signatures of windows without singularities and minutiae form sinusoidal-shape waves with the same frequency as the ridges in the oriented window. Thus the frequency of ridges can be directly estimated from consecutive x-signatures by calculating the distance between their wavelengths [27].

From a normalized image G and an orientation image O, the local ridge frequency is estimated as follows:

1. Divide G into blocks of size $w \times w$

2. For each block centered at (i, j), compute an oriented window of size $l \times w$ that is defined in the ridge coordinate system

3. For each block centered at (i, j), compute the x-signature, $X[0], X[1], ..., X[l-1]$, of the ridges and valleys within the orientated window, where

$$X[k] = \frac{1}{w} \sum_{d=0}^{w-1} G(u, v), k = 0, 1, ..., l-1 \tag{3.11}$$

$$u = i + (d - w/2) \cos O(i, j) + (k - l/2) \sin O(i, j), \tag{3.12}$$

$$v = j + (d - w/2) \sin O(i, j) + (l/2 - k) \cos O(i, j) \tag{3.13}$$

   Let $\tau(i, j)$ be the average number of pixels between two consecutive peaks in the x-signature, then the frequency, $\Omega(i, j)$, is computed as $\Omega(i, j) = \frac{1}{\tau(i,j)}$. If no consecutive peaks can be detected from the x-signature, then the frequency is assigned a value of -1 to differentiate it from the valid frequency values.

4. For invalid frequency values, a frequency value between the range [1/3, 1/25] is estimated.

5. Blocks with minutiae or singular points or corrupted regions do not form well-defined sinusoidal-shaped waves and thus can not be used to estimate the frequency. For such cases, the frequency values are interpolated from frequencies of neighbouring blocks which have well-defined frequencies. This interpolation is performed as follows:

   a) For each block centered at (i, j)

$$\Omega'(i, j) = \begin{cases} \Omega(i, j) & \text{if } \Omega(i, j) \neq -1 \\ \frac{\sum_{u=-w_\Omega/2}^{w_\Omega/2} \sum_{v=-w_\Omega/2}^{w_\Omega/2} W_g(u,v) \mu(\Omega(i-uw, j-vw))}{\sum_{u=-w_\Omega)/2}^{w_\Omega/2} \sum_{v=-w_\Omega/2}^{w_\Omega/2} W_g(u,v) \delta(\Omega(i-uw, j-vw)+1)} & \text{otherwise} \end{cases} \tag{3.14}$$

   where

$$\mu(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{3.15}$$

$$\delta(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.16}$$

   W_g denotes a discrete Gaussian kernel with a mean of 0 and a variance of 9, while $w_\Omega = 7$ is the size of the kernel

   b) If there exists at least one block with the frequency value of -1, then swap Ω and Ω' and go to step (a)

6. Lastly, a low-pass filter is used to remove any possible outliers in Ω':

$$F(i,j) = \sum_{u=-w_\Omega/2}^{w_l/2} \sum_{v=-w_\Omega/2}^{w_l/2} W_l(u,v)\Omega'(i-uw, j-vw) \tag{3.17}$$

W_l is a 2D low-pass filter with unit integral and $w_l = 7$ is the size of the filter

3.5 Mask Region Generation

The mask region generation process segments a normalized image into recoverable and unrecoverable regions per processing block. The mask is then used to separate the Region Of Interest (ROI) from the rest of the image. Hong et al [27] use the amplitude α, frequency Ω and variance v of each block centered at pixel (i,j) to characterize a sinusoidal-shape wave into the two regions. Let $X[1], X[2], ..., X[l]$ be the x-signature of a block centered at (i,j). The three features are computed as follows:

1. $\alpha =$ (average height of the peaks - average depth of the valleys)
2. $\beta = \frac{1}{\tau(i,j)}$, where $\tau(i,j)$ is the average number of pixels between consecutive peaks
3. $v = \frac{1}{l}\sum_{i=1}^{l}[X[i] - (\frac{1}{l}\sum_{i=1}^{l})X[i]]^2$

This process is used to assist determine the quality of a fingerprint image. If the percentage of the recoverable regions is less than a certain threshold, the image is rejected.

3.6 Ridge Filtering

Gabor filters are bandpass filters that have frequency-selective and orientation-selective properties. Thus the success of the filtering stage of fingerprint enhancement relies on the accurate construction of the orientation field and ridge frequency from the previous stages for parameter tuning. As mentioned earlier, fingerprint patterns are essentially oriented texture patterns. This property together with the ability to estimate local ridge frequency makes Gabor filters ideal for fingerprint filtering as the orientation and frequency parameters of Gabor filters can be tuned to match the local ridge orientation and frequency [56].

The Gabor filter used has the general form

$$H(x,y;\phi,f) = exp\left\{-\frac{1}{2}\left[\frac{x_\phi^2}{\delta_x^2} + \frac{y_\phi^2}{\delta_y^2}\right]\right\}\cos(2\pi f x_\phi), \tag{3.18}$$

$$x_\phi = x\cos\phi + y\sin\phi, \tag{3.19}$$

$$y_\phi = -x\sin\phi + y\cos\phi \tag{3.20}$$

where ϕ is the orientation of the Gabor filter, f is the frequency of a sinusoidal plane wave, and δ_x and δ_y are the space constants of the Gaussian envelope along the x and y axis,

respectively. The values for the frequency and orientation images come directly from the frequency and orientation images, respectively. The standard deviation values δ_x and δ_y in this context are set to 4.0 each [27]. The enhanced image E is then defined as follows

$$E(i,j) = \begin{cases} 255 & \text{if } R(i,j) = 0 \\ \sum_{u=-w_g/2}^{w_g/2} \sum_{v=-w_g/2}^{w_g/2} h(u,v : O(i,j), F(i,j)) G(i-u, j-v) & \text{otherwise} \end{cases} \tag{3.21}$$

where R is the recoverable mask ($R = 0$ represents an unrecoverable region), G is the normalized image, F is the frequency image and O is the orientation image. The value for w_g set to 11 specifies the size of the Gabor filters.

3.7 Binarization

Binarization is a pixel-wise operation which converts a gray-scale fingerprint image into a binary image with 1-valued pixels representing ridges and 0-valued pixels representing the valleys. The Gabor filters used to enhance the fingerprint images have DC-balanced waveforms resulting in filtered images with zero mean pixel values [56]. This mean value (zero) is then used as a threshold in order to convert the greyscale filtered image to a binary image. Binarization is achieved through the thresholding technique. The mean value of 0 is used as the global threshold to transform an image E into a binarized image B as follows:

$$B(i,j) = \begin{cases} 1 & \text{if } E(i,j) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

3.8 Thinning

Thinning reduces the binarized images to unit width skeletons. This reduces the amount of data the minutiae extractor has to process and helps to make critical features such as bifurcations, lakes and ridge endings more visible and easier to extract. Thinning is a pixel-wise operation which requires neighbouring pixel values to make a decision about the deletion of a particular pixel. The thinning algorithm used in this paper is adopted from the work presented in [65]. This algorithm modifies the two-subiteration algorithm presented in [66] to preserve connectivity properties and produce thinner skeletons.

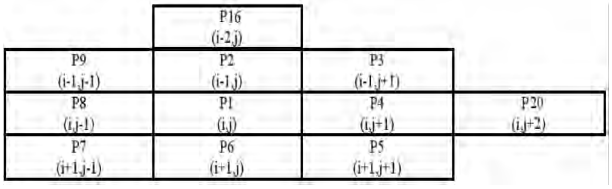

Figure 3.2: Support window used by the thinning algorithm

The algorithm uses operators with the support window shown in Figure 3.3. $C(P1)$ is the number of 8-connected components of 1s in $P1$'s 8-neighbourhood. $T(P1)$ is the number of distinct 0 to 1 transitions and $G(P1)$ is number of 4-connected components. Symbols "$^-$" , "$\wedge$" and "$\vee$" refer to logical complement, AND and OR, respectively; and + refer to arithmetic addition. A function $N(P1)$, defined in (3.23) detects end-points and help produce unitary skeletons [65].

$$N(P1) = min[N_1(P1), N_2(P1)] \tag{3.23}$$

where

$$N_1(P1) = (P9 \vee P2) + (P3 \vee P4) + (P5 \vee P6) + (P7 \vee P8) \tag{3.24}$$

and

$$N_2(P1) = (P2 \vee P3) + (P4 \vee P5) + (P6 \vee P7) + (P8 \vee P9) \tag{3.25}$$

$N_1(P1)$ and $N_2(P1)$ each break the ordered set of $P1$'s neighbour pixels into four pairs of adjoining pixels and count the number of pairs which contain one or two 1s [65].

The function $L(P1)$ is defined as :

$$\begin{aligned} L(P1) = (((\bar{P}2) \wedge P6 \wedge (\bar{P}20)) \wedge (P1 \vee P0 \vee P7) \\ \wedge (P5 \vee P4 \vee P3)) \vee ((\bar{P}0) \wedge P4 \wedge (\bar{P}16)) \end{aligned} \tag{3.26}$$

checks for m-connectivity. The pseudocode of the thinning algorithm is presented in algorithm 1.

Algorithm 1: Procedure for the thinning process that extracts the approximate of the skeleton of a given binary image

Input: Binary filtered image, I
Output: Thinned image, T
Variables: Deletion counter,$DelCounter$; iteration counter, $Iter$;

```
1:  Read I, Initialize DelCounter = 1
2:  while DelCounter ≠ 0 do
3:    DelCounter ← 0
4:    for Iter = 1 : 2 do
5:      Update T
6:      for j = 1 : M, i = 1 : N' do
7:        if pixel p(i, j) = 1 then
8:          if T(P1) = 0 and G(P1) = 0 and C(P1) > 2 and L(P1) = 0 then
9:            p(i, j) ← 0
10:           DelCounter++
11:         end if
12:       else
13:         Read 8-neighbours of p(i, j)
```

```
14:            if Iter = 1 then
15:              if C(P1) = 1; and 2 ≤ N(P1) ≤ 3;
                 and (P2 ∨ P3 ∨ P̄5) ∨ P4 = 0 then
16:                p(i, j) ← 0
17:                DelCounter++
18:              end if
19:            end if
20:            if Iter = 2 then
21:              if C(P1) = 1; and 2 ≤ N(P1) ≤ 3;
                 and (P6 ∨ P7 ∨ P̄9) ∧ P8 = 0 then
22:                p(i, j) ← 0
23:                DelCounter++
24:              end if
25:            end if
26:          end if
27:        end for
28:      end for
29:    end while
```

The algorithm preserves connectivity and produces non-spurious unitary skeletons. $N(P_1)$ is able to detect end-points whether or not they have one or two 8-neighbors. Unlike Zhang-Suen [66], in algorithm 1 2-pixel thick diagonal lines are not deleted.

3.9 Minutiae Extraction

After enhancement, minutiae features are extracted. Minutiae is defined as local discontinuity in the fingerprint patterns [20]. This discontinuity can be in forms of islands, ridge endings, bifurcations, etc. There are over 150 different defined minutiae types in the literature [2]. Figure 3.3 shows examples of minutiae structures.

For practical purposes, only ridge endings and bifurcations are usually used. The minutiae extraction algorithm takes as input, a thinned image I and an orientation image O. To detect minutiae, the method of counting neighbours in a 3×3 block is used. A ridge ending has exactly 1 neighbour, while a bifurcation has exactly 3 neighbours. However, not all pixels with 3 neighbours reflect a bifurcation, for this reason, a lookup table is used to define minutiae from false minutiae [2]. Figure 3.4 shows various scenarios where 3 neighbours encountered form genuine minutiae data.

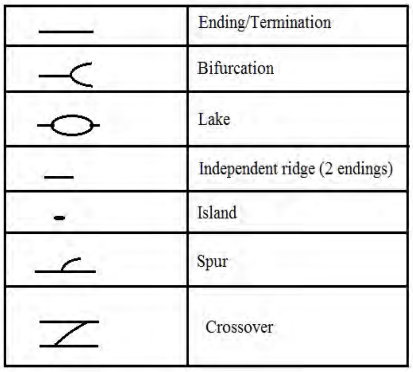

Figure 3.3: Example minutiae structures (redrawn from [2])

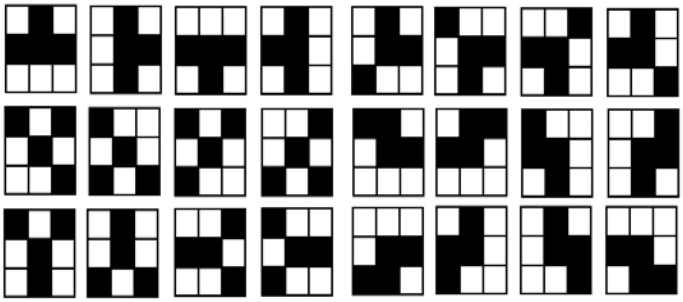

Figure 3.4: Genuine bifurcation minutiae data (adapted from [2])

Minutiae points are represented and stored as 4-tuple sets in the following manner:

$$M = \{m_i | m_i = (x_i, y_i, \phi_i, t_i); x_i, y_i \in \mathbb{N}, 0 \leq \phi_i \leq 360, t_i \in [0,1]\} \tag{3.27}$$

where (x_i, y_i) is the minutiae coordinate, t_i is the minutiae type with 0 and 1 representing ridge ending and bifurcation respectively. The inclination angle ϕ_i is as:

$$\phi_i = \tan^{-1}\left(\frac{y_1^i - y_2^i}{x_1^i - x_2^i}\right) \tag{3.28}$$

where x_1^i, x_2^i, y_1^i and y_2^i are represented in Figure 3.5.

3.10 Classification

Fingerprint patterns consists of global/macro features larger than minutiae which can be used in matching as well as classification. Classification is a process of categorizing fingerprint patterns into different classes using only macro features. These features are identified by using the orientation field curves. The classification process is used to partition the database into classes according to the similarity in the macro features. Two macro features

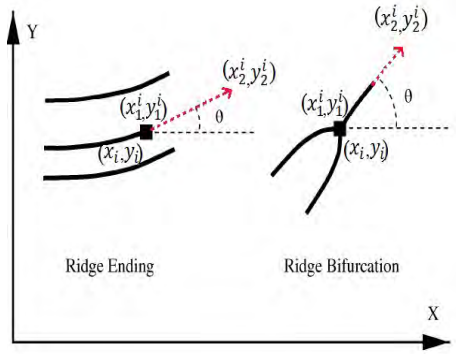

Figure 3.5: Representation of a minutiae point

are defined here: a core and a delta. *" A core is the area around the center of the fingerprint loop and a delta the area where the fingerprint ridges tend to triangulate"* [3]. Figure 3.6 shows an example of a core and a delta.

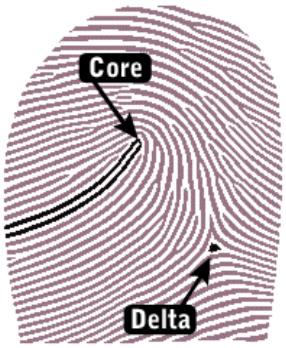

Figure 3.6: A delta and a core (taken from [3])

The Henry classification system defines fingerprint classes: arches, loops and whorls [20].

1. **Arches**
   Arches represent only about 5% of the fingerprint patterns. The ridges that make up an arch flow in one direction without doubling back. They can be sub-dived into plain arch, radial arch, ulnar arch and tented arch.

2. **Whorl**
   Whorls comprise between 25-30% of all fingerprint patterns and are defined by ridges that circle the core(s). Whirls posses two or more deltas.

3. **Loops**
   Loops cover about 65% of all fingerprint patterns. They are characterized by ridges

which loop/turn backwards. They are classified as left, right, radial or ulnar according to the direction of the downward slope.

4. **Composites**
   Composites are patterns which include a combination of the arch, loop and whorl on the same print.

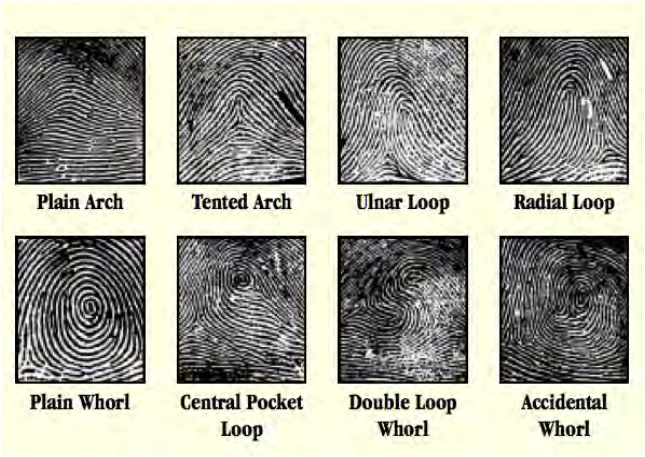

Figure 3.7: Ridge macro classes [4]

The approach proposed in [3] is used for classifying the fingerprints into five classes: Central Twins (CT), Tented Arch (TA), Plain Arch(PA), Left Loop (LL) and Right Loop (RL).

The CT class is made of fingerprints whose ridge patterns form circular patterns along the center of the print. These patterns include whorls and double loops shown in Figure 3.7. The two patterns have one thing in common, they contain two cores and may or may not contain up to two deltas. Ideally, these two patterns should contain two deltas and two cores and hence most classification algorithms use this as a classification rule, but Msiza et al [3] state that this is a great limitation because acquisition conditions are hardly ever ideal and more often than not images do not give full view of the fingerprint, and the delta information may be missing.

The TA class is characterized by ridge patterns that enter the fingertip on one side, making a rise that is at least 45^0 along the center of the fingertip and exit on the opposite side of the finger. TA fingerprints consist of one delta and one core.

The PA class is characterized by ridge patterns that enter the fingertip on one side, make a slight rise (less that 45^0) and exit on the opposite side of the finger. PA fingerprints neither have a core nor a delta.

The LL class consists of ridge patterns that enter the fingertip on the lefthand side of the finger, loop backwards in the middle of the fingertip and exit the fingertip on the same, lefthand side of the fingertip. They contain one core and one delta.

The RL class shares the same properties as the LL class except that the ridges enter and exit on the righthand side of the fingertip.

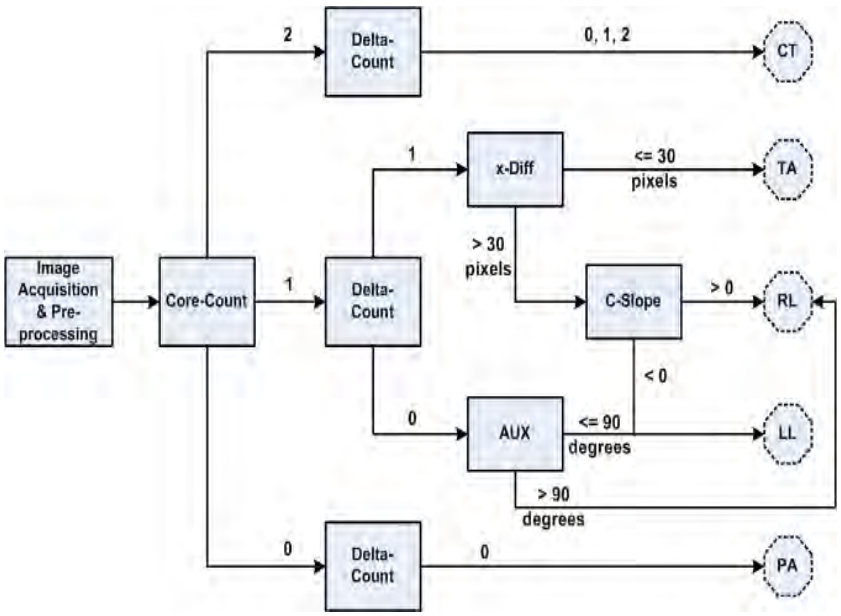

Figure 3.8: Classification procedure (taken from [3])

Figure 3.8 shows the chart diagram of the classification procedure. C-slop is defined as the gradient of the line joining the core and the delta, while the auxiliary angle (AUX) is an angle that the base of the fingerprint image makes with the line joining the core and the center of the fingerprint foreground at the foot (bottom) of the fingerprint area. Finally, x-diff is the distance, in pixels, between the core and the delta along the x-axis [3]. An image first goes through the enhancement procedure after which it is sent to the classifier. The classifier then allocated all present cores from the fingerprint ridge pattern. If none is present, the print is classified as belonging to the PA class. If one core is detected, the classifier proceeds to search for any present deltas. If no delta is detected, the AUX angle is computed; if its greater than 90^0, print is classified as RL, if the AUX angle is greater than or equal to 90^0, the print is classified as LL. If there is one delta present, the classifier computes the x-diff. If x-diff is greater than 30 pixels, and the C-slope is negative, the print is classified as LL. If on the other hand the C-slope is positive, the print is classified as RL. If after having detected one core and one delta, the computed x-diff is smaller than or equal to 30 pixels, the print is classified as TA. If two deltas are detected, the print is classified as belonging to the CT class.

Classification helps in reducing the size of the database to be searched by grouping images with similar features into the same class through a process formally known as winnowing [26]. For this research, the winnowing technique used is binning. The database is logically partitioned into the five classes which reduces the search space.

3.11 Alignment

This research adopts alignment proposed by Msiza et al [5] based on True Fingerprint Center Point (TFCP). The algorithm [5] tracks the TFCP both horizontally and vertically. The x-coordinate is tracked by navigating horizontally through the x-axis, while the y-coordinate is tracked by navigating vertically through the y-axis. The navigation begins at the geometric center (x_b, y_b) of the image:

$$x_b = 0.5 \times w \tag{3.29}$$

$$y_b = 0.5 \times h \tag{3.30}$$

where w is the width of the frame and h is the height of the frame. Reason for this is because, intuitively, subjects will place their finger more or less along the center of the scanner.

Figure 3.9: A fingerprint rotated clockwise (taken from [5])

Algorithm 2: A procedure for locating the TFCP based on the fingerprint mask image [5]

Input: Fingerprint mask image, M
Output: TFCP coordinates, (x_f, y_f)
Constants: W, the width of M; and H, the height of M
Variables: Extreme right x-value (navigation from x_f),x_r; extreme left x-value (navigation from x_f), x_l; extreme right y-value (navigation from y_f), y_u; and extreme left y-value (navigation from x_f), y_l

1: Compute: (x_b, y_b) using (3.29) and (3.30)
2: **if** $M(x_b, y_b) = 1$ **then**
3: Initialize: $x_r = x_b$

4: **while** $M(x_r, y_b) = 1$ && $x_r \leq W$ **do**
5: x_r++
6: **end while**
7: Initialize: $x_l = x_b$
8: **while** $M(x_l, y_b) = 1$ && $x_l \geq 0$ **do**
9: x_l--
10: **end while**
11: Initialize: $y_u = y_b$
12: **while** $M(x_b, y_u) = 1$ && $y_u \geq 0$ **do**
13: y_u--
14: **end while**
15: Initialize: $y_l = y_b$
16: **while** $M(x_b, y_l) = 1$ && $y_l \leq H$ **do**
17: y_l++
18: **end while**
19: Compute: $(x_f = \frac{x_r+x_l}{2}, y_f = \frac{y_u+y_l}{2})$
20: **end if**

Algorithm 2 illustrates a used procedure by [5] to locate the TFCP of a fingerprint image.

Figure 3.10: A fingerprint rotated anti-clockwise (taken from [5])

Algorithm 3: A procedure for determining the alignment direction of a rotated fingerprint [5]

Input: Fingerprint mask image, M; TFCP x-coordinate, x_f; and extreme right y-value

(navigation from TFCP y-coordinate y_f), y_u
Output: Alignment direction, D
Constants: W, the width of M
Variables: x_u;C_1 and C_2

1: Initialize: $x_u = x_f$
2: Start: at point (x_u, y_u), the *upperEdge*
3: **if** $M(x_u, y_u) = 1$ **then**
4: **while** $M(x_u, y_u) = 1$ && $x_u \leq W$ **do**
5: x_u++
6: **end while**
7: Compute: $C_1 = x_u - x_f$
8: Re-initialize: $x_u = x_f$
9: **while** $M(x_u, y_u) = 1$ && $x_u \geq 0$ **do**
10: x_u--
11: **end while**
12: Compute: $C_2 = x_f - x_u$
13: **if** $C_1 < C_2$ **then**
14: D is *clockwise*
15: **else**
16: **if** $C_2 < C_1$ **then**
17: D is *anti-clockwise*
18: **end if**
19: **else**
20: D is *upright*
21: **end if**
22: Re-initialize: $x_u = x_f$
23: **end if**

Algorithm 3 illustrates the procedure used by [5] to determine the alignment direction of a rotated fingerprint. Variables $upperEdge, TFCP, C_1, C_2, C_3, C_4$ and Q used in the algorithms 2,3 and 4 are shown as symbols on Figures 3.9 and 3.10

Algorithm 4: A procedure for determining the alignment angle of a rotated fingerprint [5]

Input: Alignment direction, D;C_1;C_2; TFCP y-coordinate, y_f and extreme right y-value (navigation from y_f), y_u
Output: Alignment/rotation angle, Q
Variables: C_3 and C_4

1: Initialize: $Q = 0^0$
2: **if** D is not *upright* **then**

Compute: $C_4 = y_f - y_u$
if D is not *clockwise* **then**
  Add: an offset, $C_3 = C_1$ to the left of C_2
else
  if D is not *anti-clockwise* **then**
    Add: an offset, $C_3 = C_1$ to the right of C_2
  end if
end if
$Q = \tan^{-1}\left(\frac{C_2+C_3}{C_4}\right)$
end if

Algorithm 4 illustrates the procedure used by [5] to determine the alignment angle of a rotated fingerprint. The alignment procedure correctly aligns the template data with the probe data in order to ensure that the same features sets are matched to each other on the same locations [26]. When performed correctly, alignment reduces errors during the matching stage by ensuring that the same points/same locations are compared by the matcher.

3.12 Matching

Matching is defined by the NSTC "Biometric Glossary" as *" the process of comparing a biometric sample against a previously stored template and scoring the level of similarity (difference or hamming distance). Systems then make decisions based on this score and its relationship (above or below) a predetermined threshold"* [26]. There are four basic approaches for matching; image-based, ridge pattern-based, minutiae pattern-based and graph matching schemes [29]. Matching defines a similarity metric between a probe and a template. A probe is considered a match if the similarity distance is above a certain predetermined threshold [29]. Minutiae pattern-based matching scheme requires registration. Registration involves alignment based on rotation/transaltion.

A probe fingerprint first goes through enhancement, alignment and classification before arriving to the matcher. During the matching/searching stage, an aligned probe minutiae set P is matched to the template minutiae set corresponding to the claimed identity (for verification) or to all contents of the partition/bin corresponding to the ridge class obtained from the template by the classifier. Let and $T = (t_1, t_2, ..., t_n)$ denote the probe minutiae set and the template minutiae set, respectively. The matching procedure is detailed in Algorithm 5.

Algorithm 5: A procedure for matching with winnowing-based serial search

---

Input: Probe minutiae set, $P = (p_1, p_2, ..., p_n)$; Template minutiae set, $T = (t_1, t_2, ..., t_m)$
Output: Similarity score, $S(T, P)$
Variables: $dist\_diff, angular\_diff$
Compute: Euclidean matrix for P and T

$$dist2D_T = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.31}$$

$$dist2D_P = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{3.32}$$

Initialize: $count = 1$
while $count > min(m,n)$ **do**
  if $minutiaeType((p_{count}) == minutiaeType(t_{count}))$ **then**

$$dist\_diff = |dist2D_{t_{count}} - dist2D_{p_{count}}| \tag{3.33}$$

$$angular\_diff = \begin{cases} |\phi_{t_{count}} - \phi_{p_{count}}| & \text{if } |\phi_{t_{count}} - \phi_{p_{count}}| \leq 180^0 \\ 360^0 - |\phi_{t_{count}} - \phi_{p_{count}}| & \text{if } 180^0 \leq |\phi_{t_{count}} - \phi_{p_{count}}| \leq 360^0 \end{cases} \tag{3.34}$$

  end if
  if $angular\_diff < threshold$ **then**

$$S(T,P) = \frac{1}{min(n,m)} \sum_{t_i \in T} min_{p_j \in P} dist\_diff(t_i, p_j) \tag{3.35}$$

  end if
end while

The serial searching algorithm uses the five classes discussed in the classification section. During enrollment, each fingerprint is classified, indexed and placed and the appropriate bin depending on the class. When a probe is submitted for identification it goes through the same procedure and only the corresponding partition/bin is searched.

3.13 Experimental Result Analysis

The fingerprint enhancement algorithm was tested on 60 CASIA fingerprint database images. Figure 3.11 shows the results of the timed experiments, separated into segmentation time, orientation estimation, frequency estimation, filtering, thinning, alignment, minutiae detection and classification.

A winnowed search is compared to a winnowed search at different database sizes of up to 4000 records in Figure 3.12. It can be seen from the diagram that an exhaustive search is not efficient at all, it consumes a lot of time. Where as the winnowed search maintains low search times, which indicate very low penetration rates.

3.14 Conclusion

A serial fingerprint recognition algorithm based on Hong et al's and Mzisa et al's works was developed and tested on the CASIA database. Fingerprint recognition (on the identification

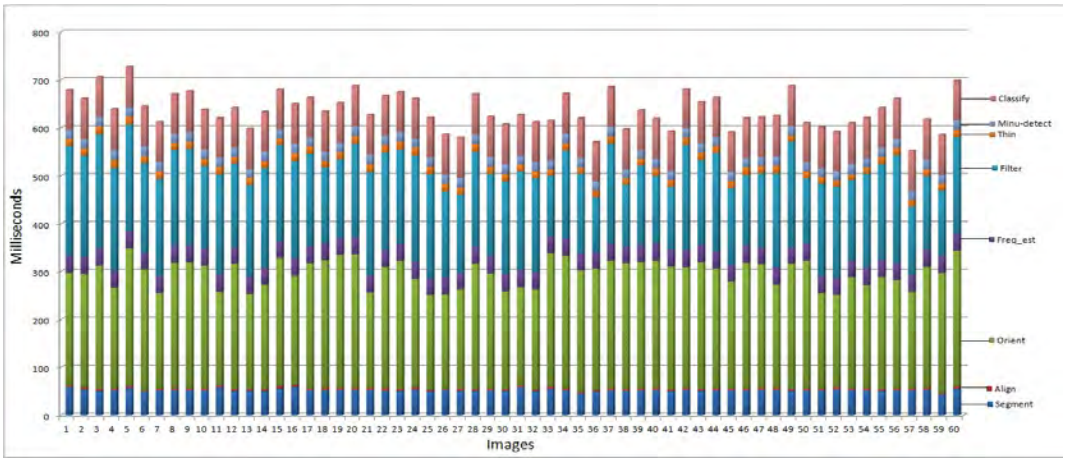

Figure 3.11: Fingerprint enhancement performance on 60 images

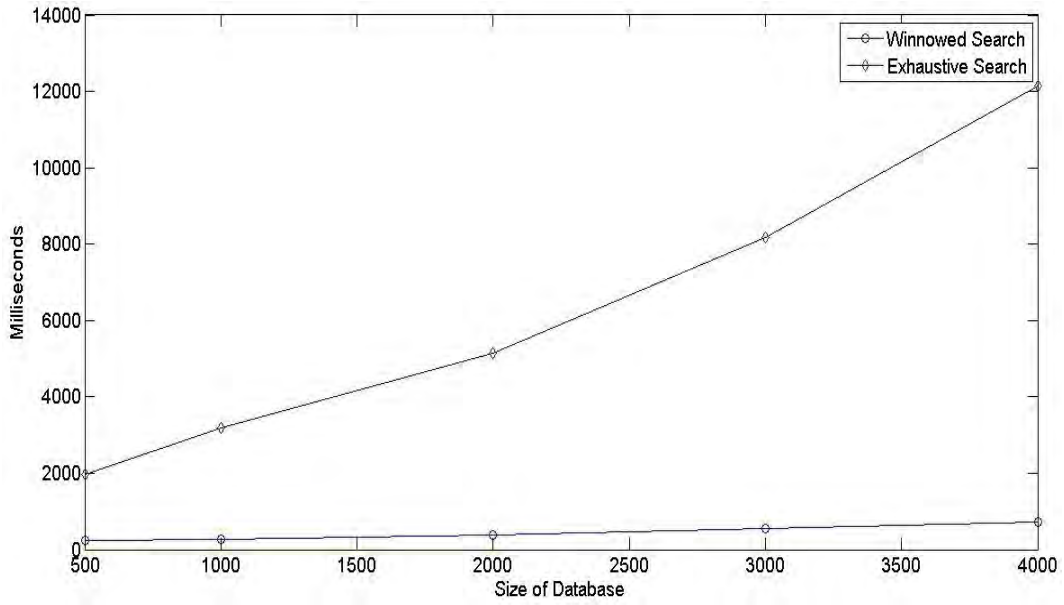

Figure 3.12: Winnowed search vs the exhaustive search

side) is computational intensive and requires a lot of processing power. The response time of the enhancement subsystem (with the highest being 727 ms) is not practical for real-time systems. Search is also ready tedious when performed exhaustively. A winnowing technique known as binning is applied to decrease the searching complexity and ultimately improve efficiency.

Chapter 4

Parallel Implementation

In this chapter, a parallel and distributed fingerprint recognition is presented. The algorithm uses the best parts of Hong et al's and Thai's algorithms and applies neccessary optimizations in order to boost the performance.

4.1 Introduction

This chapter discusses the system architecture and the distribution of the fingerprint recognition algorithm. Mixed-mode parallel and distributed computation is used to take advantage of the multicore cluster. Distributing the computation poses a few design complications. Distributed image processing often requires excessive inter-processor communication in order to access neighbour information to process boundary pixels for pixel-wise operations. Communication (message passing) is associated with large amounts of overhead and this brings forth the task of proper optimization and minimization of communication.

4.2 Architecture Study

There are two types of parallelism: data parallelism and task parallelism [30]. Task parallelism assigns different tasks to different processors which are then required to apply the tasks to the same data. Data parallelism on the other hand partitions the data and assigns it to different processors which then apply the same tasks on the independent sets of data. Data parallelism is generally easier and less error prone in comparison to task parallelism. Task parallelism tends to be error prone because it is unstructured and depends on careful thread coordination to avoid data racing [30]. Data parallelism scales easily to highly parallel hardware because there is very little if any shared data (i.e. data parallel programs tend to be embarrassingly parallel) [30].

The Single Program Multiple Data (SPMD) parallel programming model uses data parallelism. In SPMD algorithms, data is divided as evenly as possible and multiple processing elements execute the same program at independent data points (see Figure 4.1), avoiding locksteps. SPMD programs run on distributed environments and it is the model this research

adopts. A fingerprint image is split up and assigned to different processors which then work independently on the sub-images.

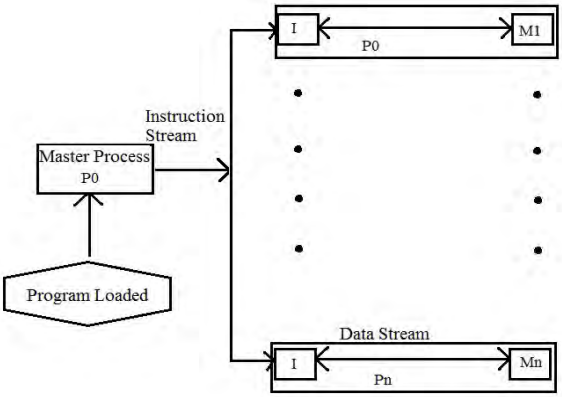

Figure 4.1: Single Program Multiple Data

The system is parallelized in two levels: per-processor and across processors. Per-processor parallelization refers to the fine-grain parallelization achieved through multithreading. Through multithreading, we take advantage of the multicore cluster by using OpenMP. Across processor parallelization refers to the coarse-grain parallelization or distributed computing, where processing is distributed across the processing nodes. The multilevel parallelization shown in Figure 4.2 leverages the cluster architecture by matching the hardware hierarchy. Message passing paradigms such as Message Passing Interface (MPI) and Parallel Virtual Machine (PVM) can be used for coarse parallelization with multiple threads running on each core for fine-grain parallelization.

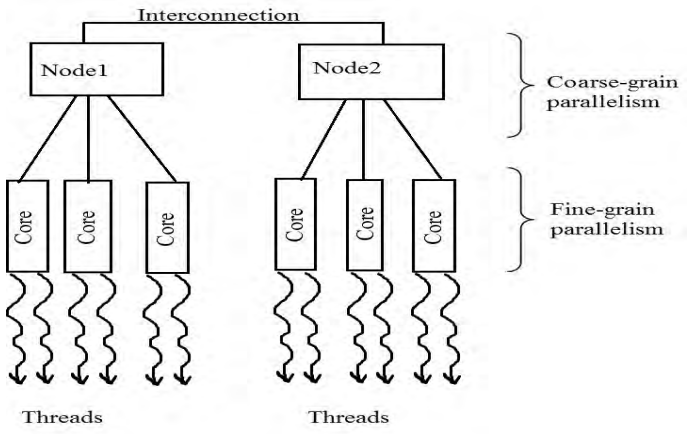

Figure 4.2: Parallelism structure architecture

4.2.1 Data Dependence

The image processing enhancement techniques have strong dependence between pixels. Data dependence can hugely impede the parallelism of an algorithm by making the data path overlap. Figure 4.3 models data dependency graphically.

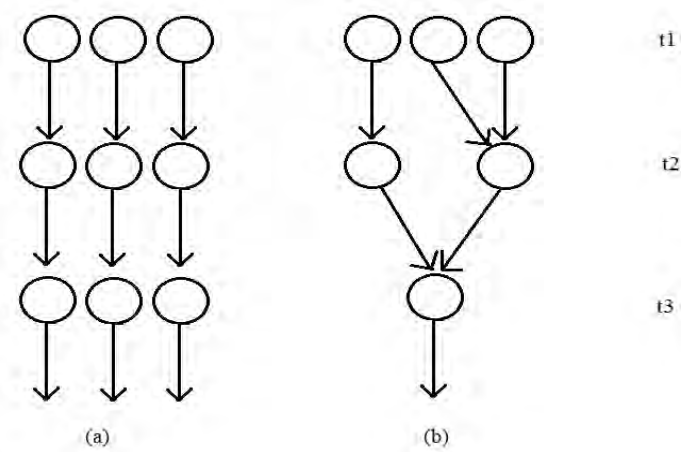

Figure 4.3: (a) Data independence (b) Example of data dependence

When data dependence occurs, parallelism often leads to race conditions which produce erroneous results, deadlocks or potentially trash the entire system. For this reason, not all the algorithms forming part of the recognition process can be parallelized, e.g. minutiae extraction. For algorithms that can be parallelized, we overlap our input data along the boundaries where data dependency can not be satisfied within the processor. Figure 4.4 shows an example of overlapped input data.

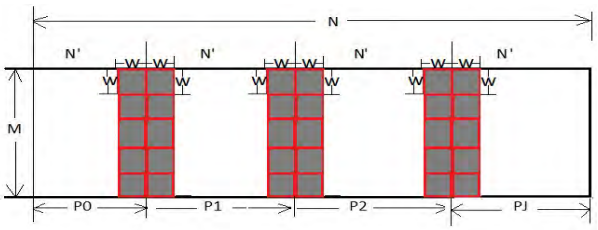

Figure 4.4: Data overlapped along processor boundaries

The overlaps are created through a process known as prefetching. Prefetching not only helps us deal with boundaries, it also helps improve the performance of the system. All data is fetched before it is needed, avoiding idle processors.

4.2.2 Data Layout

In SPMD parallel programs, global data is partitioned and distributed among the processing nodes. There are many approaches that can be taken to partition data for processing by a distributed algorithm. We investigate two types of data partitions: row-wise and column-wise partitions. The choice of data partitioning patterns has a direct impact on the performance of an algorithm. Partitioning an image in a row-wise manner provides contiguous[5] access to data. Each processor can access all its data in a single atomic read operation. An atomic operation is an operation which is indivisible, i.e. results indicate that an entire operation occurred or nothing happened at all. This means that operations can not be interleaved. When an image is partitioned in a column-wise manner, access to data is not contiguous. If each processor is assigned $M \times N'$ subarray of an image, then the distance between two consecutive rows is $N > N'$. It is not possible for a processor to access all its data in a single read operation, as is the case with row-wise partition [6]. Each processor requires M read operations. The same applies for write operations. In image processing operations such as thinning, pixels require access to neighbour information. This becomes a problem when working with SPMD programs. The boundary pixels do not have access to their neighbour information, which can lead to erroneous data.

4.2.3 Distributed Memory Model

Most fingerprint enhancement techniques operate in a block-wise manner as opposed to pixel-wise. This makes distributing the data a slightly more complex. There is a need to ensure that all partitions are in multiples of the processing block size. Let I be an $N \times M$ gray scale image and np be the number of processors. The following derivation shows how the data is distributed when a processing block of size $W \times W$ is used.

The remainder theorem states that if $r, d \in \mathbb{N}$ with $d > 0$, then $\forall a \in \mathbb{R}, \quad \exists q \in \mathbb{N}$ such that:

$$a = qd + r \tag{4.1}$$

where $0 \leq r < d$. Let N' be the number of columns on each processor. We can express $(\frac{N}{np})/W$ in terms of equation (4.1):

$$\frac{N}{np} = qW + r \tag{4.2}$$

The difference between the integer r and window size W represents the number columns that each processor needs to add to $\frac{N}{np}$ in order to have local columns that are a multiple of W. Hence:

$$N' = \frac{N}{np} + (W - r) \tag{4.3}$$

This ensures that the partitions are in multiples of W so that no pixels are left unprocessed. The image is split into np sub-images of size $N' \times M$, each operated on by its respective processors [18].

[5] Assuming row-major representation in memory, e.g. C arrays

4.2.4 Parallel Input/Output

MPI offers parallel I/O, where each processor is responsible for its own I/O operations. Parallel I/O allows computations and I/O operations to overlap. This increases performance as it reduces chances of idle processors which are waiting on I/O operations in order to continue processing.

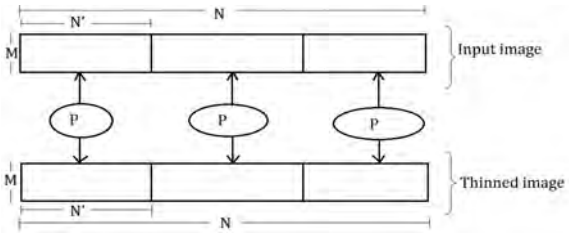

Figure 4.5: Parallel input/output diagram showing how I/O is performed by the algorithm

Figure 4.5 illustrates parallel I/O. The image is split up and assigned to different processors. Each processor works on its own region of the file and writes back the results to a corresponding region on the output image.Using parallel I/O leverages performance of a distributed algorithm. Algorithms with serial I/O where only the master node is responsible for I/O operations suffer from overhead caused by idle processors, because while the master is performing I/O, the rest of the processors are not doing anything. The enhancement algorithm presented here makes use of parallel I/O.

4.2.5 System Configuration

The system used for experiments has the following specifications:

Table 4.1: System Configuration

Model	SuperMicro
Filesystem	GPFS
Network	Gigabit Infiniband
Number of nodes	5
CPU Cores	80
CPU Cores per Node	16
CPU Model	Intel Xeon
CPU Speed	2.4 GHz
Peak Performance	16 TFlops

This is a powerful publicly available cluster. Access to the cluster can be obtained on `www.chpc.ac.za`

4.3 Image Enhancement Algorithm Description

The fingerprint enhancement algorithm takes in a raw gray-scale digital image as input, applies a series of steps and outputs a thinned binary image ready for feature extraction. This algorithm performs the following tasks in sequence:

1. Normalization;
2. Mask region estimation;
3. Orientation field estimation;
4. Ridge frequency estimation;
5. Ridge filtering;
6. Binarization;
7. Ridge thinning;

All these operations are block-wise with an exception of normalization, binarization and thinning which are pixel-wise operations. In order to facilitate pixel-wise operations which depend on neighbour data, two solutions overlap the data across processors, while the third solution depends on message passing. Since the block-wise operations require the sub-images to be in multiples of processing windows, it is necessary to overlap the window containing the boundary pixels for all processors. This is done by modifying the initial sub-image size of $N' \times M$ to add ghost cells of size $W \times M$ which gives new sub-image size of $(N'+W) \times M$. For notation simplicity we assign $N' = N' + W$. Each overlap portion is of size $W \times M$ in order to ensure that the overall sub-image sizes are in multiples of processing block sizes.

4.3.1 Normalization

We employ the technique used by Hong et al [27] to perform normalization. For a gray-scale partial image I_p, we denote M_p as the estimated mean and VAR_p as the estimated variance. A normalized pixel is computed as follows:

$$N_p(i,j) = \begin{cases} M_0 + \sqrt{\frac{VAR_0(I_p(i,j)-M_p)^2}{VAR_p}} & \text{if } I_p(i,j) > M_p \\ M_0 - \sqrt{\frac{VAR_0(I_p(i,j)-M_p)^2}{VAR_p}} & \text{otherwise} \end{cases} \tag{4.4}$$

where M_0 and VAR_0 represent the desired mean and variance, respectively. M_0 and VAR_0 have values 0 and 1, respectively. The reason we give the mean and variance these values is because a Gaussian function has a normal distribution.

4.3.2 Mask Region Estimation

Masking is a process that segments an image into two types of regions: recoverable or unrecoverable region. Segmentation is an image processing technique which separates strong correlated parts of the image into regions [67]. We use thresholding to segment the image into the two regions. Thresholding is a a segmentation process that transforms an input gray scale image to another image with a lesser number of gray level [68]. Most of the time an image is transformed into an image consisting of two gray levels, in this case, it is called binarization. Masking is a process that thresholds an image into two segments that contain either recoverable or unrecoverable regions. Only the standard deviations of the normalized image are used to estimate the mask. Standard deviations are computed for each block, and if they fall below a given threshold the block is assigned to unrecoverable region, otherwise it is assigned to recoverable. From the mask image, the ROI can be constructed by removing all the unrecoverable regions in mask from the normalized image.

4.3.3 Ridge Orientation Estimation

The local orientation of pixels per block must be estimated. Figure 4.6 shows an example of a ridge orientation at pixel (i, j).

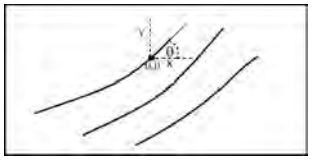

Figure 4.6: Orientation of a ridge at pixel (i, j)

Each processor divides its sub-image into $W \times W$ blocks. Then each block computes the x and y directional gradients $G_x^p(i, j)$ and $G_y^p(i, j)$. To compute these gradients, the first order derivative of a Guassian function is used. The gradient sub-image is given by the vector:

$$\bigtriangledown G^p(i, j) = [G_x^p(i, j), G_y^p(i, j)]^T \tag{4.5}$$

The gradient vectors are estimated using Cartesian coordinates. After which they are converted to polar coordinates in order to obtain double angles and squared lengths.

$$\begin{bmatrix} G_\rho^p \\ G_\phi^p \end{bmatrix} = \begin{bmatrix} \sqrt{(G_x^p)^2 + (G_y^p)^2} \\ \tan^{-1} \frac{G_y^p}{G_x^p} \end{bmatrix} \tag{4.6}$$

with $-\frac{1}{2}\pi \leq G_\phi^p \leq \frac{1}{2}\pi$. When these are obtained, the gradient vectors are squared and expressed in terms of the double angles. Using trigonometric identities, the vectors are expressed as follows:

$$\begin{bmatrix} (G_\rho^p)^2 \cos 2G_\phi^p) \\ (G_\rho^p)^2 \sin 2G_\phi^p) \end{bmatrix} = \begin{bmatrix} (G_\rho^p)^2 (\cos^2 G_\phi^p - \sin^2 G_\phi^p) \\ (G_\rho^p)^2 (2 \sin G_\phi^p \cos G_\phi^p) \end{bmatrix} = \begin{bmatrix} (G_x^p)^2 - (G_y^p)^2 \\ 2G_x^p G_y^p \end{bmatrix} \tag{4.7}$$

The square gradients are then averaged to obtain:

$$\frac{1}{n}\begin{bmatrix}\sum_W[(G_x^p)^2-(G_y^p)^2]\\ \sum_W 2G_x^pG_y^p\end{bmatrix}=\begin{bmatrix}G_{xx}^p-G_{yy}^p\\ 2G_{xy}^p\end{bmatrix} \tag{4.8}$$

This gives the variances and crosscovariance of G_x^p and G_y^p averaged over a block of size $W\times W$. To obtain the orientation field we divide the average square gradients by the absolute values of the squared gradients.

$$\left|\sum_W((G_x^p)^2,(G_y^p)^2)\right|=\sqrt{(G_{xx}^p-G_{yy}^2)^2+(2G_{xy}^p)^2} \tag{4.9}$$

The detailed derivations of some of these equations can be obtained from [61]. The local orientation of the block centered at pixel (i,j) can then be estimated by $(\Phi_x^p(i,j),\Phi_y^p(i,j))$ in the following manner:

$$\Phi_x^p(i,j)=\frac{G_{xx}^p-G_{yy}^p}{\sqrt{(2G_{xy}^p)^2+(G_{xx}^p-G_{yy}^p)^2}} \tag{4.10}$$

$$\Phi_y^p(i,j)=\frac{2G_{xy}^p}{\sqrt{(2G_{xy}^p)^2+(G_{xx}^p-G_{yy}^p)^2}} \tag{4.11}$$

The orientation field is smoothed using a low-pass Gaussian filter to reduce possible effects of noise. The field is filtered as follows:

$$\Phi_x'^p(i,j)=\sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}}\sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}}\Im(u,v)\Phi_x^p(i-uw)(j-vw) \tag{4.12}$$

$$\Phi_y'^p(i,j)=\sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}}\sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}}\Im(u,v)\Phi_y^p(i-uw)(j-vw) \tag{4.13}$$

where $\Im$ denotes a Gaussian low-pass filter of size $w_\Phi\times w_\Phi$. The final orientation sub-image image is given by:

$$O_p(i,j)=\frac{\pi+tan^{-1}(\frac{\Phi_x'^p(i,j)}{\Phi_y'^p(i,j)})}{2} \tag{4.14}$$

4.3.4 Frequency Estimation

We use the approach given by Hong et al to perform local ridge frequency estimation. The x-signature signals form discrete sinusoidal-shape waves which consists of the same frequencies as ridges in the oriented window. This can be used to directly estimate the local ridge frequencies by averaging the number of pixels between the wavelengths, denoted as $\tau(i,j)$. The ridge frequency F_p for a block centered at pixel (i,j) is thus computed as:

$$F_p(i,j)=\frac{1}{\tau(i,j)} \tag{4.15}$$

Corrupted blocks and the ones that contain singularities and minutiae do not form well-defined sinusoidal-shape waves. For such blocks, an estimation for F_p is interpolated from neighbouring blocks [27].

4.3.5 Ridge Filtering

Gabor filters are used to remove noise and preserve the true ridge structure. The following is an even-symmetric Gabor filter given by a cosine wave modulated by Gaussian [56]:

$$H(x,y;\theta,f) = exp\left\{-\frac{1}{2}\left[\frac{x_\theta^2}{\sigma_x^2} \quad + \quad \frac{y_\theta^2}{\sigma_y^2}\right]\right\}\cos(2\pi f x_\theta), \tag{4.16}$$

$$x_\theta = x\cos\theta + y\sin\theta, \tag{4.17}$$

$$y_\theta = -x\sin\theta + y\cos\theta \tag{4.18}$$

where θ is the orientation of Gabor filter, f is the frequency cosine wave, σ_x and σ_y are standard deviations of the Gaussian envelope along the x and y axes, respectively. In order for a Gabor filter to convolute a pixel (i,j) belonging to processor p, it requires the corresponding orientation pixel $O_p(i,j)$ and frequency pixel $F_p(i,j)$ of that pixel. The enhanced pixel, $E_p(i,j)$, is computed as follows:

$$E_p(i,j) = \sum_{u=-\frac{w_x}{2}}^{\frac{w_x}{2}} \sum_{v=-\frac{w_y}{2}}^{\frac{w_y}{2}} G(u,v;O_p(i,j),F_p(i,j))(N_p(i-u,j-v)) \tag{4.19}$$

where G denotes a Gabor filter and N_p denotes the normalized fingerprint sub-image of processor p, and w_x and w_y are the width and height of the Gabor mask, respectively. Hong et al [27] fixed both σ_x and σ_y to 4.0. This becomes problematic when there are variations in the value of a ridge frequency. It can lead to non-uniform enhancement, Thai [56] used the values σ_x and σ_y that are dependent on the ridge frequency parameter.

$$\sigma_x = k_x F_p(i,j), \tag{4.20}$$

$$\sigma_y = k_y F_p(i,j), \tag{4.21}$$

where k_x and k_y are some constant variables.
In order to accommodate Gabor waveforms of different sized bandwidths, in [56] the author set the filter size to depend on standard deviations parameters

$$w_x = 6\sigma_x, \tag{4.22}$$

$$w_y = 6\sigma_y \tag{4.23}$$

where w_x and w_y are the width and height of the Gabor filter mask, and σ_x and σ_y are the standard deviations of the Gaussian envelope along the x and y axis, respectively.

4.3.6 Binarization

Binarization process is identical to that of the serial implementation. The mean value of 0 is used as the global threshold to transform the sub-image E_p into binarized sub-image B_p as follows:

$$B_p(i,j) = \begin{cases} 1 & \text{if } E_p(i,j) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

where B_p is the binary sub-image of processor p and E_p is the filtered sub-image.

4.3.7 Thinning

The thinning algorithm is identical to that used on the serial implementation. I/O occurs only twice during enhancement, when the raw image is first read and at the end of thinning. Since thinning is a pixel-wise operation which relies on neighbour data, extra care has to be taken when performing thinning and I/O. Pixels along the partitioning axis do not have access to their neighbouring pixels and hence can not be correctly processed. There are two ways around this (discussed in the Section 4.6):

1. Inter-processor communication to exchange boundary pixels (known as halo exchange)

2. Overlapping processor data along the boundaries

4.4 Minutiae Extraction

Minutiae points are represented as 3-tuple sets consisting of geometric coordinates and minutiae type. The inclination angle i not considered in order to promote rotation invariance. Mathematically, a minutiae point is represented as:

$$M = \{m_i | m_i \in (x_i, y_i, t_i); x_i, y_i \in \mathbb{N}, t_i \in [0,1]\} \tag{4.25}$$

where (x_i, y_i) is the minutiae coordinate and $t_i = 0$ represents a ridge ending and $t_i = 1$ represents a bifurcation. Removing the inclination angle from the minutiae representation aids in making the minutiae matching algorithm rotational invariant, therefore eliminating the need for alignment and registration.

4.5 Matching/Searching

Through closer inspection of the minutiae on fingerprint patterns, it can the observed that the minutiae points in fact form a graph. Figure 4.7 shows a possible subgraph overlaid on the minutiae points marked on a thinned image.

The graph G depicted on the diagram can be expressed as follows:

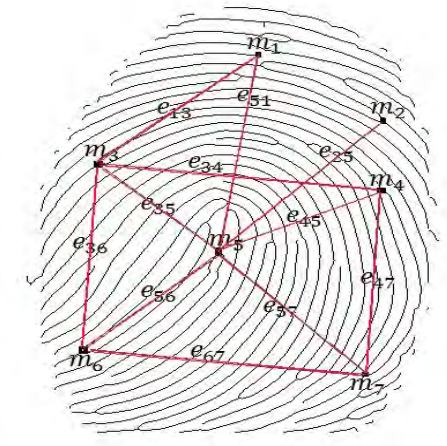

Figure 4.7: Graph pattern formed by minutiae points

$$\begin{aligned} G &= (V_i, E_i) \; where \; , \\ V_i &= \{m_i | m_i \in (x_i, y_i, t_i)\}, \\ E_i &= \{e_{ij} | e_{ij} = dist(m_i, m_j)\} \end{aligned} \tag{4.26}$$

where x_i, y_i are the geometrical coordinates, $t_i \in [0,1]$ is the minutiae type: 0 represents a ridge ending and 1 represents bifurcation and $dist(m_i, m_j)$ is the Euclidean distance between points m_i and m_j. Graph matching can thus be applied to fingerprint minutiae matching.

Graph isomorphism is a very large and established research field [68]. Isomorphism helps to prove that two graphs are structurally the same; i.e. given two graphs, is there a 1-to-1 mapping of vertices between the two graphs that preserves their adjacency [4]? Mathematically, given $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, $G_1 \cong G_2$ (G_1 is isomorphic to G_2) if there exist a function f such that

$$(a,b) \in E_1 \iff (f(a), f(b)) \in E_2 \quad \text{for any } a, b \in V_1 \tag{4.27}$$

Graph isomorphism can be applied to fingerprint minutiae matching [69]. Using graph isomorphism eliminates common problems associated with minutiae matching, including rotation and translation [69]. Each vertex of the graph represents a minutiae point as a 3-tuple set consisting of the x and y coordinates and the minutiae type. The inclination angle is not included on the minutiae representation for matching in order to facilitate in making the algorithm rotational invariant. Each edge connects a minutiae point to other all other minutiae points.

Graph isomorphism is used to match minutiae by connecting each minutiae point of the template to all other minutiae points there by forming a complete graph. A probe is then treated as a subgraph, and subgraph matching is performed on the template. The reason behind matching a subgraph rather than a full graph is because it can not be guaranteed that the same number of minutiae points will be detected from the probe as was detected on the

template. If the number of minutiae points obtained from the probe is greater than that of the template, then the template is treated as a subgraph and matching is performed on the probe. Matching can still be performed if some points are missing as subgraph matching visit each and every node on the larger graph and compares it with the subgraph nodes. If $G_t = \{V_t, E_t\}$ and $G_p = \{V_p, E_p\}$ are graph representations of the template and the probe respectively, the matching then becomes the task of finding a function F, that transforms X into F(X), where X is a set of at least 16 vertices (minutiae) of the probe, and forms a subgraph $G_{p,F(X)} \subseteq G_p$, such that

$$G_{p,F(X)} \cong G_{t,S}, \tag{4.28}$$

with $G_{S,t} \subseteq G_t$. Hence the similarity score is simply the number of matched vertices[6].

Parallel search mimics the phenomenon of distributed (decentralized) storage in that the database is divided into virtual sub-databases or partitions equal to the number of computing nodes. Each node works completely independent on its own partition and returns the candidate with the highest similarity/match score based on the above algorithm. While centralized storage does not scale up well (false match rates increase exponentially), when using parallel search on centralized databases, it eliminates this concern.

4.6 Dealing With Boundary Pixels

Distributed image processing algorithms are required to provide a way in which the boundary pixels can access their neighbouring processors boundary data as it becomes impossible to process a pixel without knowledge of its neighbouring pixels.

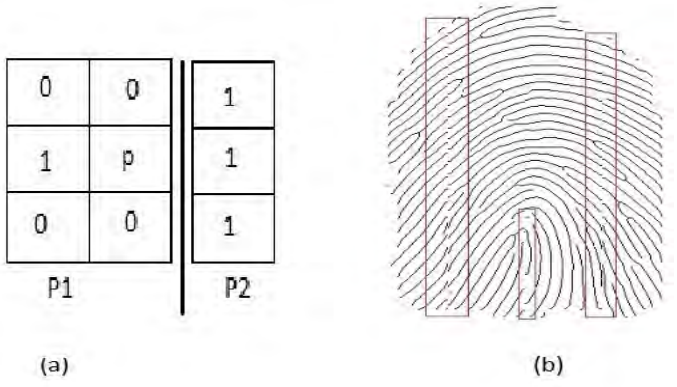

Figure 4.8: (a) Boundary pixels between P1 and P2 (b) Example of an image processed disregarding neighbour data

[6]NOTE: Graph matching is outside the scope of this research. For the detailed graph isomorphism algorithm employed please consult the VFLib2 library available at `http://www.cs.sunysb.edu/~algorith/implement/vflib/implement.shtml` and [70].

Figure 4.8 shows an example of a connected component being partitioned. Boundary pixel p according to processor P1 has a total value of one 1-valued neighbours, which by definition of the thinning algorithm entails that p is a ridge ending pixel. This means p will be treated as an endpoint and preserved even though when we look at P2 we see it is in fact not a ridge ending.
Distributed algorithms need to provide processors with a way to access their neighbouring processors' boundary data. When a partition divides a component into two parts, the contours describing these components will no longer be closed [71]. Kwok [71] defined different contour configurations at the borders of the sections by using chain code representations for border pixels. The configurations help preserve connectivity of components along the borders. In order to access neighbouring processor boundary data, a communication channel may need to be established between the processors which is often expensive. As an alternative, processors may be allowed overlapping access to boundary data. The image is partitioned into subarrays which are then assigned to processing nodes. In order for boundary pixels to access their neighbouring pixels, processors need to overlap data along boundaries or use message passing to share data. The overlapped areas of subarrays shown in Figure 4.9 are usually referred to as ghost cells.

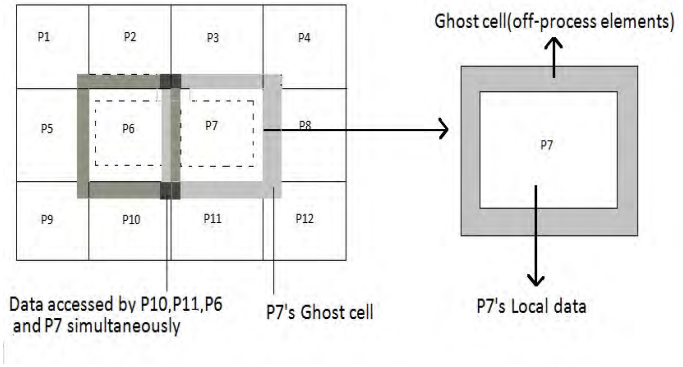

Figure 4.9: Data overlap among processors. The shaded areas represent the overlapped regions

Whenever more than one processor access the same file region for a read and write operation, we often run into nasty racing conditions. Data race occurs when a processor writes to a file region that has not yet been read by all processors that are required to read before any write, or when processors interleave their write operations. This can lead to disastrous results. It is for this reason that MPI requires a developer to enforce operational atomicity.
Consider a scenario shown in Figure 4.10 where column-wise overlapping data is being written to a file. If the write operation is not atomic, it might be interleaved and since processors arrive at their write operations in any arbitrary order, it is impossible to know before hand which processor will write after which. This could result in a final solution that does not reflect the actual computation configuration.

The next three subsections present some solutions to dealing with boundary pixels.

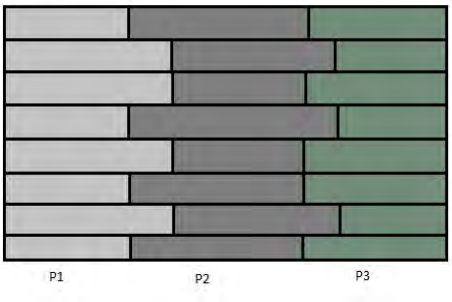

Figure 4.10: Interleaved write operation (adapted from [6])

4.6.1 Halo Exchange

In order to process boundary pixels, neighbouring processors establish communication along the boundaries. A processor is allocated extra memory along the boundaries known as ghost cells. These are used to store data from neighbouring processors, which is used only for computational purposes.

Performing a halo exchange consists of processing nodes sharing their data with their neighbouring processors through message passing. Unless architecture specifically optimizes for it, halo exchange is quite an expensive operation which is unscalable. Increasing the number of processors increases the number of communication nodes. Communication often incurs a large overhead which needs to be minimized. Direct memory access is easier to use than message passing. With message passing, processors must agree to communicate. Each processor must first send its buffers and then wait for the corresponding buffers to arrive from neighbouring processors [72].

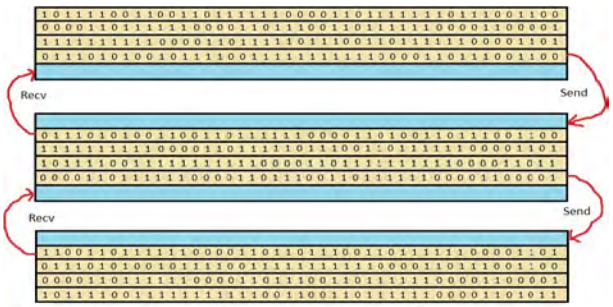

Figure 4.11: Halo exchange between three processes

Figure 4.11 shows a diagram of 3 processors engaging in a halo exchange. Each processor maintains a ghost cell which will accommodate its neighbours' boundary subarrays. The ghost cells are updated using MPI send/recv prior to any processing. The code fragment shown in Figure 4.12 shows how MPI is used to perform the halo exchange on a row-wise partition.

```
1. for(rank==0; rank<np; rank++){
2.       if(rank==0){
3.                MPI_Send(boundaryArr, boundaryArr.length,
4.                                MPI_INT, 1, tag, MPI_COMM_WORLD);
5.                MPI_Recv(ghostArr, ghostArr.length,
6.                                MPI_INT, 1, tag, MPI_COMM_WORLD, &status);
7.                }
8.       else if(rank==np-1){
9.                MPI_Send(boundaryArr, boundaryArr .length,
10                               MPI_INT, rank-1, tag, MPI_COMM_WORLD);
12.               MPI_Recv(ghostArr, ghostArr.length,
13                               MPI_INT, rank-1, tag, MPI_COMM_WORLD, &status);
14.               }
15.      else {
16.               MPI_Send(boundaryArr1, boundaryArr1.length,
17.                               MPI_INT, rank-1, tag, MPI_COMM_WORLD);
18.               MPI_Recv(ghostArr1, ghostArr1.length,
19.                               MPI_INT, rank-1, tag, MPI_COMM_WORLD, &status);
20.                MPI_Send(boundaryArr2, boundaryArr2.length,
21.                              MPI_INT, rank+1, tag, MPI_COMM_WORLD);
22.               MPI_Recv(ghostArr2, ghostArr2.length,
23.                              MPI_INT, rank+1, tag, MPI_COMM_WORLD, &status);
24.                }
25.       }
```

Figure 4.12: MPI code fragment for performing a row-wise halo exchange

4.6.2 File locking

The most intuitive way of dealing with overlaps is enforcing explicit file locks in order to grant processors exclusive access to overlaps. This can be achieved though the use of mutual exclusion synchronization. When a processor is operating on its region of the file, no other processor can operate on that region. When using column-wise partition, file locking is the worst thing a programmer can do. As mentioned earlier, column-wise partitioning results in noncontiguous data access. Locking a file region ultimately locks the entire file. Shown

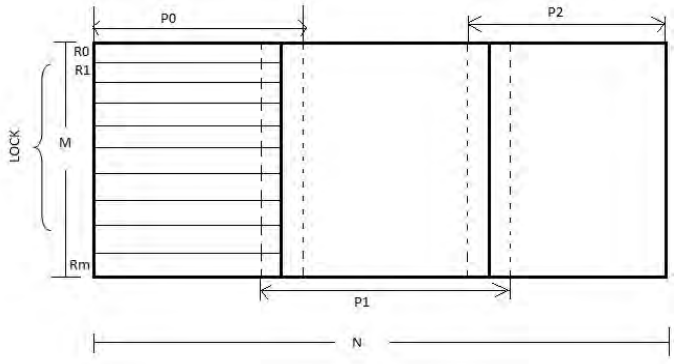

Figure 4.13: File locking in column-wise partition

in Figure 4.13, for processor P0 to write to the first element of R1 after writing to R0, it needs to traverse to the end of the row (i.e. *N* columns) before coming back to R1. Locking this operation will consequently lock the entire file resulting in completely serialized I/O operations which renders MPI parallel I/O useless. Idle processor cause quite a large overhead while waiting on the operating processor to complete its write operation. When using row-wise partitioning, file locking does not cause any atomicity problems because only one

processor can access the overlapped region at a time. The solution presented in the next section discusses a way around file locking.

4.6.3 Process-Rank Ordering

This third approach to dealing with boundary pixels is based on a strategy termed by Liao et al [6] as process-rank ordering. The processors are granted priority levels to be used to give them exclusive access to overlapped data. The higher ranked processor wins when an overlap is encountered. Lower ranked processors then must modify their requests by subtracting the overlaps. Data resulting in overlaps will only be written by the processor with the higher rank. This effectively eliminates all overlaps and achieves atomicity. The overhead in rank ordering is the cost of re-generating access regions for all processors [6].

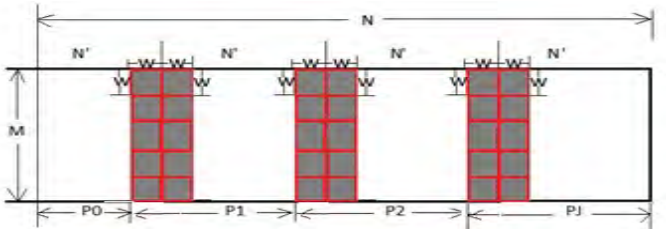

Figure 4.14: Overlapped data access using process-rank ordering

Figure 4.14 shows a graphical view of the file views resulting from process-rank ordering. A file view for processor Pi ($0 < i < j$), is an $M \times N'$ subarray and the file view for P0 and Pj are $M \times (N' - W)$ and $M \times (N' + W)$, respectively. Since each processor surrenders its write to the rightmost column, all overlaps are removed and MPI atomicity is maintained. A code fragment presented in Figure 4.15 shows how the file view are constructed for column-wise partitions.

```
1. sizes[0] = M;          sizes[1] = N;     //global image dimensions
2. l_sizes[0] = M;        l_sizes[1] = N/P; //local subarray dimensions
3. if(rank==0) l_sizes[1]-=R;
4. starts[0] = 0;         starts[1] = (rank==0)?0:rank*(N/p-R);
5. MPI_Type_create_subarray(2, sizes, l_sizes, starts,
6.                          MPI_ORDER_C, MPI_INT, &filetype);
7. MPI_Type_commit(&filetype);
```

Figure 4.15: MPI code fragment for column-wise file view construction

Figure 4.16 summarizes the parallel algorithm decomposition in graphical view.

4.7 Conclusion

This chapter presents a cheaper way of improving the scalability of fingerprint identification systems through distributed and parallel processing. Three different strategies of processing

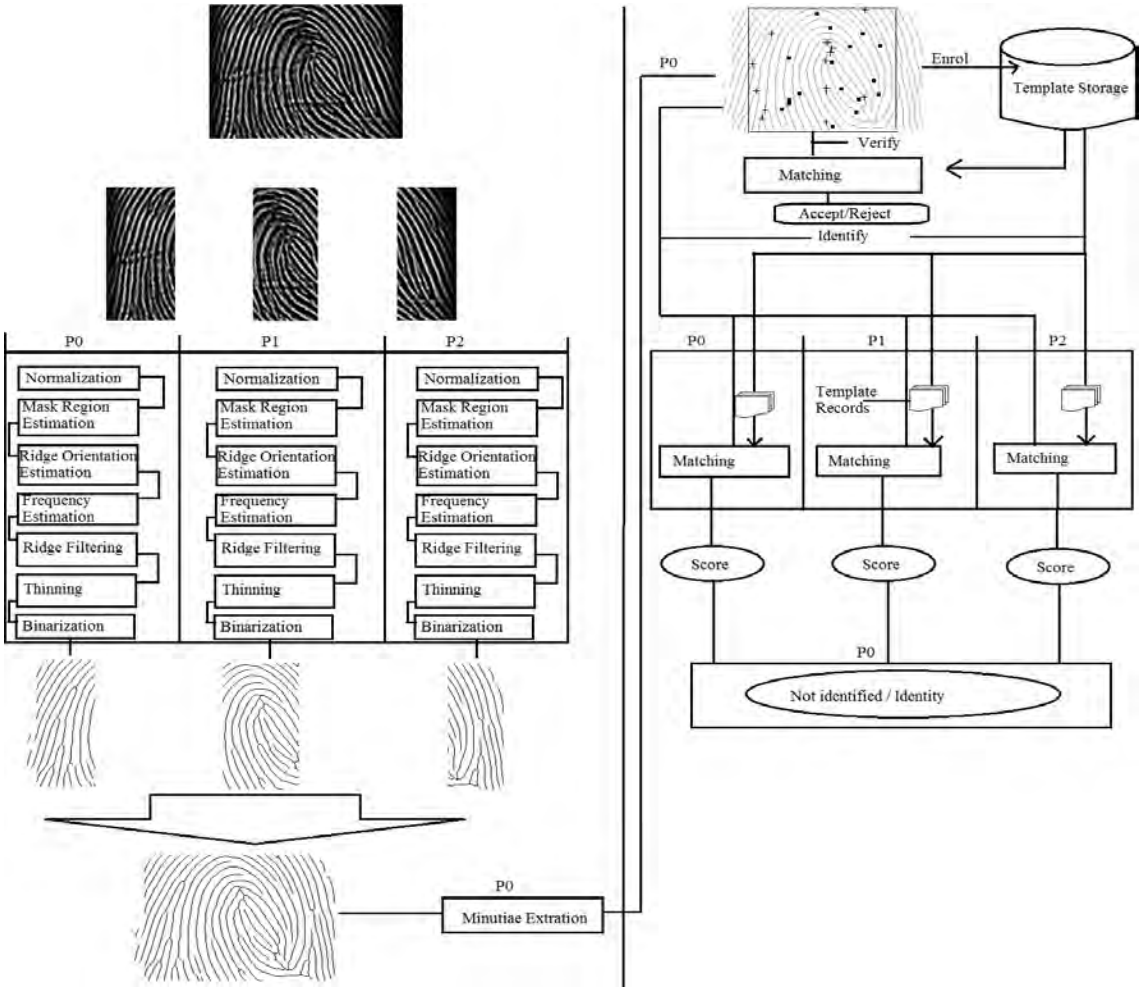

Figure 4.16: Graphical view of the system code flow

boundary pixels are discussed along with a novel approach to memory distribution of block-wise image processing operations on distributed environments.

Chapter 5

Performance Analysis

This chapter discussed the factors that affect the performance of distributed systems along with the necessary optimizations to improve the efficiency of such systems. A novel performance model is introduces which enables performance prediction.

5.1 Introduction

Distributed systems are collections of autonomous computing systems which are connected by some network. These systems work together as one entity to solve large problems by splitting them up into smaller subproblems in a divide and conquer manner. The processing time of algorithms running on distributed systems is calculated as the total computation time plus the time spent on communication among the processors. Performance is an important part of software development. Clients often need to know the expected performance so that they can make an informed decision on whether or not to invest in a project. For developers, performance prediction gives a good idea of how the system may behave, allowing them to locate possible bottlenecks from the system before development [73]. Performance prediction methods in literature can be classified into three categories; analytical [74–79], profile-based [80, 81] and simulation-based [82–85]. Analytical methods work by decomposing an application into an algebraic expression [77] and model the performance mathematically. Simulators on the other hand analyze the source code directly, which reliefs users of the duty of having to analyze lengthy programmatic features into mathematical models. They characterize the code and the hardware it is running on and use the resulting models collectively to derive the predictive execution data. Although simulation-based approaches have high accuracy, they have high computational cost [86]. Existing simulations-based approaches include MPI-SIM [83], PACE [87], WARPP [88] and SimOs [84]. Analytical solutions have the advantage of efficiency over the rest of the prediction methods, however, it is limited by the fact that many complex systems are analytically intractable [89].

Amdahl came up with a law for predicting performance of parallel systems, which has long after been disputed. The skepticism surrounding Amdahl's law is over the assertion that parallel processing is unscalable [74]. Amdahl's law stipulates that even when the serial fraction of a problem, say s, is considerably small, the maximum attainable speedup is only

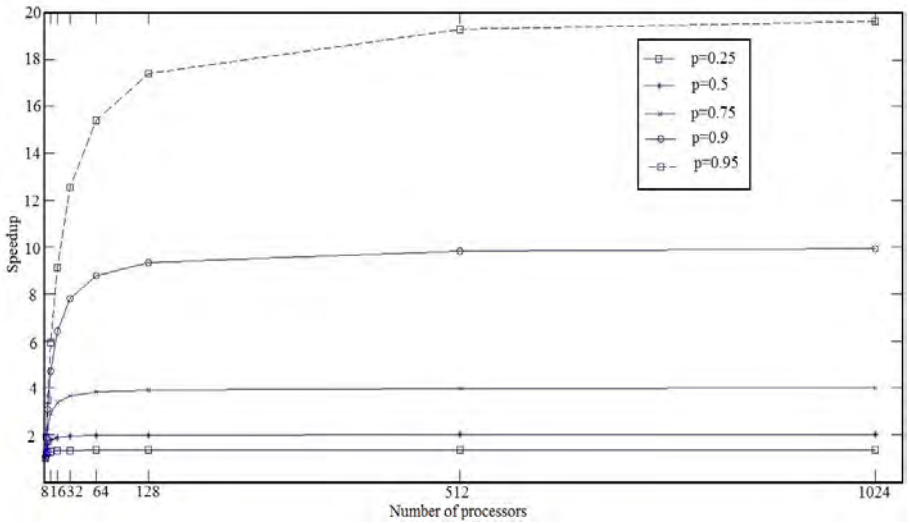

Figure 5.1: Amdahl's fixed-size speedup

$\frac{1}{s}$ even for an infinite number of processing nodes [74]. If s is the time spent by N processors executing the serial fraction of the computation time of a program and p is the time spent executing the parallel portion, then Amdahl's law states that the estimated speedup is given by:

$$Speedup = \frac{1}{(s + \frac{p}{N})}, \text{ with } s = 1\text{-}p \tag{5.1}$$

Amdahl's law assumes that the problem size remains fixed after parallelization. This, however, is usually not the case. It has been shown that in practise, parallel processing workload scales up with the number of processers [74, 75]. Gustafson [74] discussed the concept of scalable parallel processing and introduced the scaled-sized model for speedup. When it comes to parallel processing, Gustafson states that the parallel portion of the program scales up with the problem size, while the serial portion, comprised of program loading, serial bottlenecks and I/O, stays fixed.

Figure 5.1 plots five curves using Amdahl's law. From the graph, the assumption that p is independent of N is implicit, even though this is hardly ever the case [74]. Figure 5.2 plots five curves using Gustafson's law under conditions identical to those of Figure 5.1. The plot shows great scalability without any upper bounds. Gustafson argued that the workload of parallel problems scales up with the increase of processing nodes making the speedup linearly dependent on the number of processors N.

To derive Gustafson's law, consider using a serial processor to process the entire workload. It would take $s + pN$ to complete the task. From this the scaled speedup is calculated as:

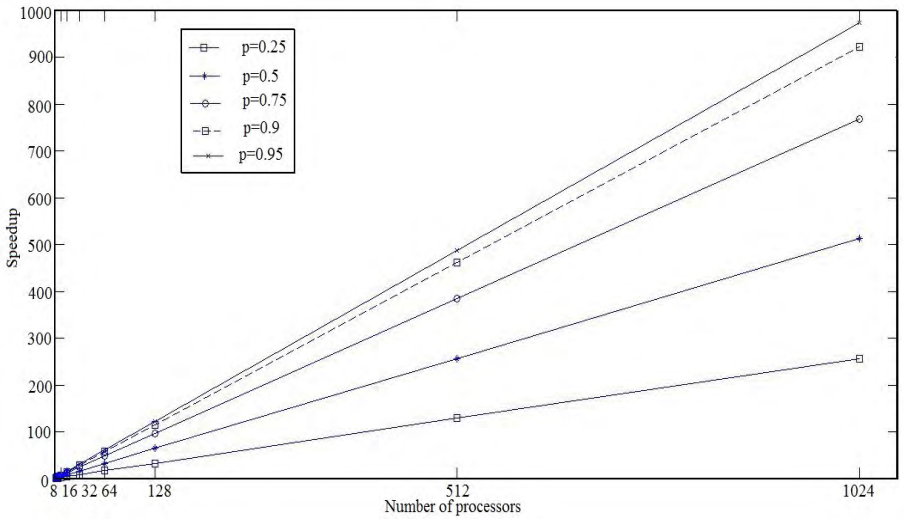

Figure 5.2: Gustafson's scaled-size speedup

$$\begin{aligned} Scaled\_speedup &= \frac{(s+pN)}{(s+p)} \\ &= s+pN \end{aligned} \tag{5.2}$$

Hill & Marty [90] revised Amdahl's law for multicore architectures. In order to apply Amdahl's law in a multicore environment, a cost model for performance of the cores that the chip supports is required. Assume a symmetric multicore architecture with each core having its own L1 cache, where the memory bound is the cumulated capacity of the L1 caches. Variable $perf(r)$ is defined as the sequential performance of a powerful core with r Base Core Equivalents (BCEs). *"Under Admahl's law, the speed up of symmetric multicore chips depends on the software fraction that is parallelizable (f), the total chip resources in the BCEs (n), and the BCE resources (r) devoted to increasing each core's performance"* [90]. The resulting speedup for the symmetric multicore architectures is as follows:

$$Speedup(f,n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f.r}{perf(r).n}} \tag{5.3}$$

Figure 5.3 is a plot of six curves at different f-values. Like Amdahl's law, Hill & Marty's corollary lacks scalability. Since the corollary applies Amdahl's concepts, it made the same inaccurate assumption that problem workload remains fixed after parallelization. This assumption lead to the conclusion that multicore architectures' scalability is questionable, which was quickly challenged by Sun & Chen [75]. Sun & Chen applied the scalable computing principals presented by Gustafson's law. The same hardware model architecture proposed by Hill & Marty was used to demonstrate the scalability of multicore architectures through a fixed-time model (as opposed to fixed-size) [75]. Sun & Chen define the fixed-time speedup as:

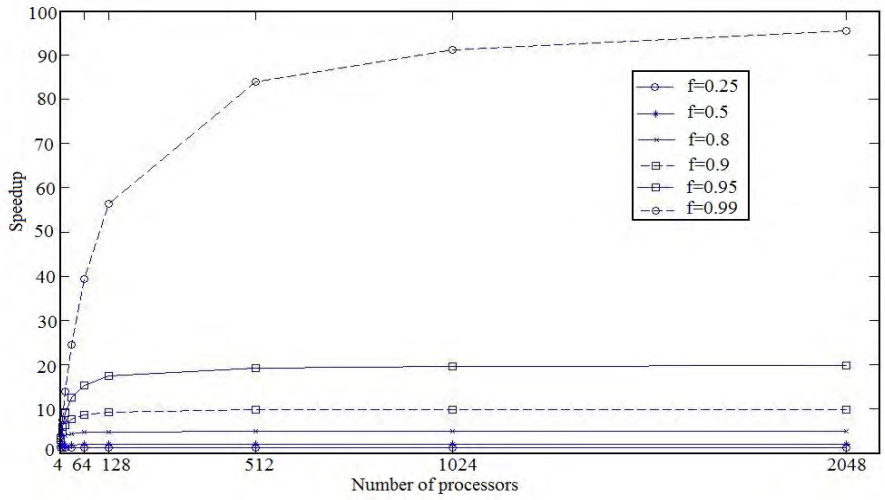

Figure 5.3: Hill & Marty's fixed-size performance model

$$Speedup_{FT} = \frac{Sequential\ Time\ of\ Solving\ Scaled\ workload}{Parallel\ Time\ of\ Solving\ Scaled\ workload} \tag{5.4}$$

Let w be the original workload and w' be the scaled workload. Supposing the time taken to process w sequentially is the same as the time taken to process w' in parallel using m processors.

Assuming that the scale of the workload is only on the parallel portion; w' becomes:

$$w' = (1-f)w + mfw \tag{5.5}$$

Therefore

$$Speedup_{FT} = \frac{Sequential\ Time\ of\ Solving\ w'}{Parallel\ Time\ of\ Solving\ w'} \tag{5.6}$$

$$Speedup_{FT} = \frac{Sequential\ Time\ of\ Solving\ w'}{Sequential\ Time\ of\ Solving\ w} \tag{5.7}$$

$$\frac{w'}{w} = \frac{(1-f)w + mfw}{w} = (1-f) + mf \tag{5.8}$$

which gives Gustafson's law [74]. The scaled-sized model assumes that the scaling is only at the parallel portion. Based on this assumption and following (5.3), Sun & Chen constructed the fixed-time speedup model to be:

$$\frac{(1-f)w}{perf(r)} + \frac{fw}{perf(r)} = \frac{(1-f)w}{perf(r)} + \frac{fw'}{perf(r)m} \tag{5.9}$$

If we let $n = mr$ be the scaled number of cores, with $n = r$ being the initial point, then $w' = mw$. The final scaled speedup compared with $n = r$ becomes:

$$\begin{aligned} Speedup_{FT} &= \frac{Sequential\ Time\ of\ Solving\ w'}{Sequential\ Time\ of\ Solving\ w} \\ &= \frac{\frac{(1-f)w}{perf(r)} + \frac{fw'}{perf(r)}}{\frac{w}{perf(r)}} = (1-f) + mf \end{aligned} \tag{5.10}$$

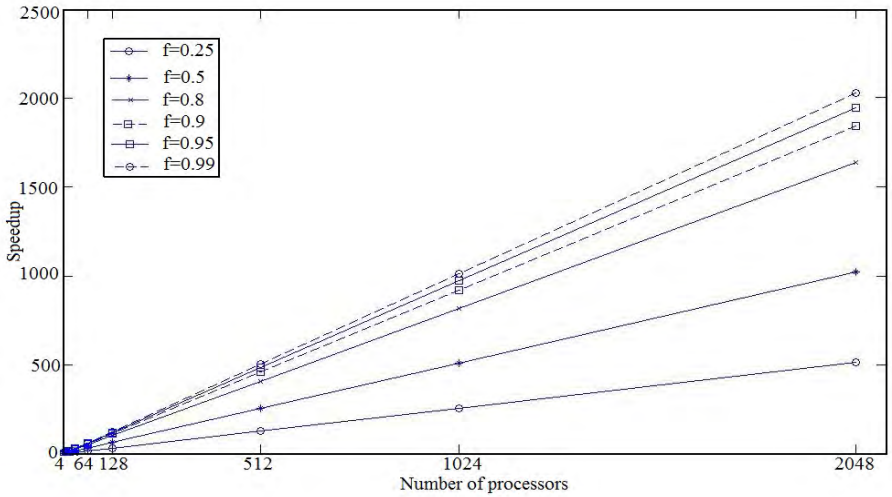

Figure 5.4: Sun & Chen's fixed-time performance model

Figure 5.4 shows six curves of the fixed-time performance model at different f-values. The fixed-time speedup model demonstrates the scalability of multicore systems. Like the scaled-sized model, it is linearly dependent on the number of processors m. Although Sun & Chen successfully model the performance of parallel processing on multicore systems, they do not cater for distributed multicore systems.

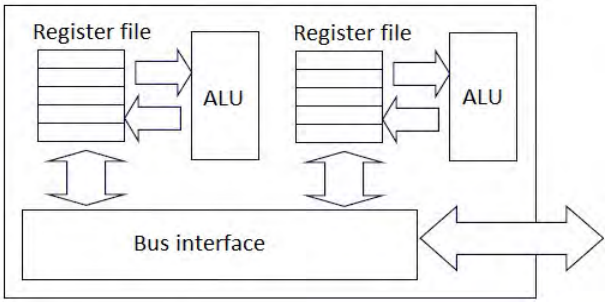

Figure 5.5: Structure of a dual-core system (taken from [7])

Distributed computation differs from parallel computation in the way in which memory is used. In parallel systems, all processing elements use the same shared memory for communication and I/O, whereas distributed systems are autonomous systems with private memory connected by a network which is used for communication between the processing nodes. Figure 5.5 shows an internal structure of a parallel/shared memory system in the form of a dual-core system, while Figure 5.6 shows an example of a multicore distributed system, where multicore machines are combined by a network to function as one.

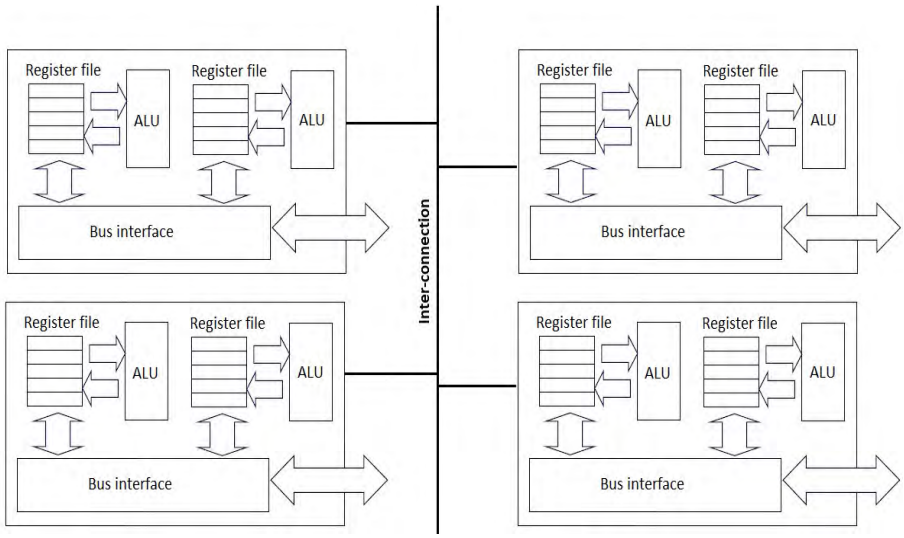

Figure 5.6: Example structure of a multicore cluster (adapted from [7])

Parallel code often runs on the same system and thus has no need for external communication. Distributed code, on the other hand, can not work without external communication. This communication, however, often consists of some overhead which, in large amounts, can affect performance drastically. The performance prediction models discussed above do not address the communication issue associated with distributed processing. For this reason, this research presents a way of predicting performance for code distributed on multicore clusters.

5.2 Factors Affecting Performance in Distributed Systems

The performance of a distributed algorithm is affected by more than just the application efficiency and the number of processing nodes used. Since clusters are connected by networks, network factors like latency and bandwidth have a considerable impact on the performance of a distributed system. As such, it is necessary to take into account the network influence when predicting the performance of these systems [91].

Bandwidth and latency capture the volume and time dimensions of information processing, respectively. Latency measures the time taken to complete a request, while bandwidth

measures the volume of information transmitted in a time interval [8]. The next subsections go into greater detail about the factors that impact information processing performance.

5.2.1 Application Efficiency

Algorithm efficiency is the most crucial factor when it comes to parallel algorithm performance. If an algorithm is not efficient in how it utilizes resources, even the most powerful machines can not improve its performance. Distributed algorithms have to be optimized at two levels: per-processor (i.e. each core of a machine) and across-processors (i.e. communication across the cluster) [91, 92]. Optimization techniques include code modifications and compiler optimizations. Per-processor optimizations include but no limited to [49]:

1. Loop optimization
    a) Unrolling
    b) Splitting
2. Memory optimization
    a) Prefetching
    b) Cache alignment and coherence
    c) Stride-one memory access
3. Floating point arithmetic
4. Use of optimized mathematical libraries

Across processors optimizations mainly deal with [49]:

1. Minimizing:
    a) Communication overhead
    b) Synchronization overhead
    c) Load imbalance
    d) Memory consumption
    e) Computation overhead
2. Latency hiding techniques such as overlapping computation and commucation
3. Efficient network interconnection
4. Avoiding master-only operations as they force the rest of the processors to idle

Lastly, compilers like GNU come with optimization flags such as *-ffast-math* which optimizes mathematical functions.

5.2.2 Application Latency

Application latency is defined by Shaffer [76] as the total amount of time that an application has to wait for a response after issuing a request for some data. The application delay reflects the total wait time incurred by the system, including all subsystems and kernel overhead as well as network latency [76].

Network latency is the time spent waiting, from the instantiation of an operation until the return of the desired results [76]. A distinction can be made amongst the different types/sources of network latency. Three types of network latencies are discussed; the propagation delay, transmission delay, and physical latency. Figure 5.7 is a representation of these two network latency sources as a single server open queueing system.

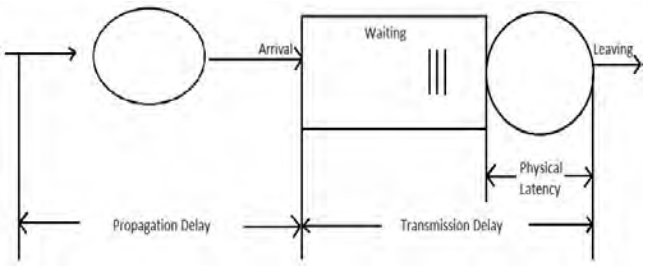

Figure 5.7: Network latency presented as a propagation and transmission delay server (adapted from [8])

Propagation Delay

Propagation delay is defined as the time taken waiting for the last bit to arrive plus the overhead that comes with the device [76]. The propagation delay can not be eliminated or avoided because the speed of light is inviolable [76]. Transmission speed can not be improved beyond propagation delay. A propagation of a certain system indicates the maximum transmission rate that the system can achieve.

Physical latency

Physical latency measures the processing time on a device without waiting (i.e. the service time). It varies according to device utilization or load [8]. The physical latency can be halved to double the bandwidth. Ding [8] showed that halving the physical latency yields better results than actually doubling the bandwidth. The system is able to perform twice the amount of work without saturation.

Transmission Delay

Transmission delay is the amount of time taken to transmit all the packet's bits into the link. In most networks, transmission of packets occurs in a first come first serve manner, which

results in queueing for transmission rates that are not high [93]. It is determined by the packet size and the transmission rate of the network and not at all affected by the distance.

5.2.3 Bandwidth

Bandwidth determines how much information can be processed within a certain time interval. It has a direct impact on the response time of data acquisition [8]. Low bandwidth can result in extremely slow systems. If an application must be able to transmit at a certain rate in order to be effective, then that application must transmit and receive at that rate. If that amount of bandwidth is not available, the application is most likely to give up [93]. Bandwidth may be increased to improve performance of a certain system and compensate for the propagation delay. However, increasing the bandwidth does not automatically guarantee performance gain. In order to benefit from high bandwidth, software often needs to be modified in order to leverage the high bandwidth. For example, applications developed for 32-bit systems may run slower on 64-bit systems [8].

5.3 Performance Model

Efficiency of a parallel algorithm is measured by the speedup attained. If T_1 is the execution time for the serial implementation, the speedup can be computed as $\frac{T_1}{T_N}$, where T_N is the execution time attained when using N processors. Efficiency is then calculated as:

$$E_N = \frac{T_N}{N} \tag{5.11}$$

An efficient algorithm attains a speedup close to N for every T_N, (i.e. $E_N = 1$). It has been established in the literature that for distributed systems, this is not always the case. As the number of processors increases, speedup of the distributed systems starts to decline. This is usually because of the increased interprocessor communication, known as message passing. Adding computation nodes increases the networks communication links which ultimately increases propagation delay.

This research focuses on the performance of multicore clusters using an analytical approach based on Sun & Chen's [75] and Shaffer's [76] works. Multicore clusters are ideal for hybrid programming, (i.e. a mixture of distributed and parallel processing). While multicore systems are scalable and provide high performance, they have their limits. Writing thread-safe programs is not easy, especially as the number of threads increases [30]. Enrico Clementi, a former IBM fellow and pioneer in computational techniques for quantum chemistry and Molecular dynamics, once said *"I know how to make 4 horses pull a cart - I don't know how to make 1024"*. Introducing clustered systems relieves the strain of using too many threads on one machine.

5.3.1 Computational Cost

In distributed processing, an application can only run as fast as the slowest processor. Thus, following Sun & Chen's fixed-time model, $perf(r)$ is redefined to be:

$$perf(r') = max(perf(r_i)) \tag{5.12}$$

where $perf(r_i)$ is the sequential performance of a powerful core of a processing node i with r BCEs. Using (5.9) and the assertion that $w' = mw$ we get the following,

$$\frac{(1-f)w}{perf(r')} + \frac{fw'}{pref(r')m} = \frac{\frac{(1-f)w}{perf(r')} + \frac{fw}{perf(r')}}{\frac{w}{perf(r')}} = (1-f) + mf \tag{5.13}$$

This gives us the expected speedup [92].

5.3.2 Communication Cost

The communication overhead associated with message passing can be quite large. Shaffer [76] proposed a theoretical predictive measure of communication cost in wide area distributed systems to be:

$$Comm\_Time = m \times [\frac{s}{b} + d \times 7.67 \times 10^{-6} + \varepsilon] \tag{5.14}$$

where m is the frequency of messages needed during the task, b is the bandwidth in bits/second, ε is the overhead incurred per message and s and d represent the size of the message and the length of the communication channel in miles, respectively.

Propagation delay is normally calculated as the reciprocal of the speed of light which is currently 299792.458 km/s. However, Shaffer stated that this value is not the same for all types of cables. Different types of cables transmit at different speeds, which is less than the actual speed of light. This speed is known as the normal velocity of propagation (NVP). Optical fiber has an NVP close to 0.7 [76].

We define the cost of sending an L bit message between two processors as:

$$T_{comm}(L) \leq \frac{L}{\tau} + (\sigma_{max} \times dist) + \varepsilon_L \tag{5.15}$$

where τ is the upper bound of the network bandwidth, σ_{max} is the maximum delay incurred by the system, $dist$ is the physical distance between the network points and $\varepsilon(L)$ is the overhead associated with each message of size L bits, i.e. the send and receive overhead [92].

5.3.3 Prediction Model

The total estimated running time is calculated as:

$$T_{EST}(m) = T_{comp}(m) + T_{comm}(m,L) \tag{5.16}$$

where $T_{comp}(m)$ is the computation time and $T_{comm}(m,L)$ is the total time spent by m nodes communicating messages of sizes L.

$$T_{comp}(m) = T_{seq}/Speedup_{FT} \tag{5.17}$$

$$T_{comm}(m,L) = \sum T_{comm}(L) \tag{5.18}$$

T_{seq} is sequential time and $Speedup_{FT}$ is defined in (5.13).

5.4 Conclusion

This chapter presented optimization techniques which can be used to improve the efficiency of distributed systems through code modifications, design choices and network manipulation. A new performance prediction model was presented which models both the computation and communication complexity of distributed applications.

Chapter 6

Experimental Results

In this the performance of the serial implementation is compared with the implementation of the distributed algorithm. The results are shown and discussed.

6.1 Introduction

The two fingerprint algorithms discussed in this research were both implemented and tested on the CASIA fingerprint database v5. This chapter presents the results obtained from the experiments and a few discussions on what those results entail.

6.2 Results

Figure 6.1 shows the results obtained using the prediction model separated into communication and computation time.

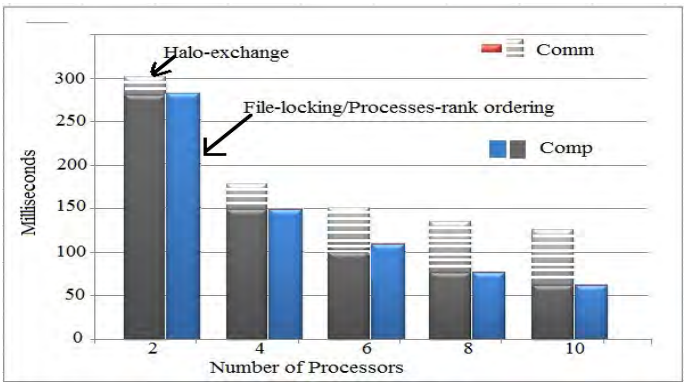

Figure 6.1: Total execution time (computation + communication).

From the graph, a drastic increase in the communication time with the increase of processing nodes can be observed on the halo exchange predictions. This is mainly due to the

frequency of message passing during the prefetching of boundaries cells as well as the size of the messages.

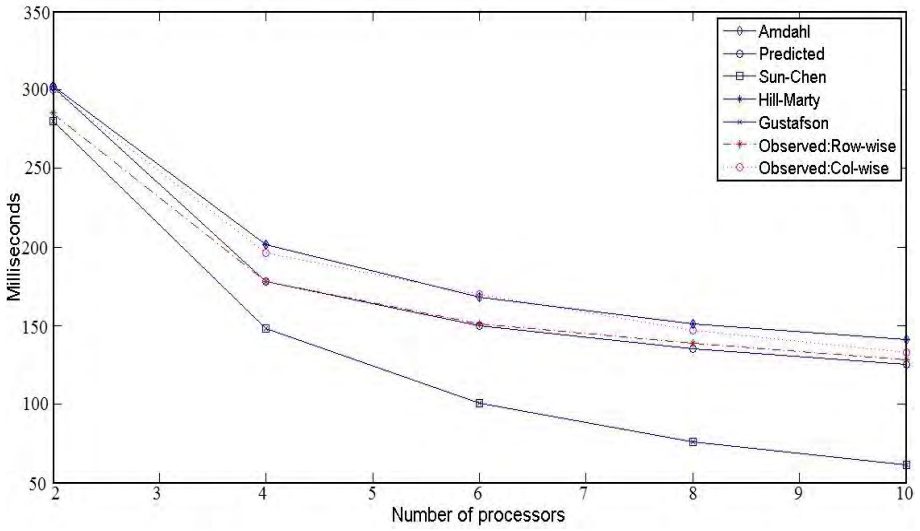

Figure 6.2: Experimental results from the halo exchange experiments plotted against the 5 prediction models

A total of three sets of experiments were performed using the algorithm. Figure 6.2 shows results obtained from using halo exchange on both row- and column-wise partitions of the data plotted with the predictions from the five models.

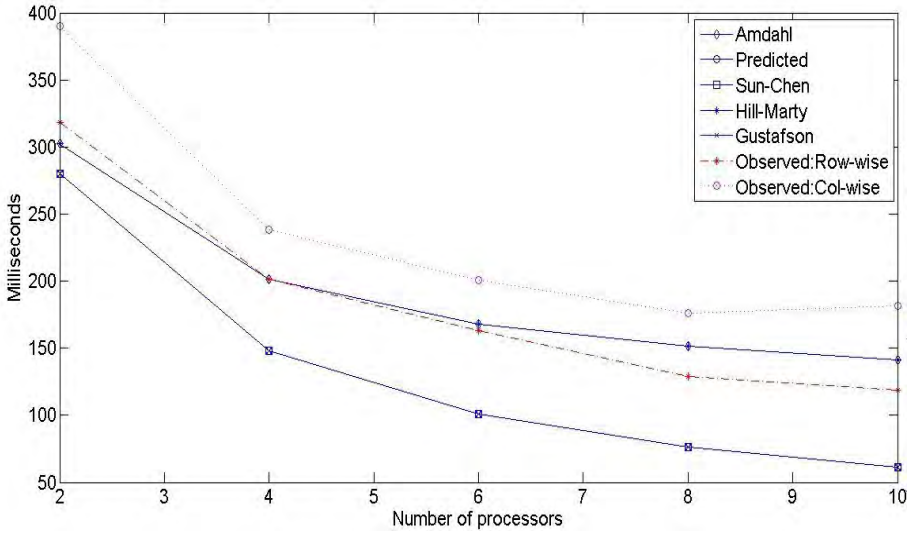

Figure 6.3: Experimental results from file locking experiments plotted against the 5 prediction models

From experiment it can be observed that Gustafson's and Sun & Chen's models over estimates the speedup, whereas Hill & Marty's and Amdahl's models under estimates. Gustafson's and Sun & Chen's models do not consider the effects of communication associated with distributed systems. The propsed model does not give the exact estimates, it over estimates the speedup but the error margins are better than those experienced by other models.

Figure 6.3 shows the results obtained of using file locking on both row- and column-wise partitions plotted with the predictions made by the five models. The performance observed from this experiment shows a good example of poor optimization techniques and bad design choices. When using column-wise partition, file locking is the worst design choice one can make.

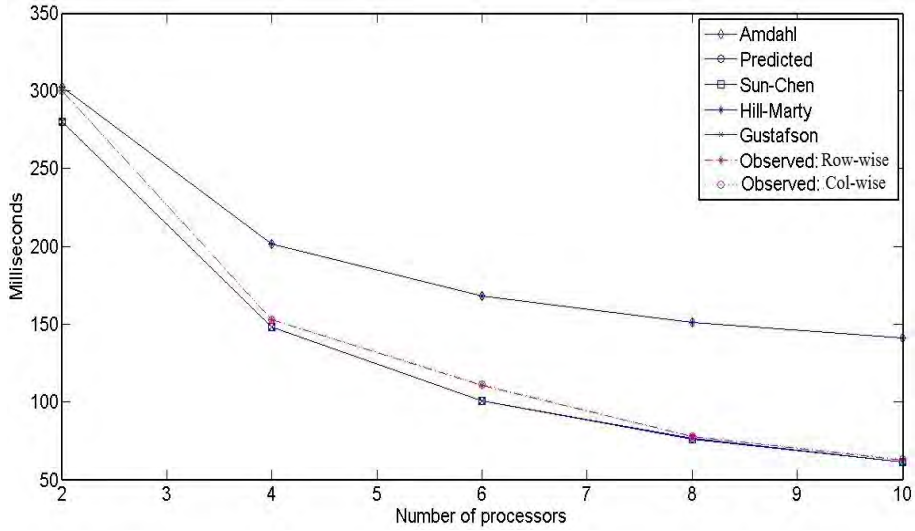

Figure 6.4: Experimental results from the process-rank ordering experiments plotted against the 5 prediction models

Column-wise partition causes noncontiguous access patterns. What this means is if each processor is assigned $M \times N'$ subarray of an image, then the distance between two consecutive rows is $N > N'$. It is not possible for a processor to access all its data in a single read/write operation. Row-wise partition is better in this manner as it does not lock the entire file.

Figure 6.4 plots the results obtained using process-rank ordering on row- and column-wise partitions. Process-rank ordering maintains full I/O parallelism for both column-wise and row-wise partitions and hence is expected to out-perform the file locking strategy. Row-wise partition requires less effort to construct, while column-wise partition requires a little more effort due to the noncontiguous access patterns. This would explain the slight variation the performance. The proposed model, Gustafson's and Sun & Chen's models estimations produces better error margins than those produced by Hill & Marty and Amdahl's models.

To compare the speed gain when using the distributed version of the enhancement algo-

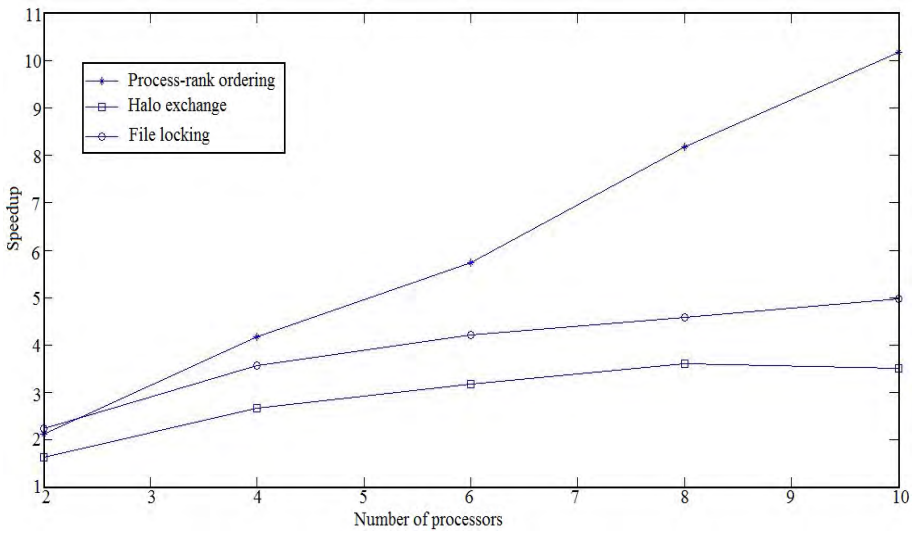

Figure 6.5: Performance gain graph showing the different speedups achieved by the different strategies.

rithm, speedup curves corresponding to the three strategies are plotted in Figure 6.5 to show a graphic view of the performance gain.

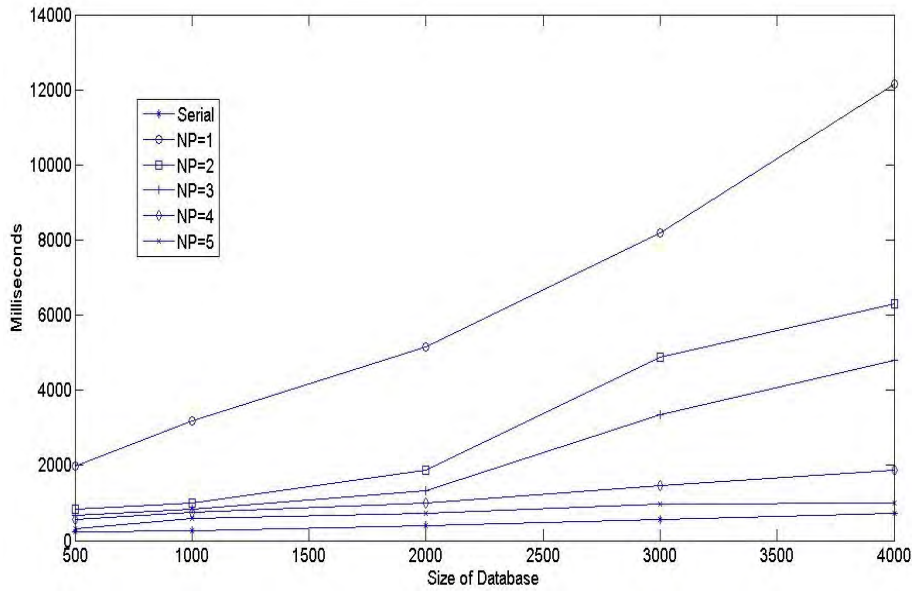

Figure 6.6: Searching times on different database sizes

A parallel searching algorithm was developed which virtually splits the database and assign the different partitions to different processing nodes. The partitioning of the database

helps in reducing the penetration rate similar to the advantage of winnowing, without the penalty of binning errors brought fourth by winnowing. Figure 6.6 shows the results from performing an exhaustive search, a winnowed search and the parallel search at different number of processing nodes. The experiments were performed at different database sizes, note that a database containing 4000 fingerprint can produce roughly 392000 minutiae records to search through.

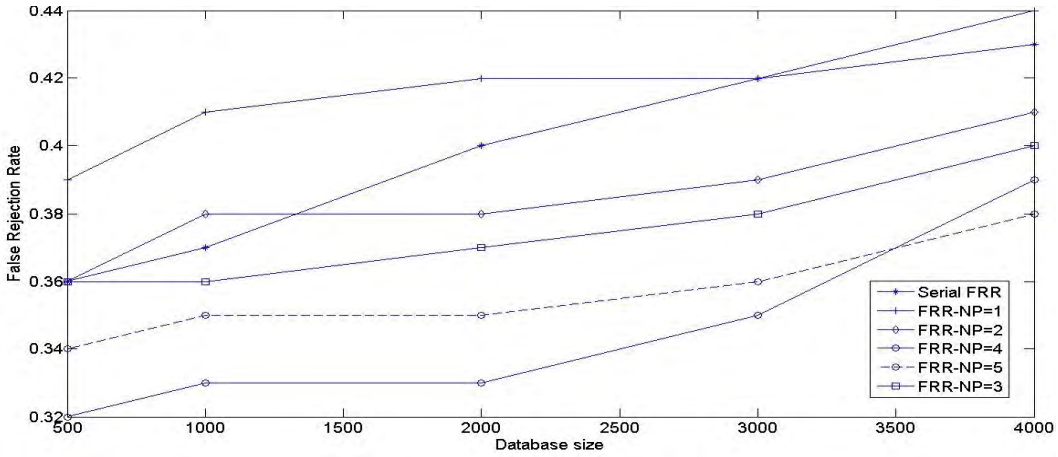

Figure 6.7: Observed False Rejection Error Rates

$NP = 1$ represents an exhaustive search (uses only one processor to search a database sized N). From the experiments, a tremendous improvement from exhaustive searching can be observed. The parallel implementation is highly scalable and performs closely to the winnowed at search at $NP = 5$ nodes with introducing any errors (as opposed to winnowing). Figures 6.7 and 6.8 plot the error rates incurred by the two systems.

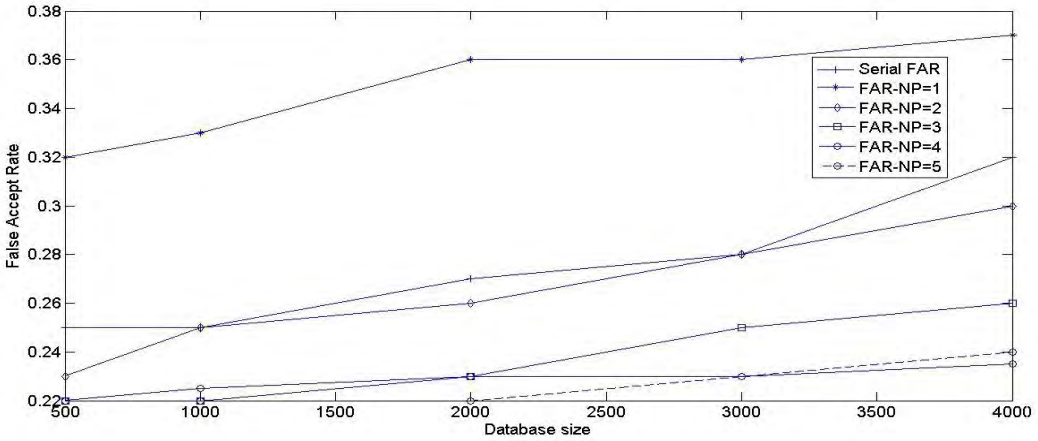

Figure 6.8: Observed False Acceptance Error Rates

The observed False Rejection Rates (FRR) show that the exhaustive search and the winnowed search have higher FRR rate in comparison to the parallel search and these rates as expected, increase with the increase of the database size.

The observed False Acceptance Rates (FAR) are fractions higher for the exhaustive search, and this is caused by the increase in database size. The number of FAR grows geometrically with the size of the database for identification systems. In verification mode, FAR is calculated as [26]:

$$FAR = 1 - FRR \tag{6.1}$$

So for identification on a database of size N [21]:

$$FAR_N = 1 - FRR^N = 1 - (1 - FAR)^N \simeq N \times FAR \tag{6.2}$$

From (6.2) and Figure 6.8 it can be seen that large scale databases scale poorly. Mhatre et al [21] state that to attempt to reduce the identification error on an biometric system, two approaches can be taken:

1. Reduce the FAR of the matching algorithm
2. Reduce the search space

The first choice can only provide limited improvement before it starts increasing the FRR by disallowing intra-user variation [21, 26]. The more practical solution is reducing the search space. This research uses a parallel searching algorithm which reduces the search space *without* compromising the accuracy of the system. Winnowing effectively increase the efficiency of the search algorithm but introduces new errors associated with classification.

6.3 Conclusion

Experimental results show great improvement on the fingerprint enhancement subsystem as well as the matching/searcing subsystem. The distribution of the fingerprint recognition process does indeed improve its performance.

Chapter 7

Conclusions and Future Work

This chapter gives an overview of the this research's contributions. After which, the author reflects on the results and gives some conclusions. Finally, some ideas for future work are to be discussed.

7.1 Conclusions

Does using a parallel and distributed paradigm vastly improves the performance of biometric identification with respect to time? Indeed this research clearly showed that parallelizing the fingerprint recognition algorithm and porting it to a distributed environment improves its response time greatly. The fingerprint enhancement subsystem gained up to 10.2 times speed up while the searching time gained an improvement of up to 12.5 times in comparison to the exhaustive search.

Is it possible to increase identification accuracy through efficient processing? This research showed that the error rates of associated with exhaustive searching increase geometrically with the increase of the database size which renders identification systems poor when it comes to scaling. In order to decrease the false acceptance rates associates with large databases, the search space needs to be reduced. While winnowing effective achieves this, it introduces new errors to the system which if not properly calibrated can lead to the system performing worse than it does under exhaustive search. Parallel searching effectively reduces the search space without introducing any new errors to the system.

Using the parallel and distributed paradigm proposed for this research, is it possible to formalized the performance of the proposed system in order to predict the performance before development? A new performance prediction model was developed during the course of this research. The model improves on Amdahl's law and factors in the effects of network on the performance of distributed systems.

7.2 Discussions and recommendations

The experimental results show that it possible to improve the performance of biometric system though High Performance Computing. Three different implementation strategies

were proposed for processing pixels during enhancement. Of the three strategies, process-rank ordering shows the promising results with great scalability, reaching up to 10.2 times speedup. While halo exchange offers performance gain, it scales poorly in comparison to process-rank ordering, achieving only 4.5 times speedup. The main reason behind the poor scalability is the overhead introduced by the excessive communication required to perform the halo exchange. Adding more processing nodes (in order to increase processing power) increases communication nodes which in turn increase the overhead incurred by the system. File locking perform the most poorly of the three strategies, only achieving up to 2.7 times speedup. Main reason for the poor performance is due to bootlenecks caused by idle processors during the locks.

While this research successfully showed an improvement on the fingerprint recognition algorithm, it still has its short comings and limitations. The most obvious one being the high false rejection rates. The proposed matching algorithm still needs further calibration as it allows very little intra-user variation. As for the enhancement algorithm, the minutiae extractor was not parallelized as distributing the thinned image would have broken up some of the minutiae points.

7.3 Future work

For future, the network (bandwidth$\times$latency) product can be manipulated in order to further increase processing time of the system. The matching algorithm still needs some improvement in order to reduce the false rejection rates which are quite high. The parallel search algorithm can be further improved by performing winnowing within each node, provided the classification algorithm is very good and produces minimal classification errors.

References

[1] P. Campisi, R. L. Carter, C. W. Crooks, and V. Govindaraju, *IEEE Certified Biometric Professional (CBP) Learning System. Module1: Biometric Fundamentals*, USA, 2010.

[2] American University of Beirut Faculty of Engineering and Architecture Department of Electrical and Computer Engineering , "Fingerprint identification - project 2," *EECE695C - Adaptive Filtering and Neural Networks Lecture Notes*.

[3] I. S. Msiza, B. Leke-Betenchuoh, F. V. Nelwamondo, and N. Msimang, "A fingerprint classification approach based on the coordinate geometry of singularities," *Proceedings of The 2009 IEEE Internation Conference on Systems, Man and Cybernetics*, pp. 516–523, Oct 2009.

[4] D. Scott. (Last accessed 23 Aug 2012) Understanding fingerprints. Internet draft. [Online]. Available: http://http://www.viewzone.com/fingerprintsx.html

[5] I. Msiza, T. Malumedzha, and B. Leke-Betechuoh, "A novel fingerprint re-alignment solution that uses the tfcp as a reference," *International Journal of Mechine Learning and Computing*, vol. 01, no. 03, pp. 297–304, Aug 2011.

[6] W.-K. Liao, K. Coloma, A. Choudhary, L. Ward, E. Russell, and N. Pundit, "Scalable design and implementations for MPI parallel overlapping I/O," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 11, pp. 1264–1276, Nov 2006.

[7] J. Barbic. (2007, May) Multi-core architectures. Lecture Notes. [Online]. Available: http://www.co-array.org/cafvsmpi.htm

[8] Y. Ding, "Bandwidth and latency: Their changing impact on performance," *Proceedings of the 35th Computer Measurement Group Conference*, pp. 67–78, 2005.

[9] L. Smith, "Mixed mode MPI/OpenMP programming," *UK High-End Computing Technology Report*, 2000.

[10] R. N. Rakvic, B. J. Ulis, R. P. Broussard, R. W. Ives, and N. Steiner, "Parallelizing iris recognition," *IEEE Transactions on Information Forensics and Security*, vol. 04, pp. 812–823, Dec 2009.

[11] H. Xu, Y. Qu, Y. Zhang, and F. Zhao, "FPGA based parallel thinning for binary fingerprint image," *Pattern Recognition, Chinese Conference*, pp. 1–4, Dec 2009.

[12] C. M. M. Alberto and A. B. E. Koki, "Fingerprint image enhancement algorithm implemented on an FPGA," 2009.

[13] M. Fons, F. Fons, E. Canto, and M. Lopez, "FPGA-based authentiction using fingerprints," *Journal of Signal Processing Systems*, vol. 66, pp. 153–189, Oct 2012.

[14] L. Hermanto, S. A. Sudiro, and E. P. Wibowo, "Hardware implementation of fingerprint image thinning algorithm in FPGA device," *International Conference on Networking and Information Technology*, pp. 187–191, Jun 2010.

[15] F. Fons, M. Fons, E. Canto, and M. Lopez, "Real-time embedded systems powered by FPGA dynamic partial self-reconfiguration: A case study oriented to biometric recognition applications," *Journal on Real-time Image Processing*, Jan 2011.

[16] R. P. Broussard, R. N. Rakvic, and R. W. Ives, "Accelerating iris template matching using commodity video graphics adapters," *IEEE International Conference on Biometrics: Theory, Applications and Systems*, pp. 1–6, Sep 2005.

[17] M. Qin, "A fast and low cost simd architecture fingerprint image enhancement," *MSc Thesis, Technische Universiteit Delft*, 2005.

[18] N. P. Khanyile, E. Dube, and R.-J. Tapamo, "Distributed fingerprint enhancement on a multicore cluster," *The 16th International Conference on Image Processing, Computer Vision, and Pattern Recognition*, vol. 02, pp. 890–897, July 2012.

[19] M. P. I. Forum, *MPI-2.2: A Message Passing Interface Standard*. High Performance Computing Center Stuttgart (HLRS), Sep 2009. [Online]. Available: http://www.mpi-forum.org/docs.html

[20] P. Campisi, R. L. Carter, C. W. Crooks, and V. Govindaraju, *IEEE Certified Biometric Professional (CBP) Learning System. Module2: Biometric Modalities*, USA, 2010.

[21] A. Mhatre, S. Palla, S. Chikkerur, and V. Govindaraju, "Efficient search and retrieval in biometric databases," *SPIE Defense and Security Symposium*, 2004.

[22] (Last accessed on 30 May 2011) FBI-IAFIS. [Online]. Available: http://www.fbi.gov/about-us/cjis/fingerprints_biometrics/iafis/iafis

[23] L. F. W. Góes, L. E. S. Ramos, and C. P. S. Martins. (Last accessed on 16 Feb 2011) Parallel image filtering using WPVM in a windows multicomputer. Internet draft. [Online]. Available: http://www.research.rutgers.edu/\~luramos/pdf/csitea02filtering.pdf

[24] D. B. Megherbi, A. J. Boulenouar, and V. Rajagopalan, "A high performence distributed memory & computing algorithm for face recognition via conformal mapping"," *Proceedings of the SPIE on Visual Information Processing*, vol. 4388, pp. 246–257, 2001.

[25] H. Xu, Y. Qu, and F. Zhao, "FPGA based parallel thinning for binary fingerprint image," *IEEE Chinese Conference on Pattern Recognition*, pp. 1–4, Nov 2009.

[26] P. Campisi, R. L. Carter, C. W. Crooks, and V. Govindaraju, *IEEE Certified Biometric Professional (CBP) Learning System. Module3: Biometric System Design and Evaluation*, USA, 2010.

[27] L. Hong, Y. Wan, and A. Jain, "Fingerprint image enhancement: Algorithm and performance evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 08, pp. 777–789, Aug 1998.

[28] K. Beyer, J. Goldstein, R. Ramakrishnan, and V. Shaft, "When is "nearest neighbour" meaningful?" *International Conference on Database Theory*, pp. 217–235, 1999.

[29] A. Jain and S. Pankanti, "Fingerprint classification and matching," 2000.

[30] G. Indrawan, B. Sitohang, and S. Akbar, "Parallel processing for fingerprint feature extraction," *International Conference on Electrical Engineering and Informatics*, pp. 1–6, Jul 2011.

[31] (Last accessed on 16 Feb 2011) Biometric identification on cloud computing. Internet draft. [Online]. Available: http://www.afisandbiometrics.com/biometric_identification_on_cloud_computing

[32] N. Vaidya and S. Sherekar, "Study of biometric data processing by hadoop," *MPGI National Multi Conference*, pp. 26–31, Apr 2012.

[33] J. You, D. Zhang, J. Cao, and M. Guo, "Parallel biometrics computing using mobile agents," *Proceedings of the 2003 IEEE International Conference on Parallel Processing*, pp. 305–312, 2008.

[34] M. Din, M. Maarof, and M. Salleh. (2006) Paralle matching system for digital non text information (fingerprint image). Internet draft. [Online]. Available: http://eprints.utm.my/4244

[35] P. Zhang, X. Guo, and J. Gadedadikar, "Online fingerprint verification algorithm and distributed system," *Journal of Signal and Information Processing*, vol. 2, pp. 79–87, Mar 2011.

[36] R. Miron and T. S. Letia, "Two server topology for a distributed fingerprint-based recognition system," *15th International Conference on System Theory, Control and Computing*, 2011.

[37] M. Hulea, A. Astilean, T. Letia, R. Miron, and S. Folea, "Fingerprint recogniton distributed system," *8th Proceedings of the IEEE International Conference on Automation, Quality and Testing, Robotics*, vol. 3, pp. 423–428, 2008.

[38] D. A. Bader, J. JaJa, D. Harwood, and L. S. Davis, "Parallel algorithm for image enhancement and segmentation by region growing with an experimental study," *Proceedings of the 10th International Parallel Processing Symposium*, pp. 414–423, Apr 1996.

[39] T. Nakamura, M. Hirooka, H. Fujiwara, and K. Sumi, "Fingerprint image enhancement using a parallel ridge filter," *IEEE Proceedings International Conference Pattern Recognition*, vol. 01, no. 08, pp. 536–539, Aug 2004.

[40] N. Ikeda, M. Nakanish, K. Fujii, and T. Hatano, "Fingerprint image enhancement by pixel-parallel processing," *IEEE Proceedings International Pattern Recognition and Machine Intelligence*, vol. 03, pp. 752–755, Dec 2002.

[41] M. Barrenechea, J. Altuna, M. Mendicute, and J. D. Ser, "A low-cost fpga-based embedded fingerprint verification and matching system," *Intelligent Technical Systems*, vol. 38, no. 05, pp. 247–260, 2009.

[42] S. Yang, K. Sakiyama, and I. Verbauwhede, "Efficient and secure fingerprint verification for embedded devices," *EURASIP Journal on Applied Signal Processing*, pp. 1–11, 2006.

[43] G. Danese, M. Giachero, F. Leporati, and N. Nazzicari, "A multicore embedded processor for fingerprint recognition," *IEEE Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, pp. 779–784, 2010.

[44] S. Vitabile, V. Conti, C. Militello, and F. Sorbello, "A self-contained biometric sensor for ubiquitous authentication," *Internation Conference on Intelligent Pervasive Computing*, pp. 289–294, 2007.

[45] G. T. Group, "Using GPU technology to solve the latent fingerprint matching problem," *GTC Express Webnar*, July 11 2012.

[46] H. Bui, M. Kelly, C. Lyon, M. Pasquier, D. Thomas, P. Flynn, and D. Thain, "Experience with BXGrid: a data repository and computing grid for biometric research," *Journal of Cluster Computing*, pp. 394–395, Dec 2008.

[47] Y. Sasaki, K. Ito, T. Aoki, and T. Higuchi, "A compact cluster computer with embedded CPUs and its application to rapid prototyping of fingerprint verification," *IEICE Electronic Express*, vol. 02, pp. 465–470, Sep 2005.

[48] K. Ito, H. Nakajima, K. Kobayashi, T. Aoki, and T. Higuchi, "A finngerprint matching algorithm using phase-only correlation," *IEICE Transactions on Fundamentals*, vol. E87-A, no. 03, pp. 682–691, 2004.

[49] R. Rabenseifner, G. Hager, G. Jost, and R. Keller, "Hybrid MPI and OpenMP parallel programming: MPI + OpenMP and other models on clusters of SMP nodes," *Tutorial M09 at SUPERCOMPUTING*, Nov 2008.

[50] B. R. Kandukuri, R. V. Puturi, and A. Rakshit, "Cloud security issues," *IEEE International Conference on Services Computing*, pp. 517–520, 2009.

[51] N. R. Putri and M. C. Mganga, "Enhancing information security in cloud computing services," *MSc Thesis, Blekinge Institute of Technology*, 2011.

[52] F. Hao, J. Dougman, and P. Zielinski, "A fast search algorithm for a large fuzzy database," *IEEE Transactions on Information Forescics and Security*, vol. 03, no. 02, pp. 203–212, June 2008.

[53] L. Hong and A. Jain, "Classification of fingerprint images," *Proceedingds of the 11th Scandinavian Conference on Image Analysis*, June 1999.

[54] I. B. Group. (Last accessed on 17 Sep 2011) The henry classification system. Internet draft. [Online]. Available: http://www.biometricgroup.com/

[55] N. Ratha, K. Karu, S. Chen, and A. Jain, "A real-time matching system for large fingerprint databases," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 08, Aug 1996.

[56] R. Thai, "Fingerprint image enhancement and minutiae extraction," *Honours Thesis, University of Western Australia*, 2003.

[57] N. K. Ratha and S. C. A. K. Jain, "Adaptive folw orientation based feature extraction in fingerprint images," *Pattern Recognition*, vol. 28, no. 11, pp. 1657–1672, 1995.

[58] S. Chikkerur, C. Wu, and V. Govindaraju, "A systematic approach for feature extraction in fingerprint images," *Biometric Authentication*, 2004. [Online]. Available: http://www.springerlink.com/index/7a63a3q9qf1p9ttt.pdf

[59] L. Hong, A. Jain, S. Pankanti, and R. Bolle, "Fingerprint enhancement," *IEEE Workshop on Applications of Computer Vision*, pp. 202–207, 1996.

[60] X. Jiang, "On orientation and anisostropy estimation for online fingerprint authentication," *IEEE Transactions on Signal Processing*, vol. 53, no. 10, pp. 4038–4049, Oct 2005.

[61] A. M. Bazen and S. H. Gerez, "Systematic method for the computation of the directional fields and singular points of fingerprints," *IEEE Transactions on Pattern Analysis And Mechine Intelligence*, vol. 24, no. 07, pp. 905–919, July 2002.

[62] M. Kass and A. Wikkin, "Analyzing oriented patterns," *Proceedings of the International Joint Conference Artificial Intelligence*, 1985.

[63] A. Almansa and T. Linderberg, "Fingerprint enhancement by shape adaptation scale-space operator with automatic scale selection," *IEEE Transations on Image Processing*, vol. 09, no. 12, pp. 2027–2042, Dec 2000.

[64] A. K. Jain, D. Prabhakar, and L. Hong, "Filterbank-based fingerprint matching," *IEEE Transactions on Image Processing*, vol. 09, no. 05, pp. 846–859, May 2000.

[65] Z. Guo and R. W. Hall, "Parallel thinning with two-subiteration algorithms," *Communications of the ACM*, vol. 32, pp. 359–373, Mar 1989.

[66] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, pp. 236–239, Mar 1984.

[67] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision,*, 3rd ed. Toronto: Thompson Learning, 2007.

[68] M. Behzad, G. Chartrand, and L. Lesniak-Foster, *Graphs and Digraphs*. Wadsworth International Group, 1979.

[69] R. de Cassia Nandi and A. L. P. Guedes. (Accessed 28 Aug 2012) Graph isomorphism applied to fingerprint matching. [Online]. Available: euler.mat.ufrgs.br/~trevisan/workgraph/regina.pdf

[70] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, Oct 2004.

[71] P. C. K. Kwok, "Thinning in a distributed environment," *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed processing*, pp. 257–263, Jan 1998.

[72] A. Wallcraft, P. Pacheco, and I. Foster. (Last accessed on 23 Aug 2011) Co-array fortran vs MPI. Internet draft. [Online]. Available: http://www.co-array.org/cafvsmpi.htm

[73] H. Wang, Y. Teo, and S. Tay, "An analytic method for predicting simulation parallelism," April 2000, pp. 211 – 218.

[74] J. L. Gustafson, "Reevaluating amdahl's law," *Communications of the ACM*, vol. 31, pp. 532–533, May 1988.

[75] X.-H. Sun and Y. Chen, "Reevaluating amdahl's law in the multicore era," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 183–188, 2010.

[76] J. H. Shaffer, "The effects of high bandwidth networks on wide area distributed systems," *PhD Thesis, University of Pennsylvania*, 1996.

[77] A. van Gemund, "Symbolic performance modelling of parallel systems," *IEEE Tans. on Parallel and Distributed Systems*, vol. 14, no. 02, pp. 154–165, 2003.

[78] D. Sundaram-Stukel and M. Vernon, "Perdictive analysis of the waveform application using LogGP," in *Proc. 7th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, vol. 34, no. 08, 1999, pp. 141–150.

[79] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eickev, "LogP: towards realistic model for parallel computation," in *Proc. 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, vol. 28, no. 07, 1993, pp. 1–12.

[80] J. Bourgeois and F. Spies, "Performance prediction of the NAS benchmark program with ChronosMix environment," in *Euro-Par '00: Proc. 6th Int'l Euro-Par Conf. on Parallel Processing*. Springer-Verlag, 2000, pp. 208–216.

[81] R. Saavedra and A. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM Trans. on Computer Systems*, vol. 14, no. 04, pp. 344–384, 1996.

[82] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: A generic framework for large-scale distributed experiments," *Proceedings of the 10th International Conference Computer Modelling and Simulation*, pp. 126–131, 2008.

[83] S. Prakash and R. L. Bagrodia, "MPI-SIM: Using parallel simulation to evaluate MPI programs," *WSC: Proceedings of the 30th Conference on Winter Simulation*, pp. 467–474, 1998.

[84] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the SimOs mechine simulator to study complex computer systems," *ACM Trans. on Modelling and Computer Simulation*, vol. 07, no. 01, pp. 78–103, Jan 1997.

[85] Y. Luo, "MPI performance study on the SGI origin 2000," *Pacific Rim Conf. on Communications, Computers and Signal Processing*, pp. 269–272, 1997.

[86] B. Cornea and J. Bourgeois, "Performance prediction of distributed applications using block benchmarking methods," in *Proceedings of the 19th International Euromicro Conference on Parallel, Distributed and Network-based Processing*, Feb 2011, pp. 183–190.

[87] S. Jarvis, D. Spooner, H. K. L. Choi, J. Cao, S. Saini, and G. Nudd, "Performance prediction and its use in parallel and distributed computing systems," *Future Generation Computer Systems*, vol. 22, no. 7, pp. 745–754, Aug 2006.

[88] S. Hammond, G. Mudalige, J. Smith, S. Jarvis, J. Herdman, and A. Vedgama, "WARPP - a toolkit for simulating high-performance parallel scientific codes," *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, vol. 41, no. 7, pp. 19:1–19:10, March 2009.

[89] R. Bagrodia, E. Deelman, S. Docy, and T. Phan, "Performance prediction of large parallel applications using parallel simulations," *ACM SIGPLAN 1999 Symposium on Principles and Practice of Parallel Programming*, pp. 151–162, May 1999.

[90] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *IEEE Computer*, vol. 41, no. 7, pp. 33–38, 2008.

[91] N. P. Khanyile, R.-J. Tapamo, and E. Dube, "An analytic model for predicting performance of distributed applications on multicore clusters," *IAENG International Journal of Computer Science*, vol. 39, no. 03, pp. 312–320, Aug 2012.

[92] N. Khanyile, R.-J. Tapamo, and E. Dube, "Performance prediction model for distributed applications on multicore clusters," *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engingeering 2012, WCE 2012*, vol. 02, pp. 1119–1124, July 2012.

[93] J. F. Kurose and K. W. Ross, *Computer Networks: A Top-Down Approach Featuring the Internet*. Addison-Wesley, 2001.