Parallel Algorithms for Iris Biometrics

Anukul Chandra Panda

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India

Parallel Algorithms for Iris Biometrics

Thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Technology

(Research)

in

Computer Science and Engineering

by

Anukul Chandra Panda

(Roll: 608CS403)

under the guidance of

Prof. Banshidhar Majhi

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India.

September 6, 2011

Certificate

This is to certify that the work in the thesis entitled ***Parallel Algorithms for Iris Biometrics*** by ***Anukul Chandra Panda*** is a record of an original research work carried out under our supervision and guidance in partial fulfillment of the requirements for the award of the degree of Master of Technology (Research) in Computer Science and Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Banshidhar Majhi
Professor
CSE department of NIT Rourkela

Acknowledgement

"The will of God will never take you where Grace of God will not protect you."
Thank you God for showing me the path...

I owe deep gratitude to the ones who have contributed greatly in completion of this thesis.

Foremost, I would like to express my sincere gratitude to my advisor, Prof. Banshidhar Majhi for providing me with a platform to work on challenging areas of parallel processing and biometrics. His profound insights and attention to details have been true inspirations to my research.

I am very much indebted to Prof. Sanjay Kumar Jena and Prof. Ashok Kumar Turuk for providing insightful comments at different stages of thesis that were indeed thought provoking.

My special thanks go to Prof. Dipti Patra, Prof. Gopal Krishna Panda and Prof. Bibhudatta Sahoo for contributing towards enhancing the quality of the work in shaping this thesis.

I would like to thank all my friends and lab–mates for their encouragement and understanding. Their help can never be penned with words.

Most importantly, none of this would have been possible without the love and patience of my family. My family to whom this dissertation is dedicated to, has been a constant source of love, concern, support and strength all these years. I would like to express my heartfelt gratitude to them.

Anukul Chandra Panda

Abstract

Iris biometrics involves preprocessing, feature extraction and identification phase. In this thesis, an effort has been made to introduce parallelism in *feature extraction* and *identification* phases. Local features invariant to scale, rotation, illumination are extracted using Scale Invariant Feature Transform (SIFT). In order to achieve speedup during feature extraction, parallelism has been introduced during *scale space construction* using SIMD hypercube. The parallel time complexity is $O(N^2)$ whereas sequential algorithm performs with complexity of $O(lsN^2)$, where l is the number of octaves, s is the number of Gaussian scale levels within an octave and $N \times N$ is the size of iris image.

During *identification*, search time plays a significant role. Indexing is done using *Geometric Hashing* of SIFT keypoints. This indexing approach achieves invariance to similarity transformations, illumination and occlusion. The traditional geometric hashing technique performs with a time complexity of $O(Mn^3)$, where M is the size of the database containing gallery images and n being the number of SIFT keypoints detected from an iris image. Here, the time complexity of geometric hashing approach is reduced by imbibing parallelism during calculation of geometric invariants and storage. SIMD hypercube architecture is used to perform computations in parallel. The time complexity of parallel geometric hashing approach is reduced to $O(Mn^2)$. During the retrieval phase, votes are cast to different gallery iris images. These vote counts are unordered which requires sorting to find the top k matches. So, parallel sorting (bitonic sorting) of vote counts using hypercube mesh architecture (HMA) is done to obtain the top k ranks among different gallery iris images. The bitonic sorting performs better than the other sequential sorting algorithms. Parity based bitonic sort through HMA is done to reduce the interprocessor communication by half. The parity based strategy finds the top k ranks with least amount of interprocessor communication. The time complexity of proposed sorting algorithm is $O(\log_2^2 M)$. Thus, the proposed iris biometric system is comparatively faster and could find its applicability in various real time scenarios.

Contents

Certificate ii

Acknowledgement iii

List of Figures vii

List of Tables ix

1 Introduction 1

1.1 Elements in Parallel Processing 3

1.1.1 Architectural Classification Schemes 3

1.1.2 Model of Communication for Parallel Platforms 6

1.1.3 Interconnection Networks for Parallel Computers 7

1.2 Literature Review 11

1.2.1 Preprocessing 12

1.2.2 Feature Representation 13

1.2.3 Identification 14

1.3 Problem Definition 16

1.4 Thesis Organization 16

2 Parallel Scale Space Creation of SIFT 19

2.1 Scale Invariant Feature Transform (SIFT) 20

2.1.1 Keypoint Detection 20

2.1.2 Keypoint Descriptor 24

2.1.3 Keypoint Pairing 25

2.2 Parallel Scale Space Construction using SIMD Hypercube 26
2.3 Asymptotic Analysis 30
2.3.1 Serial Scale Space Construction 30
2.3.2 Parallel Scale Space Construction 30
2.4 Summary 31

3 Parallel Geometric Hashing based Indexing 32
3.1 Serial Geometric Hashing based Indexing 33
3.1.1 Geometric Hashing 33
3.2 Parallel Geometric Hashing using SIMD Hypercube 36
3.2.1 Indexing Phase 36
3.2.2 Retrieval Phase 39
3.3 Asymptotic Analysis 41
3.3.1 Serial Geometric Hashing 41
3.3.2 Parallel Geometric Hashing 42
3.4 Summary 43

4 Rank Based Identification using Bitonic Sort 44
4.1 Related Works on Bitonic Sort 46
4.2 Proposed Bitonic Sorting using Hypercube Mesh Architecture 48
4.2.1 Non–Parity Strategy based Bitonic Sort 49
4.2.2 Parity Strategy based Bitonic Sort 55
4.3 Asymptotic Analysis 60
4.3.1 Non–Parity Strategy Based Bitonic Sort 60
4.3.2 Parity Strategy Based Bitonic Sort 61
4.4 Summary 61

5 Conclusions and Future Work 62

List of Figures

1.1 Computing the global sum of 3 partial sums using 4 processors 2
1.2 Flynn's Classification 5
1.3 Static Network Topologies 8
1.4 General Biometric System 10
1.5 Block Diagram of General Iris Biometrics 10

2.1 Scale space extrema 22
2.2 Maxima or minima of DOG images 23
2.3 Keypoint detection on annular iris image 24
2.4 Window is taken relative to direction of dominant orientation. This window is weighted by a Gaussian and histogram is obtained for 4×4 regions 25
2.5 A_1 and A_2 storing iris images with different sizes and Gaussian filters with different σ values, respectively 26
2.6 Mapping of Iris Images and Gaussian Filters to the Hypercube 27
2.7 Broadcast of Different Versions of Images 29
2.8 Broadcast of Gaussian Kernel 29
2.9 Final Configuration of Hypercube 29
2.10 Smoothed Images in $l \times s$ (2×4) matrix 30

3.1 An instance showing the robustness of geometric hashing to rotation, scaling and occlusion. 34
3.2 Geometric Hashing for Iris 35
3.3 Proposed Geometric Hashing 37

3.4 Mapping of detected keypoints to the shared global memory and 3D hypercube. 38
3.5 All-to-all broadcast is performed. [0,1,... n-1] represents the local memory of individual processors. 39
3.6 Parallel geometric hashing during iris retrieval phase. 43

4.1 Block diagram finding ranks of individuals using bitonic sort 45
4.2 Types of Comparators 46
4.3 Sorting network for 8 vote counts for decreasing sequence 48
4.4 Embedding Hypercube to Mesh 50
4.5 Knuth Diagram for eight vote counts (decreasing sequence) 51
4.6 Non–Parity Based Bitonic Sorting 54
4.7 Parity Based Bitonic Sorting 58

List of Tables

3.1 An example showing allocation of 8 keypoints to 8 processors and the number of computations performed by individual processors. 40

4.1 Inter–Processor Communication in Non–Parity Based Strategy 53
4.2 Local–Global Memory Communication in Parity Based Strategy . . . 59
4.3 Steps involved during Bitonic Sort 60

Chapter 1

Introduction

Most of the real time applications are computing intensive. To meet the response time requirements, high speed computers are essential. But the computing speed is limited by the sequential execution of the application. Hence, it is necessary to exploit concurrency in the execution of the application to achieve faster execution even with a single processor. In parallel computers, concurrency is maximally exploited to gain speedup. Various applications like weather forecasting, remote sensing, biometrics are associated with massive data inputs and non–linear processing. Further, they are hard real time systems and hence are better candidates of parallel processing. Parallel algorithms give a better output to the problems which can be decomposed into sub–problems. The main goal of decomposing a problem using a divide–and–conquer strategy is to enable the large task to be completed in less time. This improvement in time can be achieved by assigning each sub–task to the processors of a parallel architecture and execute them concurrently.

The performance gain due to the introduction of parallelism is generally measured by a metrics called *speedup*. Speedup is a measure that captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processor to the time required to solve the same problem on a parallel computer with p identical processors [1]. Mathematically it can be represented as,

$$speedup = \frac{T_{serial}}{T_{parallel}}$$

where, $T_{parallel}$ is the time complexity of the algorithm that solve the problem in parallel using p processors and T_{serial} is the time complexity of the sequential algorithm to solve the same problem.

To understand the importance of parallel algorithms, let us consider a problem to add n elements in parallel with n processors. The i^{th} element is mapped (distributed) to the i^{th} processor initially. The pairs of processors communicate among each other in parallel and the partial sum is stored in one of the processors having smaller processor label. The procedure is repeated until the final sum is computed. The pictorial representation of finding the sum of four elements using four processors in parallel is shown in Figure 1.1.

The time complexity (T_{serial}) for the sum of n elements using serial algorithm in a single processor system is $O(n)$ and that of parallel algorithm ($T_{parallel}$) using n processors is given by $O(\log_2 n)$ [1], *i.e.*,

$$\begin{aligned} T_{parallel} &= O(\log_2 n) \\ T_{serial} &= O(n) \end{aligned}$$

$$speedup = \frac{T_{serial}}{T_{parallel}} = \frac{O(n)}{O(\log_2 n)} \tag{1.1}$$

During parallel computation, interprocessor communication is always prevalent

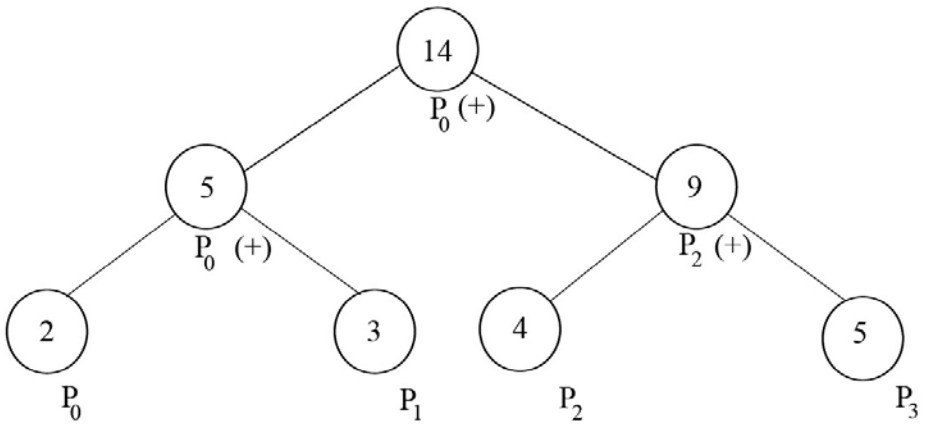


Figure 1.1: Computing the global sum of 3 partial sums using 4 processors

and becomes a bottleneck to the speedup gain. Hence, most of the present parallel processing research thrust upon reducing this communication overhead to maximize the speedup. A good parallel algorithm exploits the concurrency and implements the algorithm using an environment better suited for parallel programming. When an algorithm runs on a parallel architecture with multiple processors, the results of the computation are evident in less time.

However, the development of parallel algorithms presents unique challenges. The dependencies within concurrent tasks must be identified and correctly handled. In problem solving, a good parallel algorithm must be designed to ensure that nondeterministic issues do not affect the quality of the final output. Creating safe parallel programs can take considerable effort from the programmer. Even when a parallel program is "correct", it may fail to deliver the anticipated performance improvement from exploiting concurrency. Care must be taken to ensure that the overhead incurred by managing the concurrency does not overwhelm the program runtime. Also, partitioning the work among the processors in a balanced way is often not as easy as the summation of n numbers in parallel. The effectiveness of a parallel algorithm depends on how well it maps onto the underlying parallel computer. So, a parallel algorithm could be very effective on one architecture and may be disastrous on another [2].

In this thesis, parallel algorithms at various phases of an iris biometric system are proposed. Initially, inherent parallelism in the sub–steps is identified and subsequently, parallel algorithms are developed. For the sake of completeness, architectural classification schemes, various models and interconnection networks for parallel platforms are discussed briefly in Section 1.1 before discussing about literature review, problem definition and thesis organization.

1.1 Elements in Parallel Processing

1.1.1 Architectural Classification Schemes

There are three well–known architectural classification schemes like *Flynn's classification* which is based on the instruction streams and data streams in a computer or-

ganization, *Feng's classification* which is based on the serial and parallel processing and *Händler's classification* which determines the degree of parallelism and pipelining in a subsystem. From the above mentioned classifications, *Flynn's classification* is very popular and holds a standard [3].

Flynn's Classification

Computer organizations are characterized by the multiplicity of the hardware provided to service the instruction and data streams. Flynn's four machine organizations are listed below:

(*i*) Single instruction stream single data stream (SISD)

(*ii*) Single instruction stream multiple data stream (SIMD)

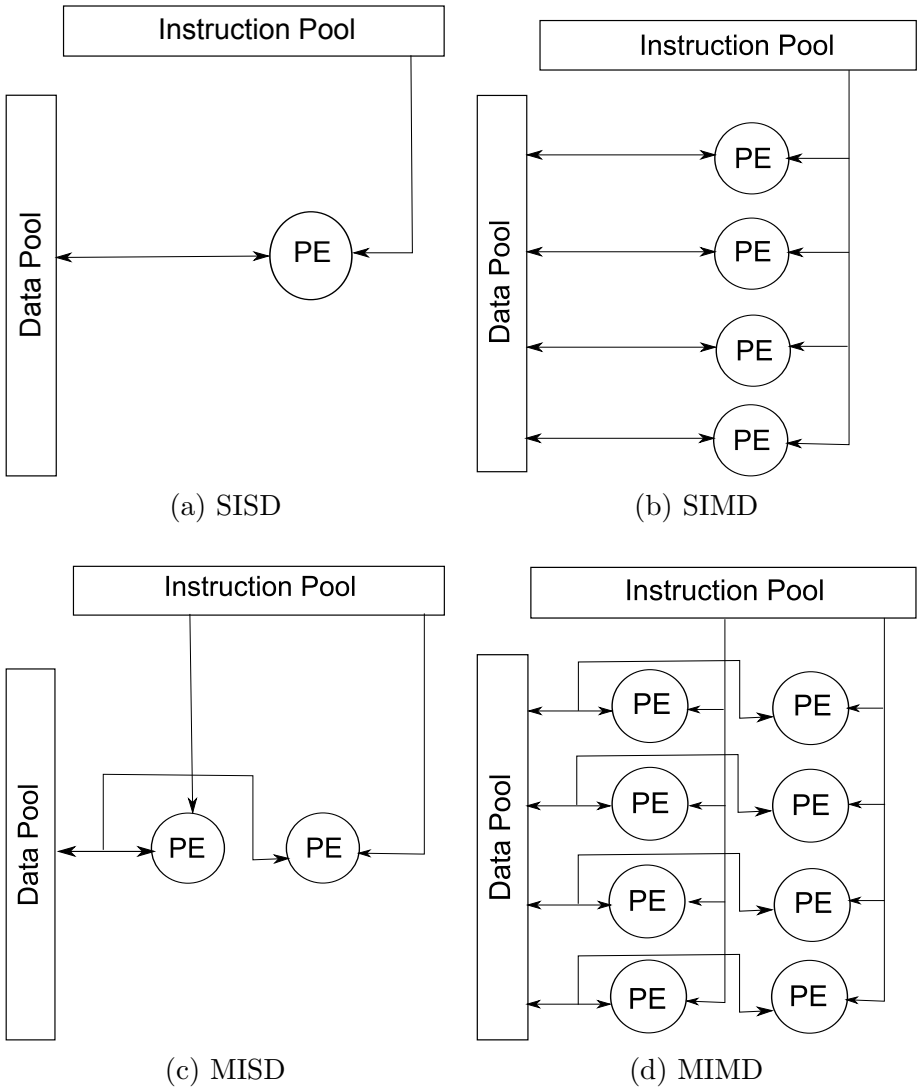
(*iii*) Multiple instruction stream single data stream (MISD)

(*iv*) Multiple instruction stream multiple data stream (MIMD)

The term *stream* represents a sequence of instructions or data as executed by a single processor. An instruction stream is sequence of instructions as executed by the machine. A data stream is a sequence of data including input, partial or temporary results used by the instruction stream. Generally, the memory modules provide the instructions and data. The control unit decodes the instructions and sends to the processing element (PE). The instruction stream and data stream are stored in the instruction pool and data pool, respectively. The flow between the data pool and the PE is bidirectional as shown in Figure 1.2 [3].

(*i*) SISD computer organization

Most of the serial computers fall under this category (Figure 1.2 (a)). Sequential execution of instructions takes place but they may be overlapped in the execution stages (pipelining). One control unit supervises all functional units. IBM 701, PDP VAX11/780, etc. are few computers falling under this category [3].



(a) SISD (b) SIMD

(c) MISD (d) MIMD

Figure 1.2: Flynn's Classification: (a) Single Instruction Stream Single Data Stream (SISD) (b) Single Instruction Stream Multiple Data Stream (SIMD) (c) Multiple Instruction Stream Single Data Stream (MISD) (d) Multiple Instruction Stream Multiple Data Stream (MIMD)

(*ii*) SIMD computer organization

A single instruction stream is concurrently broadcast to multiple processors, each with its own data stream (Figure 1.2 (b)). Most of the array processors are classified under this category. Illiac IV, BSP, PEPE [3] are the most well–known parallel architectures under this class.

(*iii*) MISD computer organization

There are multiple processing element, each receiving distinguishing instructions operating over the same data stream as shown in Figure 1.2 (c). The output of one processor become the input (operands) of the next processor. This structure has no practical utility and do not exist in physical world [3].

(*iv*) MIMD computer organization

Most multiprocessor systems and multiple computer systems fall in this category (Figure 1.2 (d)). When same data space is shared by all processors, then it is an MIMD computer system. This organization can be classified into two categories *i.e.* tightly coupled when degree of interactions among the processors is high and loosely coupled otherwise. The known architectures under this category are IBM 370/168 MP, Univac 1100/80, Tandem/16, etc [3].

1.1.2 Model of Communication for Parallel Platforms

There are two primary forms of data exchange between parallel tasks – accessing a shared data space and exchanging messages.

Shared–Address–Space Platforms

A common data space that is accessible to all processors is supported by shared–address–space of parallel platform. This shared address space is interacted by processors to modify data objects. Memory in these platforms can be local (exclusive to a processor) or global (common to all processors). The time taken by a processor

to access any memory word in the system is identical is called the Uniform Memory Access (UMA). On the other hand, if the time taken to access certain memory word is longer than others, the platform is called Non-Uniform Memory Access (NUMA).

It is important to note the difference between shared–address–space and shared memory parallel computers. The term *shared memory parallel computer* is used for architectures in which the memory is physically shared among all the processors, *i.e.*, each processor has equal access to any memory segment [1].

Message–Passing Platforms

The machine view of a message–passing platforms consist of p processors each with its own exclusive address space. Each of the processors can be either a single processor or shared–address–space multiprocessor. The interactions between processes running in different processors must be accomplished using messages; hence, the interaction is termed as message–passing. Message–passing paradigms support execution on each of the p processors [1].

These two models of communication are very well supported by different interconnection networks of parallel computers.

1.1.3 Interconnection Networks for Parallel Computers

Interconnection networks provide mechanisms for data transfer between processors or between processors and memory words. Links and switches are used to build interconnection networks. A link corresponds to physical media such as a set of wires of fibers capable of carrying data. Interconnection networks is generally classified into two types namely, ***static*** and ***dynamic***. Static networks consist of point–to–point communication links among processors and are otherwise called as ***direct*** networks.

Static networks are classified according to the dimensions. 1–D topologies include linear array whereas 2–D topologies constitute the ring, star, tree, mesh, etc. 3–D topologies contain the 3–cube–connected cycle network, 3–cube, etc. The mesh and the 3–cube are examples of 2–D and 3–D hypercube respectively [3]. A few widely used network topologies are shown in Figure 1.3.

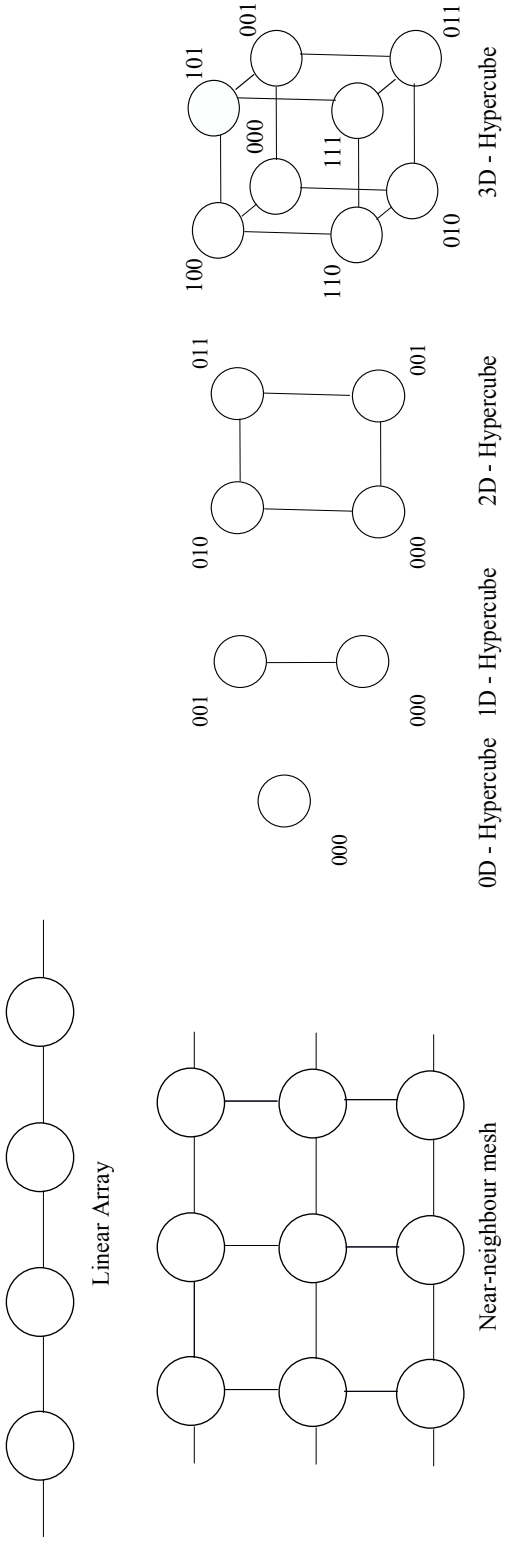


Figure 1.3: Static Network Topologies

Dynamic networks contain switches and communication links. Communication links are connected dynamically by the switches to establish a path among processors and memory words. Dynamic networks are otherwise known as ***indirect*** networks [1]. It is of two types– *single stage* and *multistage*. Data manipulator, Omega, flip n–cube, Benes network, Clos network, etc. are few well–known examples of this architecture. Although various architectures with different communication mechanisms are available, but they are implementation dependent. In turn, these implementations are dependent on parallel algorithms. It is the job of the parallel algorithm designer to map different data/tasks to a suitable architecture.

Parallel processing can be applied to many real time applications. Biometrics is one such application. It involves a substantial amount of image processing operations and hence, computation intensive. The processing steps are mostly sequential and interdependent. However, intra–stage computation involves concurrent tasks and hence is suitable for parallel processing. In this thesis, iris biometrics is considered as application to exploit the inherent parallelism.

Biometrics is the science of establishing the identity of an individual based on the physical or behavioral characteristics of the person. It is highly used to discover the identity of individuals in a group. Biometric traits can be categorized into two types, namely, behavioral and physiological. Behavioral characteristics related biometric deals with the behavior of a person. Gait, keystroke dynamics, signature and voice recognition are few well–known instances under this class. Physiological class helps in recognizing an individual based on anatomy of the body. Face recognition, fingerprint, palm print, DNA, hand geometry, iris recognition, etc are few examples. Characterization of a good biometric trait can be done based on its stability, uniqueness and robustness of the features. A generic biometric system operates by taking an input image from the user, preprocessing the image to find region of interest, extracts features, and enrolls/matches the features [4]. A general biometric system operates in two stages i.e., enrollment and matching. During enrollment the acquired image (gallery image) is preprocessed to extract region of interest, features are extracted and stored in the database. During matching stage, the features from the probe image

are compared with already stored features to find the potential match. The block diagram of a general biometric system is shown in Figure 1.4.

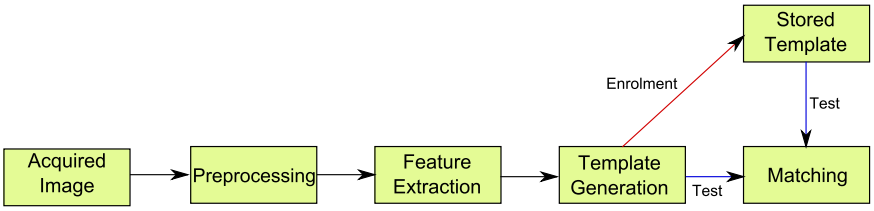


Figure 1.4: General Biometric System

Every biometric trait has its own strengths and weaknesses and it is also dependent on the application scenario. The reliability of a particular biometric trait is relied on its ability to extract the unique features in an invariant manner. For instance, the uniqueness of fingerprint features are evident over a passage of time while the features extracted from face though unique may vary due to its ability to undergo change in viewing angle, illumination and age [5].

Iris biometric is one such popular trait used worldwide due to its robustness. Amongst various available biometric traits, iris provide a promising solution to recognize an individual using unique texture patterns [6]. Iris has been proved to work efficiently where reliability and invasiveness is a major concern. Iris is a protected internal organ whose random texture is stable throughout lifetime proving its resistance to invasiveness. Recognition decisions are made with confidence levels high enough to support rapid and reliable exhaustive searches through national level databases.

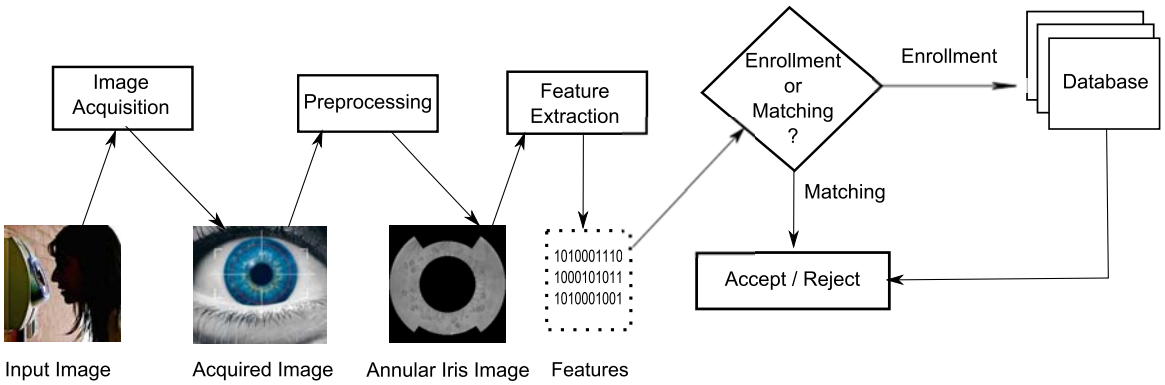


Figure 1.5: Block Diagram of General Iris Biometrics

To generate and store biometric template in the database, image processing tech-

niques can be employed to extract unique iris features from acquired image. This biometric template contains a mathematical representation of unique texture information stored in the iris, and allows comparisons to be made between individuals. When identification of an individual is to be done by an iris recognition system, their eye is first photographed, and then a template is extracted for their iris region. This template is then compared with its other counterparts stored in a database to generate the identity of an individual [6]. A general iris biometric system is shown in Figure 1.5.

It can be observed that in any biometric system stages like preprocessing, feature extraction and recognition/verification. The accuracy of any biometric system depends on the accuracy of these phases performed in sequel. Various schemes suggested by researchers are discussed below in the following section.

1.2 Literature Review

Originally Flom and Safir proposed the idea of automated biometric system in 1987 [7]. The illumination has been changed to make pupil of predetermined size to restrict the variation in the size of iris. In addition, the authors have suggested some crucial benchmarks that have influenced the research later. Pattern recognition tools have been proposed by them to extract iris features and an initial method of finding pupil has been suggested using static threshold. Daugman developed the first operational iris biometric system at University of Cambridge in 1994 [8]. To control the change in illumination, near infrared light source is used for iris acquisition. The next step is to locate the iris in the image that uses deformable templates. A deformable template is trained with some parameters and the shape of eye, to guide the detection process [9]. Daugman assumed iris and pupil boundaries to be a circular in nature. Thus, the boundary of circle can be described with three parameters: radius r, center of the circle (x_0, y_0) [5]. The operator is defined as

$$max(r, x_0, y_0)|G_\sigma(r) * \frac{\partial}{\partial r} \oint_{r,x_0,y_0} \frac{I(x,y)}{2\pi r} ds| \tag{1.2}$$

where $G_\sigma(r)$ is a smoothing kernel and $I(x, y)$ is the image of the eye. The operator searches over the image domain (x, y) for the maximum in the blurred partial derivative with respect to increasing radius r of the normalised contour integral of $I(x, y)$ along a circular arc ds of radius r and centre coordinates (x_0, y_0). After the segmentation of iris , the next step is feature description of iris for comparison. The variation of the size of iris is a major challenge in iris comparison. The iris representation should be consistent to change in size, scale, orientation, etc. The iris pattern undergoes linear deformation due to variation in illumination that causes pupil to expand or contract and change in orientation of iris due to head tilt, camera position, movement of eyeball, etc. Daugman has overcome this problem by mapping iris into dimensionless polar coordinate system [5].

An iris biometric system was developed at Sarnoff labs with a variation to Daugman's approach. For image acquisition, diffused source of light with low level light camera has been used. Pupil and iris segmentation is done using Hough transform. For matching of two iris images, the Laplacian of Gaussian filter at multiple scales have been used by the system to produce template and computes normalised correlation as a similarity measure [10]. Significant research has been done in these three models of iris recognition which are laid by Flom and Safir, Daugman and Wildes. This section discusses in detail about the different areas like feature representation and identification where the concept of parallelism is introduced.

1.2.1 Preprocessing

Preprocessing of iris involves the detection of pupil and iris boundaries which is assumed to be of circular shape. However, few authors have also worked to further improve localization performance by detecting eyelids/eyelashes [11]. Coarse–to–fine strategy is proposed by Huang et. al [12] to improve the iris localization time. This technique detects outer iris boundary in the rescaled image and information regarding iris circles are found. Further many authors have proposed the method to detect eyelids and eyelashes. Searching of two curves that satisfies polynomial equation of the form $x(t) = at^2 + bt + c$, $t \in [0, 1]$ helps in detecting eyelids. Checking variance

for each block detects eyelashes.

Various schemes have been developed as an improvement over traditional Hough transform. The authors have used canny edge detector with Hough transform to improve localization speed in [13]. Normal line algorithm is created using canny edges for detecting center and inner edge. Homocentric circle algorithm is used to detect outer edge. The authors in [14] have used bisection method to find inner boundary. Some authors have used thresholding based approaches to find coarse localisation of pupil. Pixels below a threshold is searched as pupil in [15] and circles in the limited area is found using Hough transform and edge detection. Further, an automatic iris segmentation based on local areas is proposed in [16]. In this approach, iris image is divided into rectangular grid and for each block mean is obtained.

1.2.2 Feature Representation

Features can be of two types, global and local. There already exists several global feature extraction techniques for iris [17, 18]. The main drawback of global techniques is that they fail to extract relevant features if there exists significant variations in pose, illumination and viewpoint of an individual. Local features are invariant to image scaling and rotation, and partially invariant to change in illumination and viewpoint. These local features have the capability to perform well under partial occlusions as well. In order to extract local features from iris, special points known as *keypoints* are detected where there can be a corner, an isolated point of local intensity maximum or minimum, line endings, or a point on a curve where the curvature is locally maximal. Around the neighborhood of every detected keypoint a descriptor is taken that represents the feature vector. There are various detector–descriptor schemes to extract local features like Harris corner detector [19], Scale Invariant Feature Transform (SIFT) [20], Speeded Up Robust Features (SURF) [21], etc. The Harris corner detector to locate feature points in images has been proposed by Harris and Stephens. Harris corner detector works brilliantly under occlusion but fails if the image has undergone scale change. David G. Lowe introduced a keypoint detector and feature descriptor scheme to detect and describe local features called SIFT (Scale Invariant

Feature Transform). This feature extraction technique which is based on scale space is consistent to image scaling, rotation and affine transformation, and partially invariant to change in illumination and view angle [20]. This local feature extraction technique contains various steps. The steps are construction of scale space, finding Difference of Gaussian (DOG) images, keypoint detection, keypoint descriptor assignment and keypoint pairing.

1.2.3 Identification

During identification, the comparison is made between probe iris image with enrolled iris in the database. This requires substantial time and needs indexing of the overall database so that it is partitioned based on some criteria and comparison is made with a partial subset Indexing hand geometry database using pyramid technique has been proposed in [22]. It has been claimed to prune the database to 8.86% of original size with 0% false rejection rate. In [23], an efficient indexing scheme for binary feature template using B+ tree has been proposed. In [24], a modified B+ tree indexing for biometric database indexing has been proposed. The higher dimensional feature vector is projected to lower dimensional feature. The reduced dimensional feature vector is used to index the database by forming B+ tree. Further, an efficient indexing technique that can be used in an identification system with large multimodal biometric database has been proposed in [25]. This technique is based on kd–tree with feature level fusion which uses the multi–dimensional feature vector. In [26], two different approaches of iris indexing have been analysed. First one uses the iris code while second one is based on features extracted from iris texture. In [27], an iris indexing technique based on the iris color for noisy iris images is proposed. The performance measures prove the effectiveness of iris color for indexing very large database. A robust iris indexing approach has been proposed using geometric hashing of Scale Invariant Feature Transform (SIFT) keypoints [28]. To identify the top k matches during the matching, sorting of the vote counts of the gallery iris images which have received votes due to the hashing of the probe iris image into the hash table is needed.

From the existing literature, it has been discovered that iris biometrics deals with

various features and different matching schemes. In the present thesis, SIFT features extracted from annular segmented iris is considered. For identification, geometric hashing is utilized. The overall processing is composed of various stages and each stage further consists of different phases. They are briefly discussed below,

(a) Preprocessing of the acquired image to find region of interest i.e. *iris* [29].

(b) Extraction of SIFT features.

(c) Generation of hash bins during enrollment and matching of probe image is done by using geometric hashing based indexing scheme.

(d) The top k best matches with respect to vote is found using rank based identification.

To understand the processes better, each stage is elaborated as,

- Preprocessing
  - (a) Removal of specular highlights
  - (b) Localization of Iris (*i.e.*, Pupil and Iris detection)
  - (d) Removal of eyelids by considering *sectored annular region*
- Extraction of SIFT Features
  - (a) Construction of Scale Space
  - (b) Finding Difference of Gaussian (DOG) images
  - (c) Location of DOG extrema
  - (d) Detection of keypoints
  - (e) Filtering of edge and low contrast responses
  - (f) Orientation assignment to keypoints
  - (g) Building keypoint descriptors
- Geometric Hashing based Identification

(*a*) Indexing of gallery images (*i.e.*, Indexing of Geometric Invariants)

(*b*) Retrieval of potential matches (*i.e.*, Indexing of Geometric Invariants, finding top k matches)

1.3 Problem Definition

It has been observed that each stage of the processing itself deals with a good amount of computation and requires accuracy as well. Almost all the stages are performed sequentially. Researchers have proposed various schemes in each stage of processing [20, 29]. A sequential version of geometric hashing followed by rank based identification is proposed in [28]. The performance of such a system is well accepted when the size of database is small. But as the size of database increases, the response time for an identification or verification increases further. In such a situation, performance improvement becomes a challenging task. Alternatives to handle this are to use high speed devices. But its performance is again limited by the sequential execution of each stage. Hence the remaining option is to exploit the inherent parallelism if exists. In this thesis, parallel activities in different phases of the biometric system are identified and parallel algorithms are devised for those phases with a SIMD hypercube.

After rigorous study the following phases are identified to be the candidates for parallel processing,

(*i*) Construction of scale space extrema.

(*ii*) Computation of geometric invariants and mapping of SIFT keypoints into the hash table.

(*iii*) Finding the top k best matches using bitonic sort.

1.4 Thesis Organization

The thesis is organized as follows.

Parallel scale space creation to extract SIFT keypoints is discussed in **Chapter 2**. The first step has inherent parallelism. The keypoints are generally extracted at different scales to get sufficient number of keypoints. The scale space representation consists of octaves. Each octave again contains different smoothed images, L. These smoothed images, L are obtained at different values of sigma (σ). In [20], these smoothed images were found in serial. In this chapter, an effort has been made to parallelize the computation of the smoothed images, L using SIMD hypercube.

Chapter 3 deals in the identification phase after extraction SIFT keypoints. This chapter discusses parallel geometric hashing approach. Geometric hashing is an indexing approach for model based object recognition that uses location of keypoints which are invariant to similarity transformation as an index to the hash table [30, 31]. Let, n keypoints are detected using SIFT. The key advantage of this indexing based approach is that this technique possesses invariance to various possible transformations and occlusion. Geometric hashing based approach has two stages: indexing and recognition phase. In this chapter, parallel calculation of the geometric invariants is made using SIMD hypercube. These geometric invariants are indexed to the hash table in parallel.

Chapter 4 discusses the rank–based identification of an individual during the retrieval phase of the geometric hashing. After the mapping of geometric invariants of the probe iris image, the gallery images receive votes. The number of votes that can be cast to a gallery iris image during retrieval phase is in the range of $[0, {}^nC_2(n-2)]$, where, n denotes the number of keypoints present in the probe iris image. The number of gallery iris images which can receive votes are within a range $0 \leq m \leq M$, with M being the total number of gallery iris images. Now these vote counts are generally unordered. So, a parallel sorting can often be used to find top k matches. Vote counts are sorted using both non–parity and parity based strategy to bitonic sorting algorithm using general–purpose parallel architecture called as hypercube mesh parallel architecture.

Finally, **Chapter 5** gives the concluding remarks.

Chapter 2

Parallel Scale Space Creation of SIFT

Feature extraction involves simplifying the amount of information required to describe an iris image. The purpose is real time, high confidence recognition of an individual's identity by mathematical analysis of the random patterns that are visible within the iris from some distance. There are several feature descriptors like Scale Invariant Feature Transform (SIFT) [20], PCA–SIFT [32], Speeded Up Robust Features (SURF) [33], etc. which have served brilliantly in describing the features within the iris. Among available feature extraction techniques, SIFT outperforms its counterparts due to the following reasons [34].

1. The accuracy of matching is highest for an affine transformation of 50° compared to other descriptors.

2. SIFT based descriptors performs with higher accuracy on the textured images.

3. It possesses strong invariance to blurring.

SIFT has three different phases namely, keypoint detection, keypoint descriptor assignment and keypoint pairing. In keypoint detection phase, there are various steps like detection of scale space extrema, keypoint localization and orientation assignment. It has been observed that there exists an inherent parallelism during scale space construction. A parallel algorithm for this computationally intensive step has

been developed using SIMD hypercube parallel architecture. An asymptotic time complexity is obtained for serial and parallel implementation. The sequential phases involved in SIFT are discussed in Section 2.1. Section 2.2 describes the proposed parallel scale space algorithm. The comparative time complexity analysis is discussed in Section 2.3.

2.1 Scale Invariant Feature Transform (SIFT)

Scale Invariant Feature Transform (SIFT) is a keypoint detector and feature descriptor scheme. This algorithm works in applications like object recognition, robotic mapping and navigation, biometrics, image stitching, 3D modeling, gesture recognition, video tracking, and match moving. Owing to its advantages SIFT could find its applicability in biometrics [29, 35, 36].

The local features from an iris image are computed using cascade filtering approach that minimizes the feature extraction cost by applying more expensive operations at locations that pass an initial test. The feature vector is generated by performing the phases as outlined in the following subsections:

2.1.1 Keypoint Detection

The keypoint detection begins with finding potential keypoints that are invariant to scale and orientation. For each detected keypoint the location and scale is determined. Image gradients help in assigning orientations to each location. The steps involved in keypoint detection are outlined with explanation in the following subsections.

Detection of Scale Space Extrema

The first step of keypoint detection is to find the positions and scales that can be iteratively assigned under different viewing of same object. Identification of locations consistent to scale change of the image can be found by exploring stable features across different scales using a continuous function of scale known as *scale space* [20]. The only possible scale space mask is the Gaussian function. To define the scale space,

input iris image (I) is convolved with Gaussian kernel $G(x, y, \sigma)$ as defined by

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.1}$$

where $*$ is the convolution operation and σ defines the width of Gaussian filter. The smoothed images, $L(x, y, \sigma)$ is obtained by convolving the Gaussian kernel with the iris image. The Difference of Gaussian (DOG) images are computed from two nearby scales differentiated by constant multiplicative factor k

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{2.2}$$

The scale space is divided into various *octaves*. An octave is represented by a series of smoothed image, L which is obtained by convolving the iris image with Gaussian with different values of σ varied by a constant factor. The total number of Gaussian scale levels within an octave are denoted by s. A constant number of Gaussian scale levels are found in each octave. The subsequent octave is obtained by downsampling the input iris image size by half and generating the Gaussian scale levels using equation 2.1 as shown in the Figure 2.1. This step is iterated for l such octaves. The serial scale space construction algorithm is given in Algorithm 1. This technique is found to be conducive for annular iris images since the size of iris is viable to change owing to pupil contraction and expansion. The parallelism can be exploited both within octave (Gaussian scale level computation) and across octave simultaneously.

Algorithm 1: Serial Scale Space Construction

Result: Serial Computation of Smoothed Image, L

```
1 for Each Octave (l) do
2     for Each Scale (s) do
3         Convolve input image with Gaussian kernel
4     Downsample the input image by half
```

Firstly, the calculation of the smoothed image, L (Gaussian scale levels) within the octave are found with changed values of σ. Each Gaussian scale level is independent

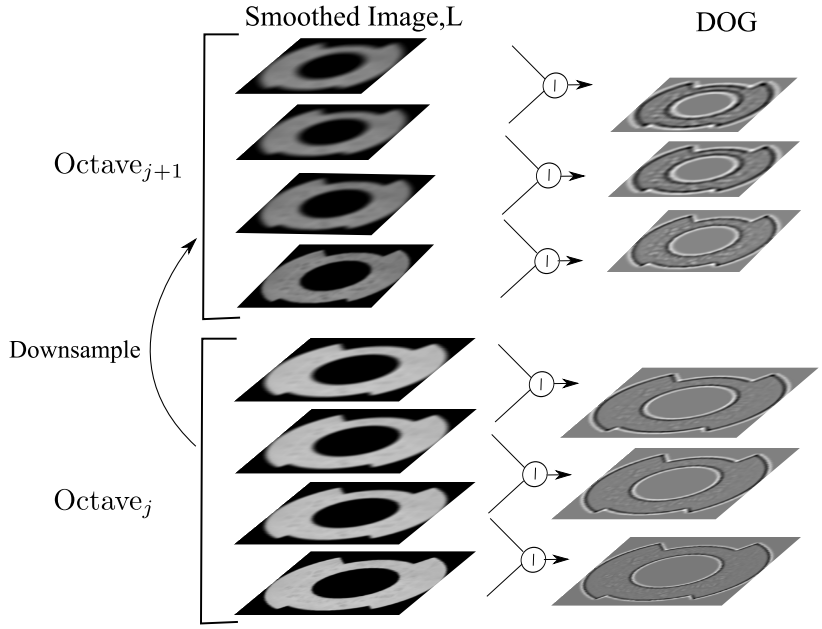


Figure 2.1: Scale space extrema for different octaves. Adjacent Gaussian images are subtracted to produce DOG images on right

of the other since the smoothed image is found for varied values of σ and hence, L is found in parallel using different processors. The same operation is carried out for change in scales within all l octaves.

Secondly, the parallelism can also be introduced across octaves. The present octave always receives the iris image with half the image size of the previous octave. This dependency can also be avoided by performing parallel computation for l octaves. These two parallelism can simultaneously be exploited by taking a SIMD Hypercube which is discussed in detail in Section 2.2.

Keypoint Localization

DOG iris images which are found in different octaves are used to detect landmark points with the help of local maxima or minima across different scales. Each sample point in DOG image is compared to eight neighbors in current image and nine neighbors in the scales (above and below). The sample point is chosen as a candidate keypoint that is either a local maxima or minima in $3 \times 3 \times 3$ regions at current and adjacent scales as shown in Figure 2.2. Once the candidate keypoints are detected the next step is to reject keypoints with edge and low contrast responses. In [20], it

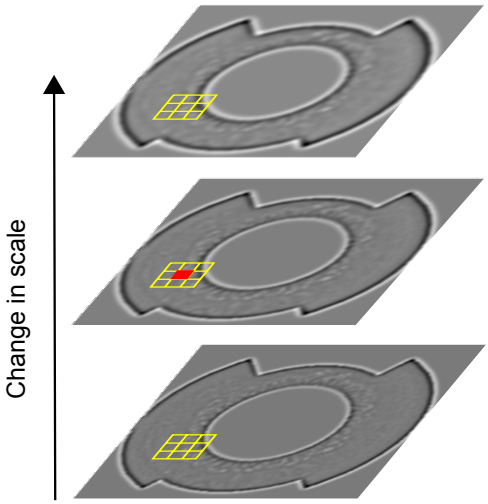


Figure 2.2: Maxima or minima of DOG images are obtained by comparing sample point (marked as red) to 26 neighbors in $3 \times 3 \times 3$ region

has been found that keypoints with low contrast are sensitive to noise or are poorly localized, hence they should be discarded.

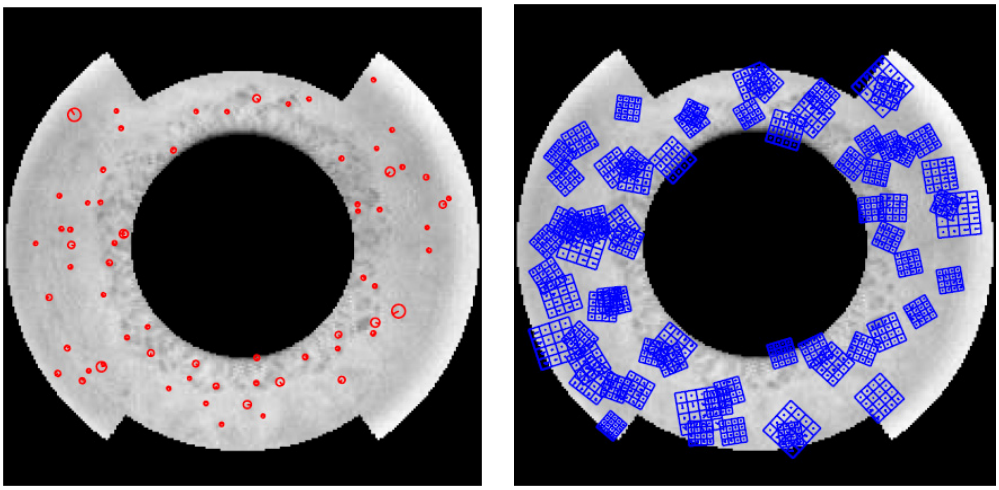
Orientation Assignment

In the previous step, an effort has been made to make features invariant to scale. However, in order to mitigate the effect of rotation, orientation assignment is used. In this step, local image gradient is used to assign one or more orientations to the keypoints. In this step, the keypoint descriptor is represented with respect to the orientation and therefore achieves invariance to image rotation. A gradient orientation histogram is computed in the neighbourhood of keypoint to determine keypoint orientation. The scale of keypoint is used to select Gaussian smoothed image L for achieving the computation in scale invariant manner. For each Gaussian smoothed image $L(x, y)$, magnitude $m(x, y)$ and orientation $\theta(x, y)$ are computed using the pixel level differences as

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \quad (2.3)$$

$$\theta(x,y) = \tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \quad (2.4)$$

The magnitude and orientation calculations for the gradient are done for every



(a) (b)

Figure 2.3: Keypoint detection on annular iris image using SIFT (a) Detected keypoints at different scales, (b) Scale and orientation of keypoints are indicated by arrows

pixel in a neighboring region around the keypoint in the Gaussian–blurred image L. An orientation histogram is formed with 36 bins such that each bin covers 10 degrees. Each sample in the neighboring window is weighted by gradient magnitude and Gaussian weighted circular window with σ of 1.5 times of scale of keypoint before adding it to orientation histogram. The peaks in this histogram correspond to dominant orientations. Once the histogram is filled, the orientations corresponding to the highest peak and local peaks that are within 80% of the highest peaks are assigned to the keypoint. This is done to increase stability during matching [20].

2.1.2 Keypoint Descriptor

Once orientation has been selected, the feature descriptor vector for each keypoint is computed such that the descriptor is highly distinguishing and partly consistent even under change in illumination, viewpoint, etc. Initially, a set of orientation histograms on 4×4 pixel neighborhoods are created. The orientation histograms are relative to the keypoint orientation as shown in Figure 2.4. The histogram contains 8 bins each and each descriptor contains an array of 16 histograms around the keypoint. This generates SIFT feature descriptor of $4 \times 4 \times 8 = 128$ elements.

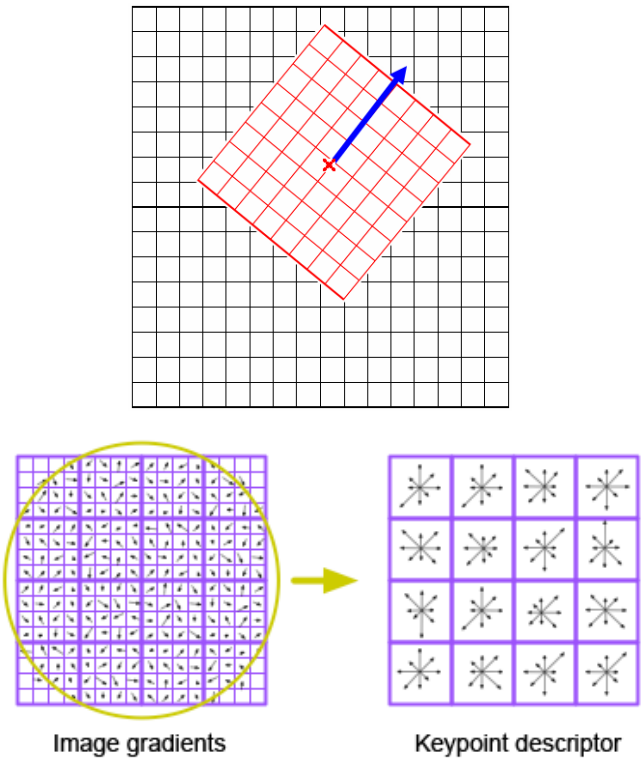


Figure 2.4: Window is taken relative to direction of dominant orientation. This window is weighted by a Gaussian and histogram is obtained for 4×4 regions

2.1.3 Keypoint Pairing

Let $p = \{p_1, p_2, p_3...p_n\}$ and $q = \{q_1, q_2, q_3...q_n\}$ be n dimensional feature descriptor for each point from gallery and probe images respectively. The Euclidean distance between p and q is defined as,

$$D(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2.5}$$

where n is 128 dimensional feature descriptor. The naive approach to nearest neighbor matching is to simply iterate through all points in the database to determine the nearest neighbor.

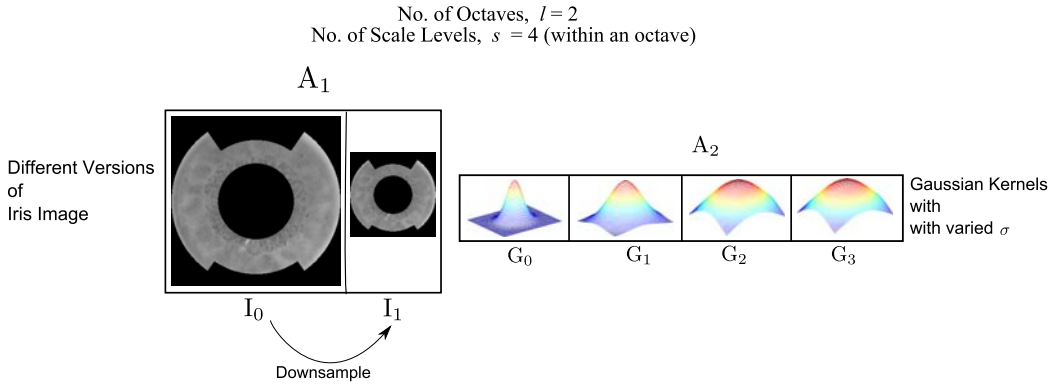


Figure 2.5: A_1 and A_2 storing iris images with different sizes and Gaussian filters with different σ values, respectively

2.2 Parallel Scale Space Construction using SIMD Hypercube

During scale space creation, there are various octaves which divides the scale space. Each octave contains different Gaussian scale levels. Each Gaussian scale level produces smoothed image, L by the convolution of iris with Gaussian kernel at a particular scale (σ). For instance, the L_i^{th} smoothed iris image is obtained by the convolution of input iris image (I_j) with Gaussian kernel (G_i) at scale (σ_i), where $0 \leq i \leq s-1$ and $0 \leq j \leq l-1$. This same operation is iterated across different octaves to obtain L . Octave_{j+1} receives the input image size that is half the input iris image size of Octave_j. The size of iris image and the number of octaves is always known a priori to execution. So, the inter–dependency between the octaves is eliminated by storing the different sizes of iris image in an array.

The $(p+1)^{th}$ position stores half the iris image size stored in p^{th} position of the array (A_1) and the Gaussian kernels for change in σ are stored in array (A_2). So, A_1 stores l versions of iris images based on the number of octaves. Similarly, s Gaussian filters with different σ values are stored in A_2 based on the number of Gaussian scale levels (s). The states of array A_1 and A_2 for two octaves and four Gaussian scale levels within the octave are shown in Figure 2.5.

The scale space representation with s Gaussian scale levels for l octaves can be constructed using an ls–processor hypercube (assume, $ls = 2^x$). The application of

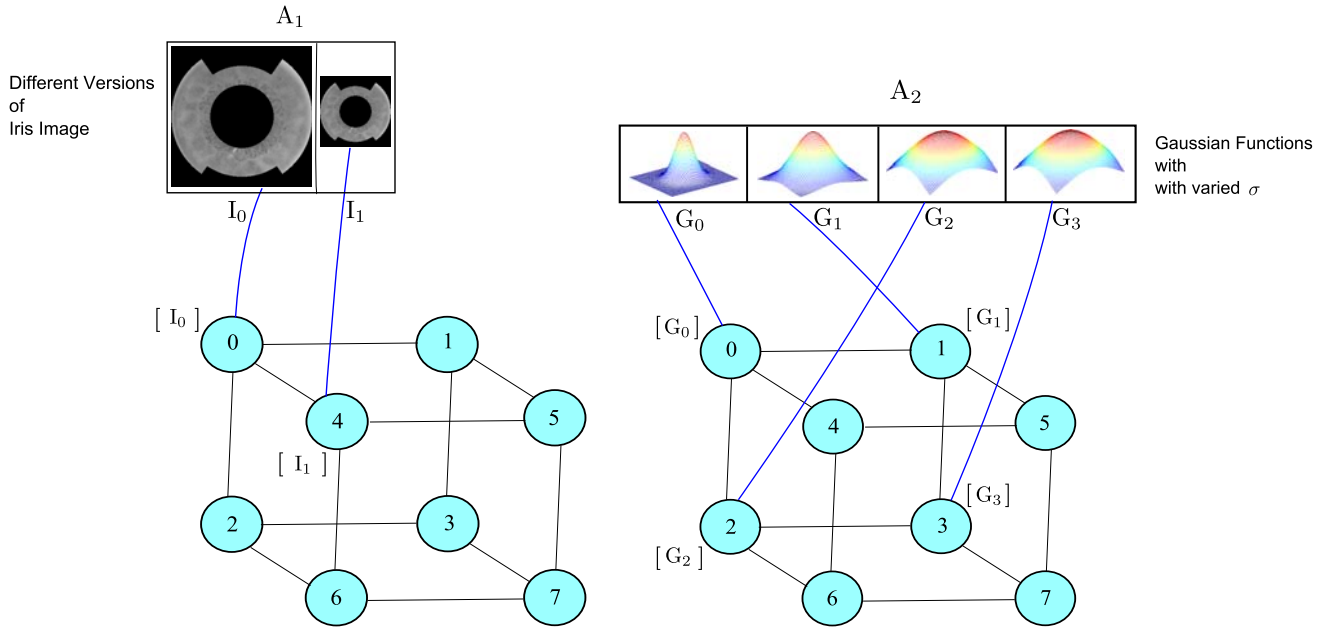


Figure 2.6: Mapping of Iris Images and Gaussian Filters to the Hypercube

hypercube facilitates in achieving both the parallelism simultaneously. The iris image labeled as j is mapped to the $((j+1)s-s)^{th}$ processor of the hypercube. Similarly, i^{th} Gaussian filter is mapped to the i^{th} processor of the hypercube. Figure 2.6 shows the mapping of iris image and Gaussian filters for two octaves and four Gaussian scale levels within octave. The content of $((j+1)s-s)^{th}$ processor of the hypercube is broadcast to processors spanning from $((j+1)s-(s-1))$ to $((j+1)s-1)$ using recursive doubling. The broadcast takes $\log_2 s$ steps. Similarly, i^{th} Gaussian filter present in i^{th} processor is broadcast to the processor with empty local memory having label of the processor as i after performing the operation *i.e.*, $(i \ \% \ s)$. The broadcast takes $\log_2 l$ steps using recursive doubling mechanism.

Algorithm 2: Parallel Scale Space Construction

Result: Parallel Computation of Smoothed Image, L

1 Store the iris images in array A_1

2 Store the Gaussian filters in array A_2

3 Map j^{th} iris image to $((j+1)s-s)^{th}$ processor of the hypercube

4 Map i^{th} Gaussian filter to i^{th} processor of the hypercube

5 **if** $y \leq$ *((j + 1)s -1) and* $y \geq$ *(j + 1)s - (s+1)* **then**

6 Perform broadcast of iris image at $((j+1)s-s)^{th}$ to y^{th} processor using recursive doubling

7 **if** *(i mod s)* $== i$ **then**

8 Perform broadcast of the i^{th} Gaussian filter to the $(i \ mod \ s)^{th}$ processor using recursive doubling

9 Convolve iris image with Gaussian filter in parallel in the hypercube

10 Store the calculated smoothed image, L in a $l \times s$ matrix

The broadcast of images and Gaussian kernels among the processors are shown in Figure 2.7 and 2.8 respectively. The 0^{th} iris image is broadcast to 1^{st} processor as shown in Figure 2.7 (a). Later 0^{th} and 1^{st} processor concurrently broadcast their content to 2^{nd} and 3^{rd} processors as shown in Figure 2.7 (b). The same communications can be performed for other images simultaneously in other octaves. The communication of Gaussian filters across different processors can performed in similar way. These communications across processors take 2 (*i.e.*, $\log_2 4$) and 1 (*i.e.*, $\log_2 2$) steps for broadcasting images and Gaussian kernels respectively. Figure 2.9 shows the final configuration of the processors. Since each processor contains iris image, I and the Gaussian filter, G, hence, the convolution take place in each processor concurrently yielding different smoothed image L. After obtaining the smoothed images they are stored in an $l \times s$ matrix as shown in Figure 2.10 and from which the DOG images are found in serial. Algorithm 2 outlines the steps involved in parallel scale space construction.

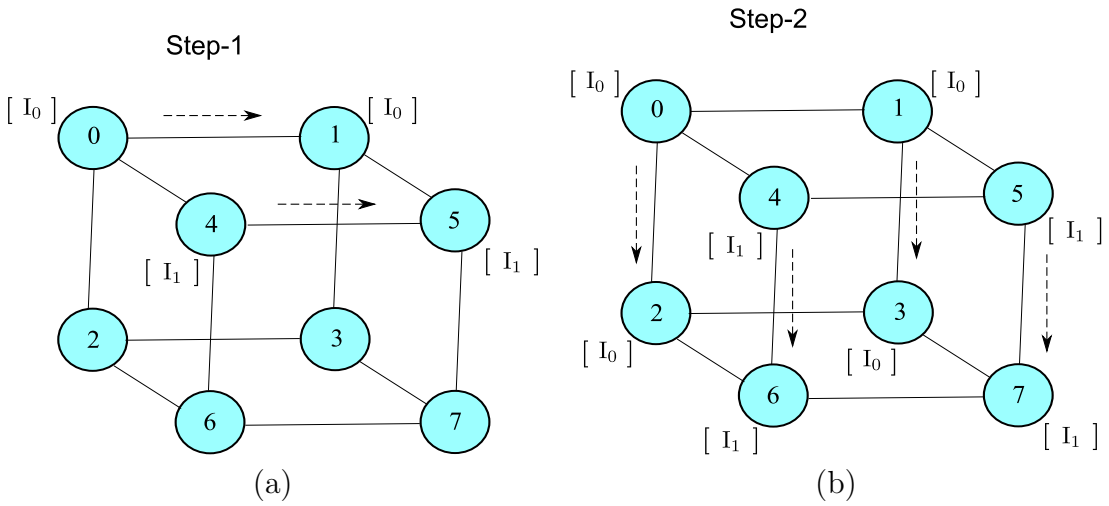


Figure 2.7: Broadcast of Different Versions of Images

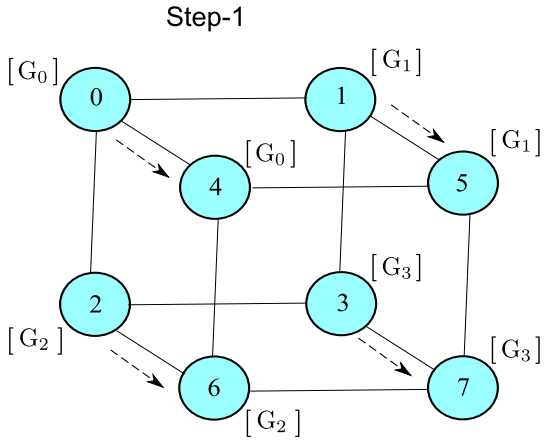


Figure 2.8: Broadcast of Gaussian Kernel

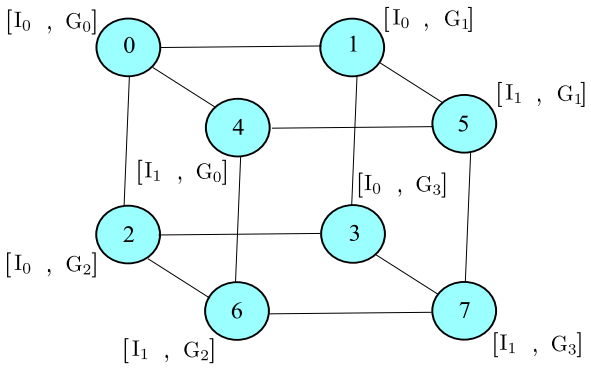


Figure 2.9: Final configuration of hypercube after broadcast. The local memory contains the iris image and Gaussian Filter required to find smoothed image, L

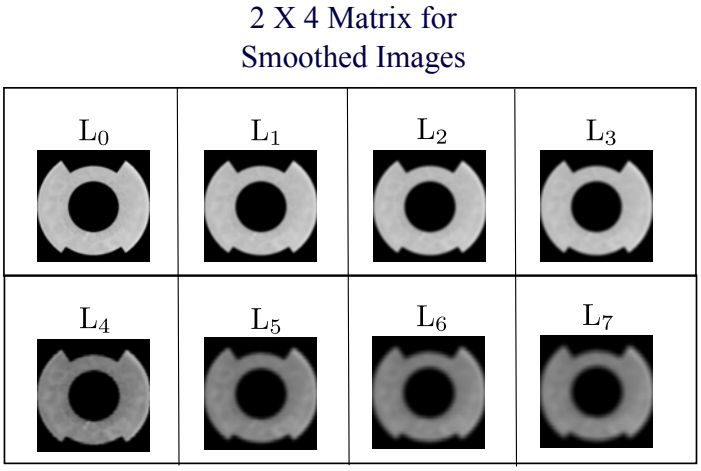


Figure 2.10: Smoothed Images in $l \times s$ (2×4) matrix

2.3 Asymptotic Analysis

2.3.1 Serial Scale Space Construction

The construction of the octaves involves the following steps:

1. The calculation of smoothed iris image, L involves a convolution at a value σ. The convolution generally takes a time complexity of $O(N^2)$, where $N \times N$ is the size of the input iris image.

2. The calculation of s Gaussian smoothed iris image, L take place at different values of σ which is changed by a constant factor. So, the time complexity now becomes $O(sN^2)$.

3. The above two operations are iteratively applied across l octaves.

Hence, the total time complexity of these serial operations denoted as T_{serial} is $O(lsN^2)$ i.e., $T_{serial} = O(lsN^2)$

2.3.2 Parallel Scale Space Construction

The time complexity of parallel octave creation with l octaves and s scale levels using SIMD hypercube is given as

$$\begin{aligned} T_{parallel} &= \log_2 l + \log_2 s + N^2 \\ &= O(N^2) \end{aligned} \tag{2.6}$$

The various terms of the above equation are,

1. $\log_2 l$ is number of computations to broadcast the iris images
2. $\log_2 s$ is the number of computations to broadcast the Gaussian function
3. N^2 denotes the number of computations to perform convolution operation in each of the hypercube in parallel

Speedup is given as,

$$speedup = \frac{T_{serial}}{T_{parallel}} = \frac{O(lsN^2)}{O(N^2)}$$

2.4 Summary

In this chapter, parallelism has been introduced both within and across octaves using ls–processor SIMD hypercube. This was possible due to the independence of different scale levels within the octave and the independence of various octaves. This has enabled a speedup of ls between parallel and serial scale space construction where l denotes the number of octaves and s represents the number of Gaussian scale levels within an octave.

Chapter 3

Parallel Geometric Hashing based Indexing

Automatic identification of an individual is an area of keen interest among the researchers. Among several existing biometric traits, iris performs with enhanced accuracy and reduced time [5]. Visually recognizing an individual reliably through iris has become an intriguing approach when an image is captured at a very short distance. A generic iris biometric system extracts features from the input image and performs identification by comparing the probe template with all templates stored in the database. The number of false acceptances grows significantly due to the increase in the size of the database. Further, the time required to find the identity of an individual is directly proportional to the size of the database [22]. Thus, there is stringent requirement to minimize the time required to claim identification.

In this chapter, an effort has been made to provide an effective indexing approach that is unaffected to possible coordinate transformation and partial obstruction. Geometric hashing is a model based object recognition indexing technique in which the location of the keypoints are used [30]. In this technique, the position of the keypoints remains unchanged under similarity transformation. During retrieval, the location of the keypoints are computed for probe image and indexed into the hash table to find the possible matches. The primary reason behind the popularity of geometric hashing is its searching speed and recognition of object efficiently. Owing to above–mentioned

advantages, geometric hashing can be very efficiently used in the field of biometrics. Geometric hashing has already been applied to iris biometrics for indexing database during identification [28]. In this chapter, an effort has been made to introduce the concept of parallelism using shared SIMD hypercube [37].

The rest of the chapter is organized as follows. Section 3.1 discusses the traditional geometric hashing approach applied on the iris image to find the identity of an individual. The proposed parallel geometric hashing is discussed in detail in Section 3.2. The asymptotic analysis of the two approaches is compared in Section 3.3.

3.1 Serial Geometric Hashing based Indexing

The concept of geometric hashing have been proposed originally for indexing. The features are matched against the set of features available in the database. This technique performs well in case the probe image has undergone various possible transformations and occlusion compared to its database counterpart. Geometric hashing is a well known approach for model based object recognition and is even successful in finding its applicability to iris indexing [28]. In this approach, the keypoints are detected using Scale Invariant Feature Transform (SIFT) [20]. The detected keypoints are used for indexing using geometric hashing. The detailed description of traditional indexing approach is given below.

3.1.1 Geometric Hashing

Geometric hashing is an indexing technique for object recognition that uses location of keypoints which are invariant to similarity transformation as an index to the hash table [30, 31]. The extracted keypoints are used for indexing the database using the spatial location. The transformed locations are stored into 2–D hash table. The key advantage of geometric hashing over other indexing approaches is that this technique possesses invariance to various possible transformations and occlusion. Thus, if the probe iris image is transformed, the keypoints are indexed to the same entries of the hash table as shown in Figure 3.1. The steps involved are indexing and retrieval and

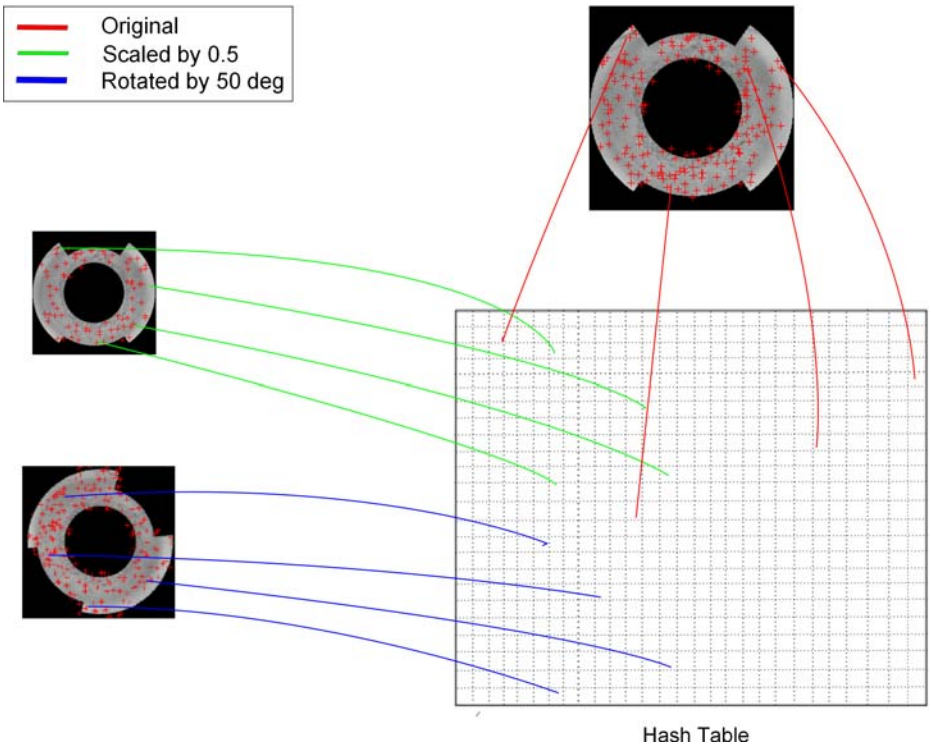


Figure 3.1: An instance showing the robustness of geometric hashing to rotation, scaling and occlusion.

are described below.

Indexing

The detected keypoints (k) on annular iris image are used for indexing the database as shown in Figure 3.2(a). Using orthonormal bases for coordinate system, two points (k_1 and k_2) are chosen as basis pair (Figure 3.2(b)) and remaining points are transformed using

$$P - P_0^i = uP_x^i + vP_y^i \tag{3.1}$$

where $P = [x\ y]$ is the keypoint to be indexed, (u, v) is the location of P after similarity transformation. P_x^i and P_y^i are defined by

$$P_x^i = \frac{k_2 - k_1}{2} \tag{3.2}$$

$$P_y^i = \mathrm{Rot}_{90}(P_x^i) \tag{3.3}$$

where Rot_{90} refers to rotation of coordinate locations by 90 degrees. The midpoint

P_0^i between k_1 and k_2 is defined by,

$$P_0^i = \frac{k_1 + k_2}{2} \tag{3.4}$$

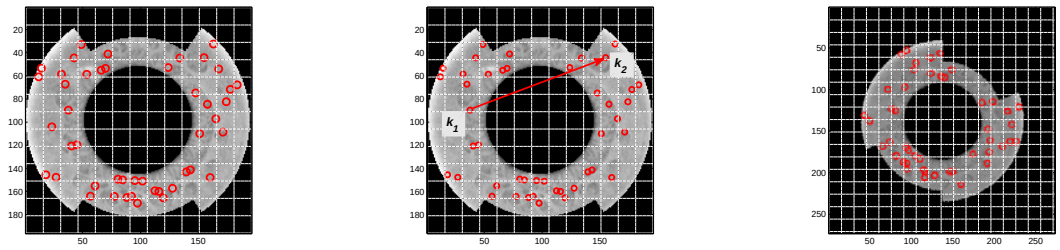


Figure 3.2: Geometric Hashing for Iris. (a) Detected keypoints on annular iris image, (b) Selection of basis pair (k_1 and k_2) (c) Transformed coordinates with respect to k_1 and k_2 as basis pair.

The transformed points with respect to basis pair (k_1 and k_2) is shown in Figure 3.2(c). For the formation of hash table, all possible ordered basis pairs of an iris image are selected to obtain transformation invariant coordinates (u, v) of the remaining points (x, y). The hash bin occupancy for the hash table is non–uniform and consists of peak that accumulates large number of entries. Wolfson and Rigoutsos have proposed an efficient technique for uniform distribution of entries in the hash table [30]. This approach has been used in [28] to have uniform distribution of entries over the hash table. The hash table at (u, v) contains entry of 3–tuple form (M, k_1, k_2) for each iris image M with basis pair $\overrightarrow{k_1 k_2}$.

Retrieval

During identification, iris images that have close proximity with the probe image are retrieved from the database. The probe image is preprocessed to detect annular portion of iris. The keypoints are localized on the annular probe iris image and arbitrarily two keypoints are chosen as ordered basis pair and transformed such that its midpoint coincides with the center of origin with direction in the positive x axis. The magnitude of basis vector has unit length. The coordinates of remaining keypoints are determined using equation (3.1) for a chosen basis pair. Each transformed entry is quantized and mapped to the hash table. For each entry found in the corresponding

hash table bin, a vote is cast. The basic assumption is that in case the probe image contains basis pair that corresponds to that of model image from database and then it is expected to receive votes from all other unoccluded points. The total number of votes for various basis pairs corresponding to each model image is determined. If the number of votes received for each model images are greater than a threshold (λ) then these images are considered to be potential matches for probe image. Further the keypoint descriptor for probe and candidate model images are compared to find top best matches.

3.2 Parallel Geometric Hashing using SIMD Hypercube

The geometric hashing discussed earlier is an efficient approach for performing iris identification in reduced time. However, this approach has a scope to further improve in terms of time by imbibing underlying parallelism. The work proposed in this chapter is an extension of geometric hashing based indexing proposed by Mehrotra et al. [28]. This chapter highlights the scope of parallelism at indexing as well as retrieval phase. The primary differences between the proposed work and traditional geometric hashing approach for iris has been outlined as follows. Firstly, during indexing phase, geometric invariants are computed and indexed into the hash table in parallel for each basis pair using SIMD hypercube. This significantly improves running time in contrast to indexing phase of traditional geometric hashing. Secondly, during iris retrieval the keypoints from probe image are used to compute geometric invariants to cast votes for the corresponding entries in the hash table in parallel. These two steps in parallel provide a significant improvement in computation time. Block diagram of the proposed model is shown in Figure 3.3.

3.2.1 Indexing Phase

The indexing approach discussed in Section 3.1.1 can be made parallel during the calculation of transformed locations (u, v) for each basis pair (k_i and k_j). In the pro-

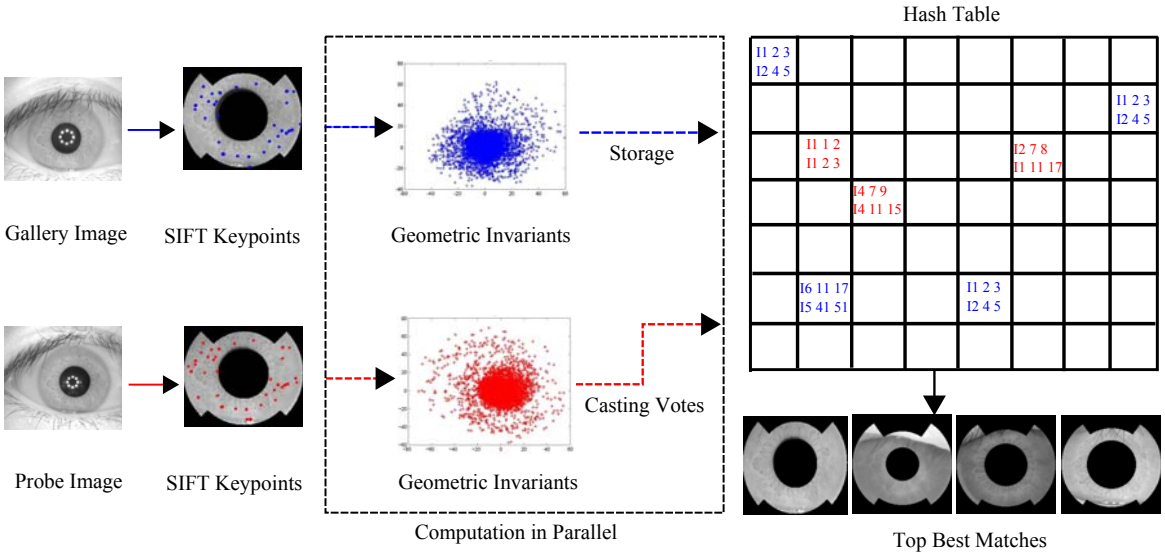


Figure 3.3: Proposed geometric hashing: Solid arrows indicate serial computation and dotted arrows represent parallel computation.

posed approach hash bin entry is computed in parallel using n–dimensional hypercube SIMD computer [37, 38]. The process starts with assigning a label to each keypoint $(0 \leq k \leq n-1)$ which are already computed and stored in a shared global memory as shown in Figure 3.4. The keypoints on the shared memory are mapped to local memory of each processor within the hypercube having same index using static mapping scheme [38]. Static mapping is used when the number of keypoints to be distributed among processors is known prior to indexing. For static mapping, data partitioning is generally done using arrays and graphs. In this approach, block distribution is used to assign uniform contiguous portions of an array (represented as shared global memory) to different processors. For instance, the k^{th} point on the shared global memory is mapped to k^{th} processor of the hypercube. The reason for storing the keypoints on shared memory is to enable concurrent access by all the processors.

Here all-to-all broadcast is performed by using recursive doubling to store the n keypoints in local memory of each processor so that it contains its own information and the information of other processors in the hypercube [38]. The local memory of individual processors is represented by square brackets as shown in Figure 3.5. After allotting n keypoints to n processors geometric hashing is used to perform indexing. In the serial geometric hashing approach, for each choice of basis pair (k_i and k_j)

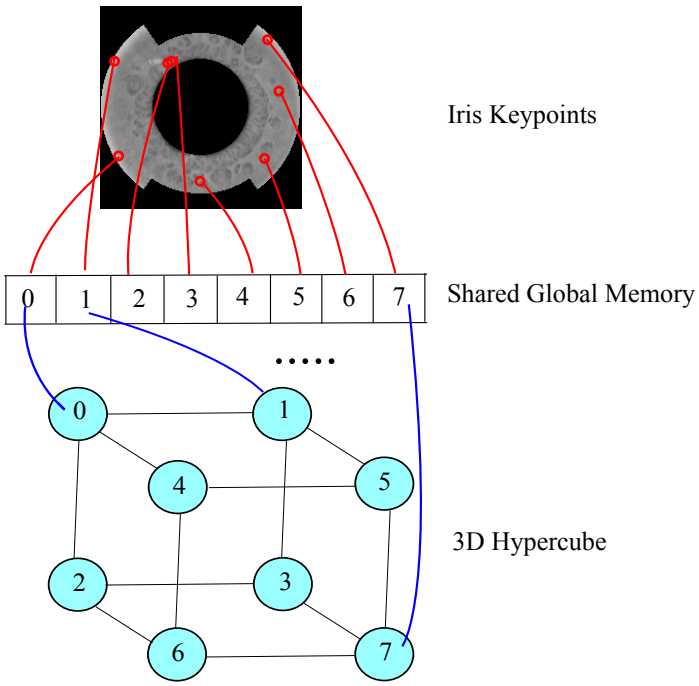


Figure 3.4: Mapping of detected keypoints to the shared global memory and 3D hypercube.

the remaining $(n-2)$ keypoints are transformed as discussed in Section 3.1.1. This computation requires ${}^{n}C_{2}$ operations to be performed serially, hence expensive. Thus, in the proposed parallel geometric hashing approach, each basis pair (k_i and k_j) is allotted to i^{th} processor with other $(n-2)$ keypoint locations. The reason for choosing all-to-all broadcast is to perform independent computation of geometric invariants (using equation 3.1) by individual processors. As it has already been mentioned by the authors in [30] that the basis pair should be chosen with positive x direction. Thus, each processor performs atmost $(n-i-1)$ computations to find the geometric invariants for allotted basis pairs. Table 3.1 shows the allocation of basis pair to each processor. Due to parallel execution and novel mapping strategy the time required to perform indexing reduces significantly. The invariants are indexed into 2-D hash table and each entry is represented by 3-tuple (M, k_i, k_j). Algorithm 3 outlines the steps involved in parallel indexing phase.

Algorithm 3: Parallel Indexing Phase

Result: Computation and storage of geometric invariants in hash table

1 **for** *Each of M iris images* **do**

2 n keypoints are extracted using SIFT

3 Map n keypoints to shared global memory

4 All-to-all broadcast is performed

5 Choose two keypoints as basis pair (k_i and k_j) and assign to i^{th} processor

6 Compute geometric invariants (u, v) for each basis pair using equation (3.1) in parallel

7 Store 3-tuple entry (M, k_i, k_j) at (u, v) in the hash table

3.2.2 Retrieval Phase

During retrieval phase, the keypoints (n) are extracted from probe iris image (q). The i^{th} keypoint is mapped to i^{th} processor of the hypercube using the same mapping scheme as discussed in Section 3.2.1. Two points are chosen as basis pair and geometric invariants are obtained for remaining $(n-2)$ keypoints in parallel as discussed in Algorithm 3. The invariants (u, v) are used to index into the hash table and for each entry found, a vote is cast as shown in Figure 3.6. As parallel computation during vote counting requires virtual processor set associated to each hash bin entry, this is quite costly for hash table with large number of bins. Further, concurrent updation of votes by independent hash bin entry (represented as a processor) may

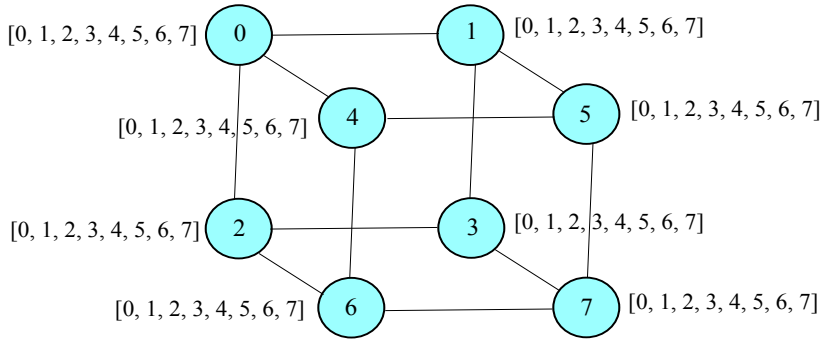


Figure 3.5: All-to-all broadcast is performed. [0,1,... n-1] represents the local memory of individual processors.

Table 3.1: An example showing allocation of 8 keypoints to 8 processors and the number of computations performed by individual processors.

Processor id	**Basis Pair**	**Remaining Points**	**No.of Computations**
0	0 1 0 2 0 3 0 4 0 5 0 6 0 7	2 3 4 5 6 7 1 3 4 5 6 7 1 2 4 5 6 7 1 2 3 5 6 7 1 2 3 4 6 7 1 2 3 4 5 7 1 2 3 4 5 6	7
1	1 2 1 3 1 4 1 5 1 6 1 7	0 3 4 5 6 7 0 2 4 5 6 7 0 2 3 5 6 7 0 2 3 4 6 7 0 2 3 4 5 7 0 2 3 4 5 6	6
2	2 3 2 4 2 5 2 6 2 7	0 1 4 5 6 7 0 1 3 5 6 7 0 1 3 4 6 7 0 1 3 4 5 7 0 1 3 4 5 6	5
3	3 4 3 5 3 6 3 7	0 1 2 5 6 7 0 1 2 4 6 7 0 1 2 4 5 7 0 1 2 4 5 6	4
4	4 5 4 6 4 7	0 1 2 3 6 7 0 1 2 3 5 7 0 1 2 3 5 6	3
5	5 6 5 7	0 1 2 3 6 7 0 1 2 3 4 6	2
6	6 7	0 1 2 3 4 5	1
7	–	–	0

create synchronization issues [39]. Thus, the top best matches are found serially using traditional geometric hashing approach. The votes are accumulated for each gallery image (M) in the hash bin indexed by probe image. These accumulated votes are compared against threshold value (λ) and corresponding gallery images are retained to get the candidate set. For each entry in the candidate set, the descriptor information is matched with that of probe image to find top best matches. The proposed iris retrieval algorithm is given in Algorithm 4.

3.3 Asymptotic Analysis

3.3.1 Serial Geometric Hashing

The cost of computing the location of n keypoints to index into the hash table is given by:

$$\begin{aligned} \sum_{i=1}^{n}(n-2)(n-i) &= (n-2)\{(n-1)+(n-2)+ \\ &\quad (n-3)+\ldots+1+0\} \end{aligned} \tag{3.5}$$

Applying arithmetic series, we get the serial complexity as,

$$\begin{aligned} T_{serial} &= (n-2)\left\{\frac{(n-1)n}{2}\right\} \\ &= \frac{n(n-1)(n-2)}{2} \\ &= O(n^3) \end{aligned} \tag{3.6}$$

$$\tag{3.7}$$

Algorithm 4: Parallel Retrieval Phase

Result: Top Matches for probe image

1 Interest points, n are extracted using SIFT
2 Repeat steps 3 to 6 from Algorithm 3
3 Cast a vote for each entry found at (u, v) in the hash table
4 Find potential images with votes greater than λ
5 Match keypoint descriptor of potential images with probe image to return top best match

As it is clearly evident from equation (3.5) that the term in the braces containing the addition indicates the sequential computation of invariants. The computations for each basis pair are independent of each other. Thus, in this chapter, the invariants for each basis pair are computed in parallel using hypercube.

3.3.2 Parallel Geometric Hashing

In the proposed scheme, the inherent parallelism is exploited at two different phases, both during indexing and retrieval phase:

For n keypoints the time required is,

$$\begin{aligned} T_{parallel} &= n + \log_2 n + (n-1)(n-2) \\ &= n + \log_2 n + (n^2 - 3n + 2) \\ &= O(n^2) \end{aligned} \tag{3.8}$$

The terms in $T_{parallel}$ can be explained as,

1. n computations for mapping of n keypoints to the shared global memory
2. $\log_2 n$ computations to perform all-to-all broadcast
3. $(n-1)(n-2)$ computations for parallel indexing of keypoints
   - (i) $(n-2)$ for finding geometric invariants by each basis pair
   - (ii) atmost $(n-1)$ computations are required by the processor to find $(n-2)$ invariants.

Retrieval Phase

The time required during retrieval is same as indexing phase (i.e., $O(n^2)$ as proved above) since the same steps are followed as mentioned in Algorithm 4.

The time complexity mentioned in the section is for single iris instance. This can be extended for M iris images in the database. Thus, the indexing as well as retrieval

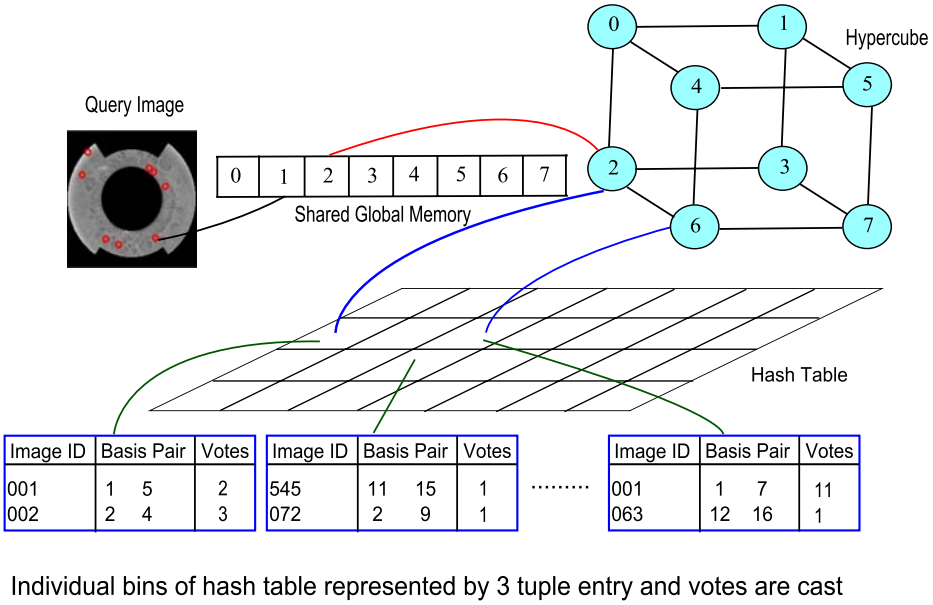


Figure 3.6: Parallel geometric hashing during iris retrieval phase.

time can be expressed as $O(Mn^2)$. This shows the improvement of proposed parallel geometric hashing algorithm over traditional algorithm [28]. The proposed approach performs with time complexity of $O(Mn^2)$ in comparison to traditional approach which performs with complexity of $O(Mn^3)$ for M iris instances.

Hence, the speedup is given as,

$$speedup = \frac{T_{serial}}{T_{parallel}} = \frac{O(Mn^3)}{O(Mn^2)}$$

3.4 Summary

This chapter proposes a parallel indexing scheme for iris biometrics. In the proposed approach, inherent parallelism has been introduced at two different phases i.e., indexing and retrieval. This scheme provides an improvement over traditional geometric hashing based indexing scheme in terms of time. It has been analytically proved that the parallel geometric hashing approach using hypercube has time complexity of $O(Mn^2)$ in comparison to traditional approach which has complexity of $O(Mn^3)$, where M is the database size. This marks the applicability of the proposed identification system in real time applications where both time and accuracy cannot be compromised.

Chapter 4

Rank Based Identification using Bitonic Sort

Sorting of data is an essential ingredient for many real life and computer science applications. Sorting is defined as the process of arranging a random collection of data into monotonically decreasing (or increasing) order. Manipulating data in sorted order is always easier than data in unsorted order, hence many algorithms are developed to serve this purpose. The sequential sorting algorithms have optimal time complexity of $O(M \log_2 M)$ to sort a random sequence of order M. Thus, when M is large, sequential sorting algorithms do not meet the real time requirements of different applications where speed is a criterion. To alleviate this limitation, parallel sorting algorithms play an important role and many such algorithms exist in literature [1]. These algorithms not only utilizes the computer hardware efficiently, but also drastically reduces the execution time. Usually parallel sorting algorithms are implemented in two models, namely special purpose architecture and general purpose architecture. Special purpose architecture includes sorting network where sorting algorithms like AKS [40], bitonic sort [41] are implemented. Bitonic sorting involves a lot of interprocessor communication. This sorting approach could find its applicability to vote counts during identification/verification of a probe iris in a gallery iris is generally encountered. As mentioned in the previous chapter, during the retrieval phase of geometric hashing based indexing, the geometric invariants of the probe iris image are mapped to the

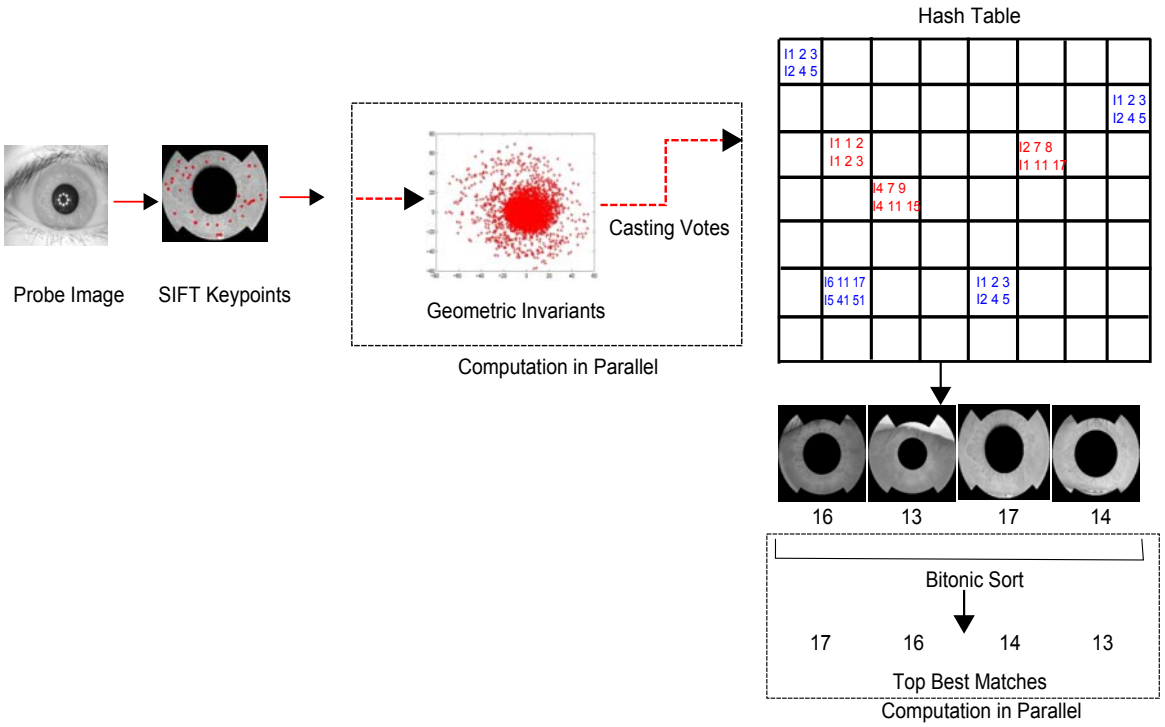


Figure 4.1: Block diagram finding ranks of individuals using bitonic sort

hash table. The corresponding mapped gallery iris receives votes and vote counts are incremented every time a mapping takes place. This generates a set of images which have non–zero vote count and are candidates for potential matches. But while finding rank based matching there is a need to find the top k best matches which need to be sorted in decreasing order of vote counts. Hence, sorting plays a vital role. In this chapter, bitonic sorting scheme using HMA to sort the bins that receives non–zero votes is used. The overall block diagram of the identification scheme along with parallel sorting is shown in Figure 4.1. In this chapter, a bitonic sorting algorithm using Hypercube Mesh Architecture (HMA) is discussed. As discussed earlier, interprocessor communication is a bottleneck for speedup gain in parallel processing. So, a bit–parity based strategy [41] is suggested to minimize this in HMA. This strategy is not applied in general purpose architecture (HMA).

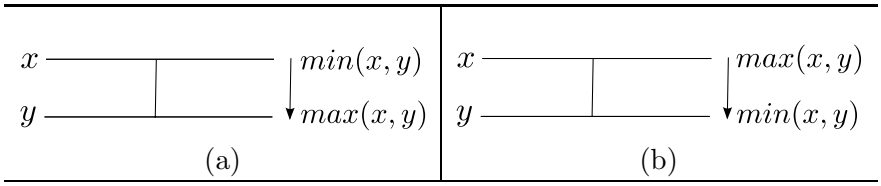


Figure 4.2: (a) Increasing Comparator (+) (b) Decreasing Comparator (−)

4.1 Related Works on Bitonic Sort

Bitonic sort is a mechanism of rearranging the bitonic sequence into a sorted sequence. A ***bitonic sequence*** is a sequence of numbers $\{a_0, a_1, \ldots, a_{m-1}\}$ with the following two properties [1]:

(i) There exists an index i, $0 \leq i \leq m-1$, such that $\{a_0, a_1, \ldots, a_i\}$ is monotonically decreasing and $\{a_{i+1}, \ldots, a_{m-1}\}$ is monotonically increasing and vice–versa.

(ii) If cyclic shift of any sequence exhibits the above property, then the sequence can also be termed as a bitonic sequence.

Example:

(a) $\{16, 15, 13, 12, 11, 14, 17, 18\}$ is a bitonic sequence because it first decreases and then increases.

(b) $\{13, 11, 12, 14, 16, 19, 18, 15\}$ is another bitonic sequence because it is a cyclic shift of $\{19, 18, 15, 13, 11, 12, 14, 16\}$.

A bitonic sequence can be rearranged to monotonically decreasing sequence through a bitonic merger [1, 41]. Let us consider an example,

Example: Let $s = \{a_0, a_1, \ldots, a_{m-1}\}$ be a bitonic sequence such that $a_0 \geq a_1 \geq \ldots \geq a_{m/2-1}$ and $a_{m/2} \leq a_{m/2+1} \leq \ldots \leq a_{m-1}$.

The sequence, s are divided into two subsequences to create two different bitonic sequences through the following criteria:

$$\begin{aligned} s_1 &= [max\{a_0, a_{m/2}\}, max\{a_1, a_{m/2+1}\}, \ldots, max\{a_{m/2-1}, a_{m-1}\}] \\ s_2 &= [min\{a_0, a_{m/2}\}, min\{a_1, a_{m/2+1}\}, \ldots, min\{a_{m/2-1}, a_{m-1}\}] \end{aligned} \quad (4.1)$$

This operation of splitting a bitonic sequence of size m into two bitonic subsequences is known as a bitonic split [1]. The above equation is used recursively to obtain shorter bitonic subsequences for each of the bitonic sequences until subsequences has size one.

Bitonic sorting can be performed on the following architectures:

(a) Special Purpose Architecture (SPA) (e.g. *Sorting Network*)

(b) General Purpose Architecture (GPA)

Two types of comparators namely, *Increasing comparator* ($+$) and *Decreasing comparator* ($-$) shown in Figure 4.2 are generally used to undergo the sorting using compare-exchange criteria in parallel. The usage of the comparators is explained in the further sections.

A bitonic sorting algorithm can be implemented in a sorting network [42]. A sorting network contains $m/2$ comparators to sort votes received by m gallery iris images with $0 \leq m \leq M$, where, M denotes the total vote counts which corresponds to the total number of gallery iris images that can receive votes (assumed to be power of 2). Here, $m \subseteq M$ because a proper subset of total number of gallery images may get votes during retrieval phase.

This algorithm involves several stages to produce sorted order of vote count for different gallery iris images. Each stage involves $m/2$ compare–exchange operations using $m/2$ comparators to sort m vote count. Each stage j has j number of steps with $m/2$ comparators in each step. The total number of stages involved to perform the sorting is $\log_2 m$. Figure 4.3 shows the stages in sorting network to sort eight vote counts in descending order. As it is evident from the figure, a total of six steps are there in the three stages of the sorting network. It may be observed that the bitonic sort algorithm is computation–intensive clearly obvious and involves a lot of inter–process communication in the sorting network. So, to reduce this, Batcher introduced parity strategy to alleviate the spurious communication between the processors in the sorting network [41].

An effort has been made to use both non–parity and parity based strategy in bitonic–sorting algorithm to sort the vote counts through a general–purpose parallel

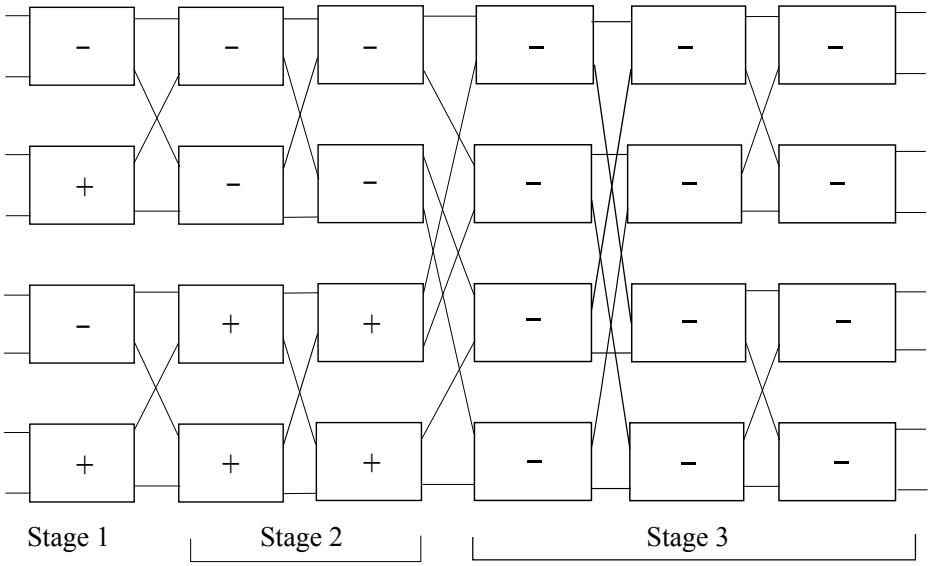


Figure 4.3: Sorting network for 8 vote counts for decreasing sequence

architecture. A general–purpose parallel architecture is chosen such that the traveling distance during the communication among the processors is minimum. So, the parallel architecture in which there is interconnection between the processors differing in exactly one bit exhibits the above property. SIMD array processor with static interconnection network like *hypercube* and *mesh connected* parallel computers are the best known to fall under this category [1]. Section 4.2 describes the proposed bitonic sorting scheme.

4.2 Proposed Bitonic Sorting using Hypercube Mesh Architecture

A bitonic sorting algorithm can also be performed in general–purpose parallel architecture. In the proposed scheme, a variant of hypercube i.e. a mesh connected parallel architecture with each processor connected to every other processor with their labels differing by exactly one bit. The compare–exchange operations take place between the processors whose binary represented labels differ in exactly one bit. The architecture is termed as *hypercube mesh architecture*(HMA). A hypercube mesh architecture with m (power of 2) processors are chosen to sort m vote count [1, 43]. The $m/2$ pairs

of processors involved during the inter–processor communication at each stage represent the $m/2$ comparators of the sorting network discussed in Section 4.1. Bitonic sort uses d–dimensional hypercube in d^{th} stage , where $d = \log_2 m$. In the variant architecture, the hypercube architecture is converted into a hypercube mesh architecture as discussed below.

The j^{th} stage of bitonic sort uses the d^{th} dimension of hypercube, where $d = d_1 + d_2$ and d is the dimension of the hypercube. The $2^{d_1+d_2}$–node hypercube has been embedded into $2 \times 2^{d_1+d_2-1}$ mesh connected topology. There are $m/2^j$ numbers of $(2 \times 2^{j-1})$ HMA. Figure 4.4 demonstrates the embedding of 1-D, 2-D and 3-D hypercube to 2×1, 2×2, 2×4 HMA, respectively. Similar conversions of hypercube to hypercube mesh architecture can be made for higher dimensions as well.

The bitonic sorting in HMA can be accomplished in two ways:

(i) Non–parity strategy based bitonic sort

(ii) Parity strategy based bitonic sort

4.2.1 Non–Parity Strategy based Bitonic Sort

Each processor is labeled with a number from 0 to $m - 1$. The binary representation of each label contains $\log_2 m$ bits. In HMA the labels of neighboring nodes differ in exactly one bit.

The original sequence of vote counts are converted into a bitonic sequence prior to the final stage of bitonic sort. During the first step of this stage, the processors which differ only in $(\log_2 m)^{th}$ bit (*i.e.*, the most significant bit) of the binary representation of their labels compare–exchange their vote counts. Similarly, during the second step of the algorithm, the compare–exchange operation occurs among the processors whose binary representation of labels differ in $(\log_2 m - 1)^{th}$ bit.

So, in general, during the j^{th} step of the final stage, inter–process communication prevails between the processors whose binary represented labels differ in $(\log_2 m - (j-1))^{th}$ bit (the least significant bit).

For better understanding of the problem, an example has been taken to find the top rank among 8 gallery iris images which has received votes during retrieval. The votes

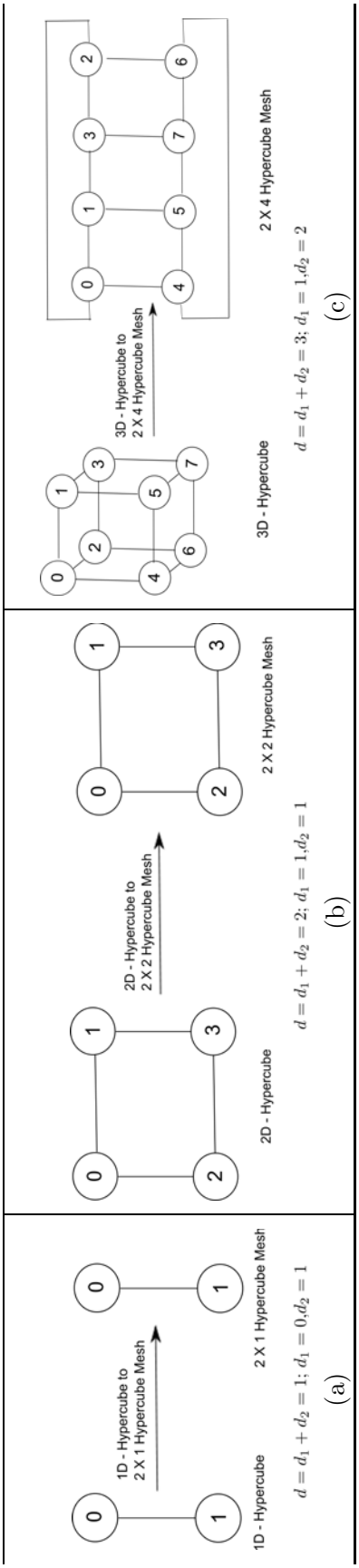


Figure 4.4: (a) Embedding 1D Hypercube to 2×1 Hypercube Mesh (b) Embedding 2D Hypercube to 2×2 Hypercube Mesh (c) Embedding 3D Hypercube to 2×4 Hypercube Mesh

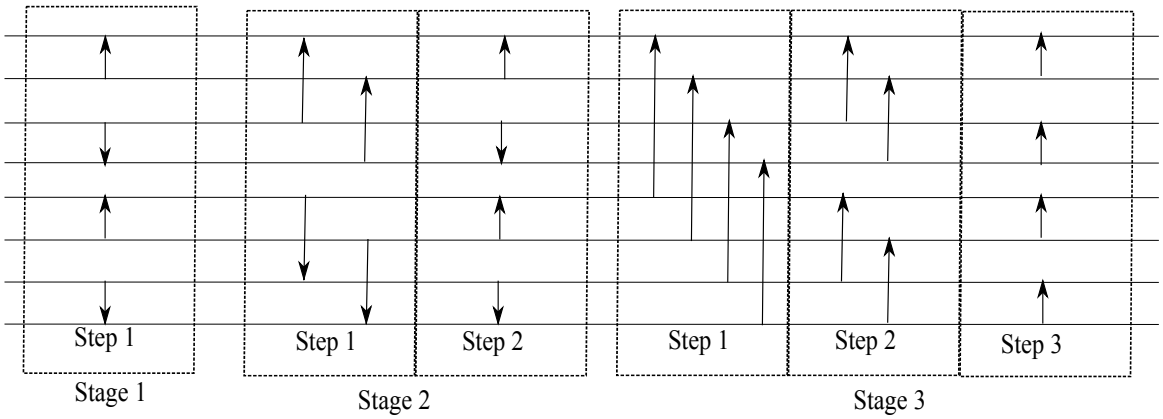


Figure 4.5: Knuth Diagram for eight vote counts (decreasing sequence)

obtained by different gallery iris images are given as $\{16, 11, 10, 15, 17, 12, 13, 14\}$. Since there are 8 gallery images which has received votes, so sorting will be performed in 3 *i.e.*, $\log_2 8$ stages. In this strategy, the compare–exchange takes place between processors. The compare–exchange operation and the type of comparator used during the communication between the processors is decided through the Knuth diagram [41]. A Knuth diagram shows how to program the algorithm in a parallel architecture (Figure 4.5). In Knuth diagram, each horizontal line represents the number of votes received by a gallery iris images stored in each processor and each vertical arrow represents a pair of processors reading two votes from each processor, comparing them, and writing them back to the corresponding processor in proper order. The processors (P_a and P_b) which are involved in the communication is depicted in Table 4.1. The type of comparators which is to be incorporated between pair of processors are decided from $FLAG_1$ as shown in the table. If $FLAG_1$ is TRUE, *increasing comparator* (I) is used otherwise *decreasing comparator* (D) is used.

The states of different processors during each step of a particular stage is shown in Figure 4.6. Each stage j has $m/2^j$ numbers of $2 \times 2^{j-1}$ HMA. Figure 4.6 also shows the HMA to be used at each stage. The right side of the figure shows the comparators which are incorporated within the processor. The left arrows and right arrows represent the decreasing and increasing comparator, respectively. The type of comparators which are to be incorporated within pairs of processors are decided through Knuth diagram and can also be found in Table 4.1. The type of compara-

tors decides the correct placement of the vote count. The second stage produces the bitonic sequence. The bitonic sequence is given as input to the final stage which gives the necessary sorted sequence. Algorithm 5 shows non–parity based bitonic sort.

Algorithm 5: Bitonic sort with Hypercube Mesh (Non–Parity Based)

Input : A : Array of M vote counts

Output: A :Array of sorted vote counts

For all comparators in parallel

```
1  FLAG_1 ← FALSE
2  for j = 1 to log_2 M do
3      if ⌊2 * comp/2^j⌋ mod 2 ≠ 0 then
4          FLAG_1 ← TRUE
5      dim ← 2^(j−1)
6      while dim ≥ 1 do
7          Q ← no % dim
8          R ← no + Q
9          addr1 ← 2 * R + Q
10         addr2 ← addr1 + dim
11         Read two vote counts (K_1 and K_2) from addr1 and addr2
12         FLAG = FLAG_1
13         if K_1 < K_2 then
14             FLAG ← !FLAG
15         if FLAG = TRUE then
16             Swap K_1 and K_2
17         dim ← dim/2
18         Write two vote counts (K_1 and K_2) back to respective processor
19 Store all the vote counts back to array A
20 Exit
```

However, the main demerit of non–parity based bitonic sorting algorithm is that

Table 4.1: Inter–Processor Communication in Non–Parity Based Strategy

(a) Stage - 1, Step - 1				
Comparator	P_a	P_b	$FLAG_1$	Comparator–type
0	0	1	F	D
1	2	3	T	I
2	4	5	F	D
3	6	7	T	I
(b) Stage - 2, Step - 1				
Comparator	P_a	P_b	$FLAG_1$	Comparator–type
0	0	2	F	D
1	1	3	F	D
2	4	6	T	I
3	5	7	T	I
(c) Stage - 2, Step - 2				
Comparator	P_a	P_b	$FLAG_1$	Comparator–type
0	0	1	F	D
1	2	3	F	D
2	4	5	T	I
3	6	7	T	I
(d) Stage - 3, Step - 1				
Comparator	P_a	P_b	$FLAG_1$	Comparator–type
0	0	4	F	D
1	3	7	F	D
2	1	5	F	D
3	2	6	F	D
(e) Stage - 3, Step - 2				
Comparator	P_a	P_b	$FLAG_1$	Comparator–type
0	0	2	F	D
1	1	3	F	D
2	4	6	F	D
3	5	7	F	D
(f) Stage - 3, Step - 3				
Comparator	P_a	P_b	$FLAG_1$	Comparator–type
0	0	1	F	D
1	2	3	F	D
2	4	5	F	D
3	6	7	F	D

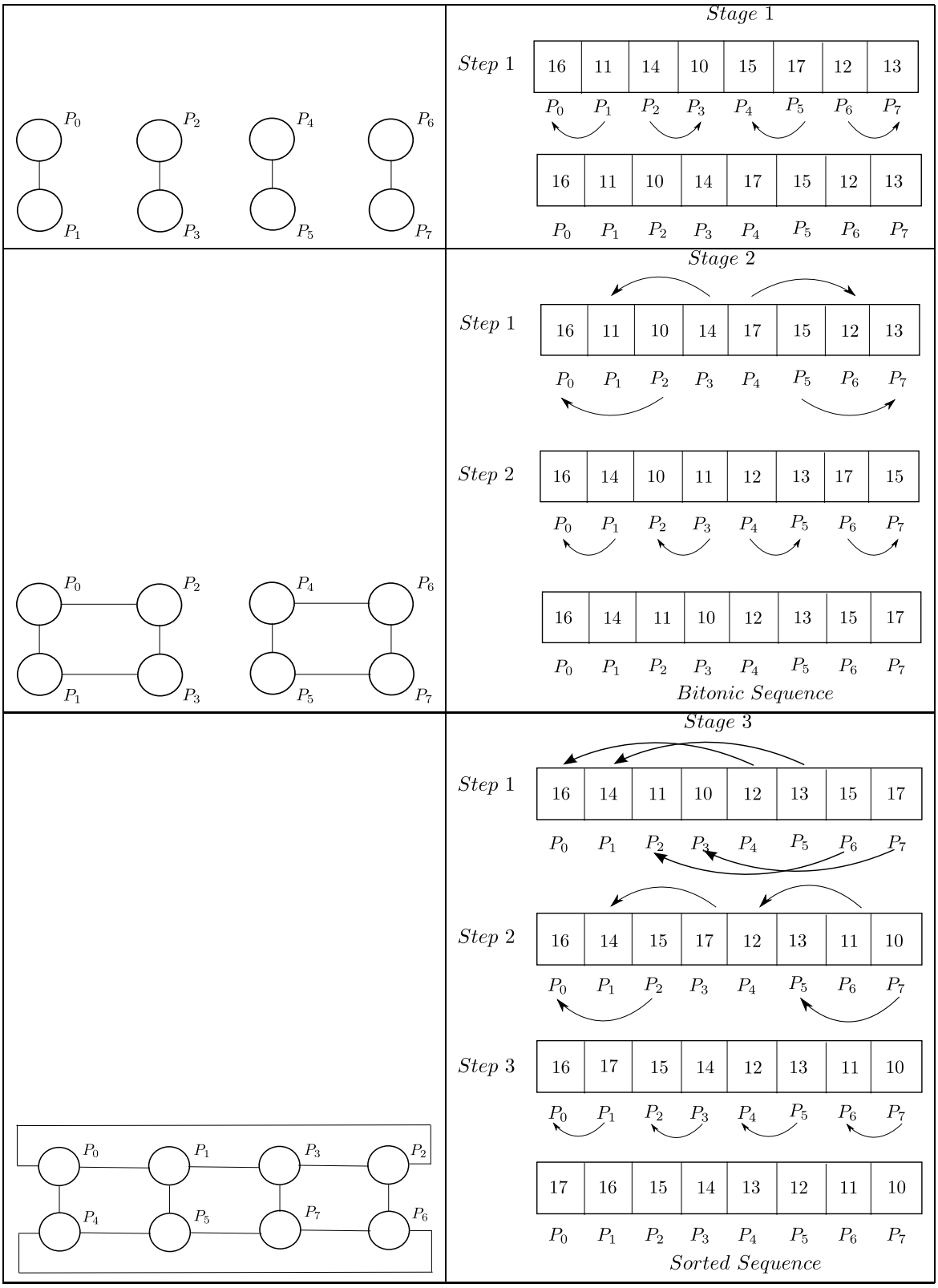


Figure 4.6: Non–Parity Based Bitonic Sorting

the bandwidth of the interconnection network must be substantial to guarantee good performance. During the bitonic sorting, each processor reads two vote counts, compares them, and writes two vote counts back to the respective processor through the interconnection network. Access through the interconnection network can be slow, since a read and write request may have to pass through multiple stages in the interconnection network. Hence, there is substantial degradation in performance due to communication through heavy network during bitonic sorting on a realistic machine. The main idea employed in the next section is that if each processor has enough local memory to store one vote count in each processor then each processor reads only one vote count from shared global memory, compares it to the vote count in its own local memory and writes only one vote count back to shared memory whereas the other vote count stays in the local memory. This reduces the number of shared memory accesses to half, therefore, leads to the performance improvement. This strategy is called as *parity strategy* [41]. Bitonic sort using the parity strategy in HMA is briefly discussed in Section 4.2.2.

4.2.2 Parity Strategy based Bitonic Sort

The parity is defined by the number of 1–bits in binary representation of the labels of the processors; if the labels has an even number of 1–bits then the processor has even–parity (i.e., 10111); if the label has an odd number of 1–bits then the processor has odd parity(e.g., 10011). In bitonic sort, each processor with even–parity label always pairs with an odd–parity label for comparison. The communication can be decreased by allowing each processor to retain the even–parity vote count in its shared global memory and just read and write the odd-parity vote count from and to local memory.

The vote counts are stored in the processor. The vote counts which are stored in the even–parity index are copied to the corresponding location of the shared global memory. The communication takes place between the odd–parity indexed processor and even–parity location of shared global memory. Since the shared global memory is available to all the processors, the odd parity indexed processor reads the votes from even parity index of shared global memory, checks the type of comparator and writes

back a relevant value to the shared global memory storing the other value in its own local memory.

Two flags ($FLAG_1$ and $FLAG_2$) are used to determine that the local memory of which processor retains maximum or minimum vote count. $FLAG_1$ decides the type of comparator as described in Section 4.1 that has to be incorporated within HMA. If $FLAG_1$ is TRUE, *increasing comparator* is used or otherwise. $FLAG_2$ depicts whether the index of local memory (lmi) or global memory (gmi) is greater. If $FLAG_2$ = TRUE, the local memory index is greater than global memory index. If $FLAG_1 = FLAG_2$ the local memory is minimum or otherwise. Table 4.2 shows the values of the $FLAG_1$ and $FLAG_2$.

The different phases of parity based bitonic sort using shared global memory and HMA is shown in Figure 4.7. The communication takes place between the shared global memory and local memory of odd–parity index of the processor of the HMA. The indices which makes communication is decided from the connectivity of the processor in HMA. The higher value is always stored in the direction of the arrow. The bitonic sequence is found in second stage.

Algorithm 6: Bitonic sort with Hypercube Mesh (Parity Based)

Input : A : Array of M vote counts

lmi : local memory index (Even Parity)

gmi : global memory index (Odd Parity)

$XOR : \bigoplus$

Output: A :Array of sorted vote counts

For all comparators in parallel

1 Load $M/2$ vote counts into lmi of even parity processors
2 Load $M/2$ vote counts into shared gmi accessible to all the processors
3 $FLAG_1 \leftarrow$ FALSE
4 **for** $j = 1$ *to* $\log_2 M$ **do**
5 **if** $\lfloor 2 * comp/2^j \rfloor \ mod\ 2 \neq 0$ **then**
6 $FLAG_1 \leftarrow$ TRUE
7 $dim \leftarrow 2^{j-1}$
8 **while** $dim \geq 1$ **do**
9 Obtain the index of the even–parity vote count: $gmi \leftarrow lmi \bigoplus dim$
10 READ a vote count from even-parity index from shared global memory
11 **if** $lmi > gmi$ **then**
12 $FLAG_2 \leftarrow$ TRUE
13 **else**
14 $FLAG_2 \leftarrow$ FALSE
15 **if** $FLAG_1 = FLAG_2$ **then**
16 LOAD lowest vote count into lmi
17 **else**
18 LOAD highest vote count into lmi
19 LOAD highest vote count into lmi
20 $dim \leftarrow dim/2$
21 Store all the vote counts back to array A
22 Exit

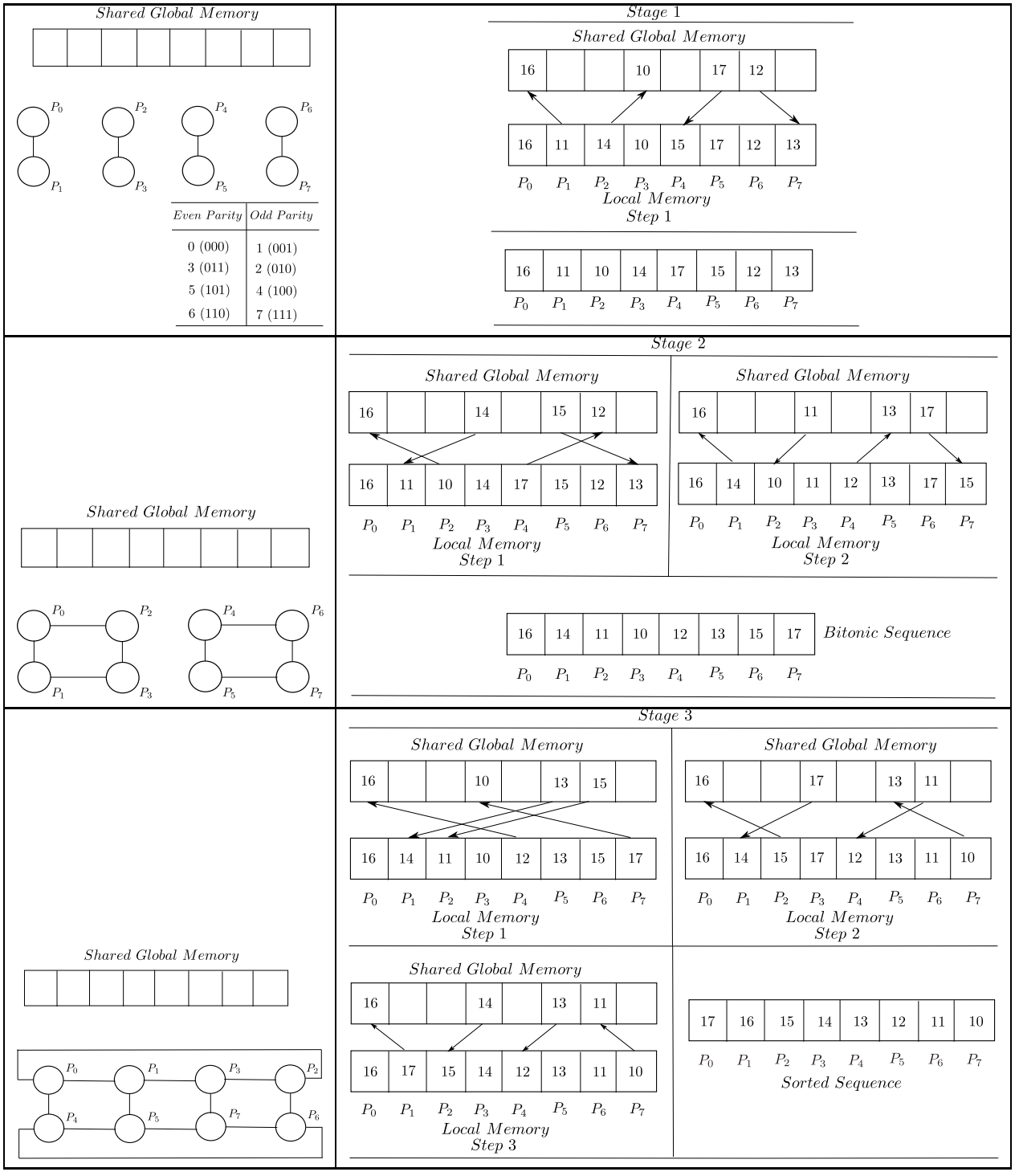


Figure 4.7: Parity Based Bitonic Sorting

Table 4.2: Local–Global Memory Communication in Parity Based Strategy

(a) Stage - 1, Step - 1							
Comparator	LGC		$FLAG_1$	$FLAG_2$	vote count		Direction
	lmi	*gmi*			*lmi*	*gmi*	
0	0	1	F	F	*low*	*high*	*gmi*
1	3	2	T	T	*low*	*high*	*gmi*
2	5	4	F	T	*high*	*low*	*lmi*
3	6	7	T	F	*high*	*low*	*lmi*
(b) Stage - 2, Step - 1							
Comparator	LGC		$FLAG_1$	$FLAG_2$	vote count		Direction
	lmi	*gmi*			*lmi*	*gmi*	
0	0	2	F	F	*low*	*high*	*gmi*
1	3	1	F	T	*high*	*low*	*lmi*
2	5	7	T	T	*low*	*high*	*gmi*
3	6	4	T	F	*high*	*low*	*lmi*
(c) Stage - 2, Step - 2							
Comparator	LGC		$FLAG_1$	$FLAG_2$	vote count		Direction
	lmi	*gmi*			*lmi*	*gmi*	
0	0	1	F	F	*low*	*high*	*gmi*
1	3	2	F	T	*high*	*low*	*lmi*
2	5	4	T	T	*low*	*high*	*gmi*
3	6	7	T	F	*high*	*low*	*lmi*
(d) Stage - 3, Step - 1							
Comparator	LGC		$FLAG_1$	$FLAG_2$	vote count		Direction
	lmi	*gmi*			*lmi*	*gmi*	
0	0	4	F	F	*low*	*high*	*gmi*
1	3	7	F	T	*high*	*low*	*lmi*
2	5	1	F	T	*high*	*low*	*lmi*
3	6	2	F	F	*low*	*high*	*gmi*
(e) Stage - 3, Step - 2							
Comparator	LGC		$FLAG_1$	$FLAG_2$	vote count		Direction
	lmi	*gmi*			*lmi*	*gmi*	
0	0	2	F	F	*low*	*high*	*gmi*
1	3	1	F	T	*high*	*low*	*lmi*
2	5	7	F	T	*high*	*low*	*lmi*
3	6	4	F	F	*low*	*high*	*gmi*
(f) Stage - 3, Step - 3							
Comparator	LGC		$FLAG_1$	$FLAG_2$	vote count		Direction
	lmi	*gmi*			*lmi*	*gmi*	
0	0	1	F	F	*high*	*low*	*lmi*
1	3	2	F	T	*low*	*high*	*gmi*
2	5	4	F	T	*low*	*high*	*gmi*
3	6	7	F	F	*high*	*low*	*lmi*

4.3 Asymptotic Analysis

The number of votes that can be cast to a gallery iris image during retrieval phase is in the range of $[0, {}^nC_2(n-2)]$, where, n denotes the number of keypoints present in the probe iris image. The number of gallery iris images which can receive votes are within a range $0 \leq m \leq M$. So, the time complexity to sort the vote count of M gallery images in two different strategies is given below.

Stage	No. of Steps
1	1
2	2
.	.
.	.
.	.
$\log_2 M$	$\log_2 M$

Table 4.3: Steps involved during Bitonic Sort

4.3.1 Non–Parity Strategy Based Bitonic Sort

Bitonic sort using non–parity based strategy in HMA involves $\log_2 M$ stages and each stage j again contains j steps. M number of processors in the hypercube which can be converted to mesh connected architecture is assumed.

Table 4.3 depicts the number of steps involved in each stage to sort M vote counts. Each step of a particular stage contains two read accesses from the processors and two write backs to the respective processor. Thus, there are four inter–processor references. So, the total number of references for each processor during bitonic sort is given below.

$$\begin{aligned} T(M) &= 4 \times \sum_{i=1}^{\log_2 M} i = \frac{4 \times \log_2 M \times (\log_2 M + 1)}{2} \\ &= 2 \times \log_2 M \times (\log_2 M + 1) \\ &= 2 \times (\log_2^2 M + \log_2 M) \end{aligned} \tag{4.2}$$

4.3.2 Parity Strategy Based Bitonic Sort

Bitonic sort using parity based strategy in HMA involves $\log_2 M$ stages and each stage j in turn contains j steps. Table 4.3 depicts the number of steps involved in each stage to sort M vote counts. Each step of a stage contains one read access to the even parity vote count from shared global memory which is accessible to all the processor. This vote count from the even parity index is compared with the vote count of the local memory of odd parity index processor writes only one vote count back to shared memory whereas the other vote count stays in the local memory. Hence, each step of a stage has two shared memory references. So, the total number of shared memory references to sort M vote counts is given by below.

$$\begin{aligned} T(M) &= 2 \times \sum_{i=1}^{\log_2 M} i = \frac{2 \times \log_2 M \times (\log_2 M + 1)}{2} \\ &= \log_2 M \times (\log_2 M + 1) \\ &= \log_2^2 \ M + \log_2 M \end{aligned} \tag{4.3}$$

Hence, the total number of shared memory references per processor to sort M vote counts using parity strategy is $\log_2^2 M + \log_2 M$, which is half of that of that of non–parity based strategy. Hence, a parity based strategy for bitonic sort is preferable.

4.4 Summary

Bitonic sort has been implemented in hypercube mesh architecture using non–parity and parity based strategy to find the top k matches in the retrieval phase. The time complexity to sort the vote counts is $O(\log_2^2 M)$ using non–parity based bitonic sort. Since, non–parity based bitonic sorting has to undergo a lot of interprocessor communication. So, parity based bitonic sorting is used to minimize the number of references among processors. The bitonic sorting algorithm used to find the rank of gallery iris images outperforms other sequential comparison–based sorting algorithms in terms of time complexity.

Chapter 5

Conclusions and Future Work

This thesis proposes the introduction of parallelism in feature extraction and identification stages of iris biometrics. The first contribution has been made in feature extraction stage. Feature extraction using SIFT constitutes various phases. In the first stage, the construction of scale space is done to find the smoothed images. The scale space contains various octaves. Parallelism has been introduced during this stage (within octave and across octaves) using ls SIMD hypercube. This introduction of parallelism has drastically improved the time complexity. The speedup gain due to the introduction of parallelism at this stage is found to be ls.

The second and most valuable contribution is done during the identification stage of iris. Geometric hashing is a model based object recognition technique used to index the geometric invariants into the hash table. Geometric hashing contains two phases, indexing and retrieval. In this scheme, the geometric invariants for ${}^nC_2$ basis pairs are to be found. These invariants have to be found for both gallery images and probe image. These geometric invariants can be found in parallel for each basis pair using the SIMD hypercube. The computation of these geometric invariants is to be done in parallel for both phases. This introduction of parallelism marks an improvement in time complexity with the speedup gain given as n.

The third contribution is again done during the retrieval phase of identification stage. The vote counts obtained by different gallery images during the retrieval phase are always found in random order. So, to determine the top k matches amongst the

gallery iris images which has received votes is tedious. Bitonic sorting algorithm is used as parallel sorting algorithm to sort the vote counts of different gallery iris images. Bitonic sorting algorithm always involves a lot of interprocessor communication during the sorting. Hence, to minimize the number of communication among the processors parity based bitonic sorting algorithm is used to find the rank. The selection of parallel sorting algorithm (bitonic sorting algorithm) gives an improved time complexity when compared to the other sequential sorting algorithms.

To conclude this thesis deals with proposition of three different parallel algorithms in an iris biometrics system. In the beginning, the stages which possess inherent parallelism are identified and a SIMD hypercube or its variant are considered as the underlying architecture. Further, SIFT features are extracted from annular iris for matching using geometric hashing scheme. The algorithms suggested are namely given as,

1. Parallel Scale Space construction for keypoint detection
2. Parallel Geometric Hashing for enrollment and retrieval
3. Parallel bitonic sorting for rank–based identification

The asymptotic time complexity for each problem has been devised for both serial algorithm and its parallel counterpart. It has been observed that there is phenomenal improvement in time complexity. However, the algorithms suffer from few limitations due to different assumptions as described below:

1. The number of keypoints for each iris is considered to be fixed.
2. The hypercube consists of 2^x numbers of processors.
3. The product of Gaussian scale levels (s) and octaves (l) is considered to power of 2.

These assumptions may not be satisfied in reality. To deal with those situations it can be further investigated for real time implementations of proposed algorithms. In addition, investigations can be made to implement these algorithms in other parallel architectures.

Dissemination of Work

1. **Anukul Chandra Panda**, Hunny Mehrotra, and Banshidhar Majhi. Parallel Geometric Hashing for Robust Iris Indexing. Accepted for Publication in *Springer Journal of Real-Time Image Processing.*

2. **Anukul Chandra Panda**, Pankaj K. Sa, and Banshidhar Majhi. Bitonic Sort in Shared SIMD Array Processor. In *Proceedings of ACM International Conference on Communication, Computing; Security*, (ICCCS '11), Feb. 12–14, 2011, NIT Rourkela, Odisha, India, pages 273–276.

Bibliography

[1] Ananth Grama, Anshul Gupta, George Karypis, and Vipin. *Introduction to Parallel Computing.* Pearson Education, second edition, 2007.

[2] Berna L. Massingill Timothy G. Mattson, Beverly A. Sanders. *Patterns for Parallel Programming.* Addison-Wesley Professional, first edition, September 25, 2004.

[3] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability,Programmability.* McGraw-Hill, 1993.

[4] A. K. Jain, P. Flynn, and A. A. Ross. *Handbook of Biometrics.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[5] J. Daugman. How iris recognition works. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1):21 – 30, 2004.

[6] J. Daugman. The importance of being random: statistical principles of iris recognition. *Pattern Recognition*, 36(2):279 – 291, 2003.

[7] L. Flom and A. Safir. Iris recognition system. U.S. Patent 4,641,349, 1987.

[8] J. Daugman. Biometric personal identification system based on iris analysis. U.S. Patent No. 5,291,560, 1994.

[9] A.L. Yuille, D.S. Cohen, and P.W. Hallinan. Feature extraction from faces using deformable templates. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 104–109, 1989.

[10] R.P. Wildes. Iris recognition: an emerging biometric technology. *Proceedings of the IEEE*, 85(9):1348–1363, 1997.

[11] K.W. Bowyer, K. Hollingsworth, and P. J. Flynn. Image understanding for iris biometrics: A survey. *Computer Vision and Image Understanding*, 110(2):281–307, 2008.

[12] Y. Huang, S. Luo, and E. Chen. An efficient iris recognition system. In *International Conference on Machine Learning and Cybernetics*, volume 1, pages 450–454, 2002.

[13] Y. Liu, S. Yuan, X. Zhu, and Q. Cui. A practical iris acquisition system and a fast edges locating algorithm in iris recognition. In *20th IEEE Conference on Instrumentation and Measurement Technology*, volume 1, pages 166–168, 2003.

[14] H. Sung, J. Lim, J. Park, and Y. Lee. Iris recognition using collarette boundary localization. In *17th International Conference on Pattern Recognition*, volume 4, pages 857–860, 2004.

[15] Q. Tian, Q. Pan, Y. Cheng, and Q. Gao. Fast algorithm and application of hough transform in iris segmentation. In *International Conference on Machine Learning and Cybernetics*, volume 7, pages 3977–3980, 2004.

[16] G. Xu, Z.F. Zhang, and Y.D. Ma. Automatic iris segmentation based on local areas. In *International Conference on Pattern Recognition*, volume 4, pages 505–508. IEEE Computer Society, 2006.

[17] L.V. Birgale and M. Kokare. Iris recognition using discrete wavelet transform. In *International Conference on Digital Image Processing*, pages 147–151, 2009.

[18] L. Masek and P. Kovesi. Matlab Source Code for a Biometric Identification System based on Iris Patterns. The School of Computer Science and Software Engineering, The University of Western Australia., 2003.

[19] C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[20] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[21] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *9th European Conference on Computer Vision*, Graz Austria, 2006.

[22] A. Mhatre, S. Chikkerur, and V. Govindaraju. Indexing biometric databases using pyramid technique. In *International Conference on Audio and Video Based Biometric Person Authentication (AVBPA)*, pages 841–849, 2005.

[23] P. Gupta, A. Sana, H. Mehrotra, and C. Jinshong Hwang. An efficient indexing scheme for binary feature based biometric database. volume 6539, page 653909. SPIE, 2007.

[24] U. Jayaraman, S. Prakash, Devdatt, and P. Gupta. An indexing technique for biometric database. In *International Conference on Wavelet Analysis and Pattern Recognition*, volume 2, pages 758–763, 2008.

[25] U. Jayaraman, S. Prakash, and P. Gupta. Indexing multimodal biometric databases using kd-tree with feature level fusion. In *4th International Conference on Information Systems Security*, pages 221–234, Berlin, Heidelberg, 2008. Springer-Verlag.

[26] R. Mukherjee and A. Ross. Indexing iris images. In *19th International Conference on Pattern Recognition*, pages 1–4, 2008.

[27] N. B. Pusan and N. Sudha. A novel iris database indexing method using the iris color. In *3rd IEEE Conference on Industrial Electronics and Applications*, pages 1886–1891, 2008.

[28] Hunny Mehrotra, Banshidhar Majhi, and Phalguni Gupta. Robust iris indexing scheme using geometric hashing of sift keypoints. *J. Network and Computer Applications*, 33(3):300–313, 2010.

[29] Hunny Mehrotra, Badrinath G. S., Banshidhar Majhi, and Phalguni Gupta. An efficient iris recognition using local feature descriptor. In *IEEE International Conference on Image Processing (ICIP)*, pages 1957–1960, Egypt, November 2009.

[30] H.J. Wolfson and I. Rigoutsos. Geometric Hashing: An Overview. *IEEE Computational Science & Engineering*, 4(4):10–21, 1997.

[31] Y. Lamdan, J. T. Schwartz, and H. J. Wolfson. Object recognition by affine invariant matching. In *Computer Vision and Pattern Recognition (CVPR)*, pages 335–344, 1988.

[32] Yan Ke and Rahul Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *CVPR (2)*, pages 506–513, 2004.

[33] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346 – 359, 2008.

[34] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615 –1630, oct. 2005.

[35] Cong Geng and Xudong Jiang. Face recognition using sift features. In *ICIP*, pages 3313–3316. IEEE, 2009.

[36] G.S. Badrinath and P. Gupta. Palmprint verification using sift features. In *Image Processing Theory, Tools and Applications, 2008. IPTA 2008. First Workshops on*, pages 1 –8, nov. 2008.

[37] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability,Programmability.* McGraw-Hill, 1993.

[38] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing.* Pearson Education. Addison-Wesley, second edition, 2007.

[39] Isidore Rigoutsos and Robert A. Hummel. Massively parallel model matching: Geometric hashing on the connection machine. *IEEE Computer*, 25(2):33–42, 1992.

[40] M. Ajtai, J. Komlós, and E. Szemerédi. An 0(n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 1–9, New York, NY, USA, 1983. ACM.

[41] Jae-Dong Lee and Kenneth E. Batcher. Minimizing communication in the bitonic sort. *IEEE Trans. Parallel Distrib. Syst.*, 11(5):459–474, 2000.

[42] K. E. Batcher. Sorting networks and their applications. In *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314, New York, NY, USA, 1968. ACM.

[43] S.J. Johnsson. Combining parallel and sequential sorting on a boolean n-cube. Proceedings of International Conference on Parallel Processing, 1984.