GPU Accelerated Parallel Iris Segmentation

Kritika Kurani

Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela - 769008, India

GPU Accelerated Parallel Iris Segmentation

Dissertation submitted in

May 2014

to the department of

Computer Science and Engineering

of

National Institute of Technology, Rourkela

in partial fulfillment of the requirements

for the degree of

Bachelor of Technology

by

Kritika Kurani

(Roll 110CS0128)

under the supervision of

Prof. Banshidhar Majhi

Department of Computer Science and Engineering

National Institute of Technology, Rourkela

Rourkela - 769008, India

Dedicated to
Papa, Mummy
and
brother Adi

Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Odisha, India. www.nitrkl.ac.in

Dr. Banshidhar Majhi
Professor

May 13, 2013

Certificate

This is to certify that the work in the thesis entitled *GPU Accelerated Parallel Iris Segmentation* by *Kritika Kurani*, bearing roll number *110CS0128*, is a record of an original research work carried out by her under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering.

Banshidhar Majhi

Acknowledgement

All the efforts I have put forward in carrying out this project would have been in- complete, if not for the kind support of many individuals as well as this institute. I would like to express my deep sense of gratitude to all of them. Foremost, I would like thank my supervisor of this project, Prof. Banshidhar Majhi, Department of Computer Science and Engineering, National Institute of Technology, Rourkela, for his incalculable contribution in the project. He stimulated me to work on the topic and provided valuable information which helped in completing the project through various stages. I would also like to acknowledge his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis.

I am obliged to all the professors of the Department of Computer Science and Engineering, NIT Rourkela for instilling in me the basic knowledge about the field that greatly benefitted me while carrying out the project and achieving the goal. I thank the admin and staff of National Institute of Technology Rourkela for extending their support whenever needed.

I also thank my friends and peers who have extended their help whenever needed. Their contribution has always been significant. Finally, I would like to take this opportunity to thank my parents, who have always been a source of inspiration and motivation for me, and also for the love they have provided in stressful periods, which has been a guiding force for the completion of the thesis.

Kritika Kurani

Abstract

A biometric system provides automatic identification of an individual based on a unique feature or characteristic possessed by the person. Iris recognition systems are the most definitive biometric system since complex random iris patterns are unique to each individual and do not change with time. Iris Recognition is basically divided into three steps, namely, Iris Segmentation or Localization, Feature Extraction and Template Matching. To get a performance gain for the entire system it becomes vital to improve performance of each individual process. Localization of the iris borders in an eye image can be considered as a vital step in the iris recognition process due to high processing required.

The Iris Segmentation algorithms are currently implemented on general purpose sequential processing systems, such as common Central Processing Units (CPUs). In this thesis, an attempt has been made to present a more straight and parallel processing alternative using the graphics processing unit (GPU), which originally was used exclusively for visualization purposes, and has evolved into an extremely powerful coprocessor, offering an opportunity to increase speed and potentially intensify the resulting system performance. To realize a speedup in Iris Segmentation, NVIDIA's Compute Unified Device Architecture (CUDA) programming model has been used.

Iris Localization is achieved by implementing Hough Circular Transform on edge image obtained by using Canny edge detection technique. Parallelism is employed in Hough Transformation step.

Keywords: Canny Edge Detection, CUDA, GPU, Hough Circular Transform, Kernel

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Personal identification has become absolutely necessary for large variety of applications like driver licenses, banking, voter id card, etc. The three main kinds of authentication are something you have (token or card), something you know (for example, a password), and something you are (biometric). Tokens and cards of one person can be used by another person. Therefore, even when the cards are recognisable, we can't be sure if the person using the card is the actual owner. Passwords are weak and easily crackable as individuals tend to make passwords that are easy to remember or write them down somewhere easily accessible. Biometrics, provides a secure method of authentication and identification, as they are difficult to replicate and steal.

As the society grows more dependent on digital systems for everything, expectations on these systems are rising. The systems are expected to process an increasing amount of information with an increase in latency which leads to an increasing demand for faster processing. We see an increase in parallelism using multi-core processors. But due to large amount of hardware and logic used, the processors become large and it becomes difficult to incorporate a large number of them onto a single chip.

Therefore, to achieve the benefits of parallelism, a processor that is inherently parallel and massively multi-core, the Graphics Processing Unit (GPU) is used. For attaining this innate parallelism, higher level APIs such as CUDA can be used in

order to accomplish General Purpose processing on GPU (GPGPU).

1.1 Parallel Computing

Parallel computing aims to solve a complex problem by dividing the problem into subsets and solving each problem as a separate thread that can be executed in parallel [2]. The IEEE Standard Dictionary of Electrical and Electronics Terms denes an algorithm as, a prescribed set of well dened rules or processes for the solution of a problem in a nite number of steps [4]. Some processes of an algorithm can run simultaneously in parallel and some must run sequentially one after the other. A algorithm where all the tasks could all be performed in parallel simultaneously due to their data independence is called a parallel algorithm.

1.1.1 Classification of Parallel Systems

The most famous processor taxonomy was proposed by Flynn based on the data and the operations performed on this data [26]:

(i) SISD (Single Instruction Single Data) - A single processor executes a stream of instructions to operate on data stored in a single memory.

(ii) SIMD (Single Instruction Multiple Data) - It describes an architecture where computer with multiple processing elements can exploit data parallelism, a property of the data stream, which signifies a large mass of data of a similar type that needs the same instruction operated on it. Examples of such applications include video compression, medical image analysis , graphics processing and so on.

(iii) MISD (Multiple Data Single Instruction) - Various functional units perform different operations on the same data. Some applications of MISD architecture are neural networks and data flow machines.

(iv) MIMD (Multiple Instruction Multiple Data) - Machines employing MIMD architecture have many processors running asynchronously and independently. Each processor has its local data and runs its own instructions on it. Multicore processors and multithreaded processors are based on this model.

1.1.2 Speedup

The potential benefit of parallel computing is typically measured by the time it takes to complete a task on a single processor versus the time it takes to complete the same task on parallel processors [6].

$$Speedup = \frac{T_s}{T_p} \tag{1.1}$$

where,

T_s is execution time of sequential algorithm

T_p is execution time of parallel algorithm

There are many factors which influence this speedup. Some of the parallel computing overheads are:

Data Dependency

During the implementation, data dependency between two blocks or two statements requires memory access. More number of dependencies imply further access time or more inter processor communication. This degrades the real time performance. The presence of dependency between two computations implies that they cannot be performed in parallel. The fewer the dependencies, the greater the parallelism [7].

Communication Overhead

For parallel problems, the time required in inter-processor communication, not encountered in the serial interpretation, is summed up with the execution time of processors. This communication time adds to overhead. Kernel call to invoke a communication handling routine initiates communication of data between processors. Then, the data is transmitted between processors. Finally, the communication is completed by resolving the send and received data. Lot of time is wasted in the transmission and management of the data. Allocation of communication resources and achieving a connection contribute to communication latency.

Idling

Processing elements in a parallel system may become idle due to many reasons such as synchronization, load imbalance, and presence of serial components in a program.

For many parallel applications, it is impractical to predict the size of the subtasks allocated to various processing elements. Therefore, it is impossible to coordinate the problem division among the processing elements and maintaining uniform workload. When different processing elements have disparate workloads, some processing elements might be idle for a time duration when others are working on the problem. In certain parallel programs, processing elements need to synchronize at some points during execution of the parallel program . If not all processing elements are ready for synchronization at the same time, then those that are ready shortly will be idle until all the processors are prepared. Parts of an algorithm which can't be parallelized, allow only a single processing element to work on it. While one processor works on the serial part, rest of the processors must wait [1].

1.1.3 Communication Techniques

In serial computation, program description is usually helpful for optimization of pivotal sections of code. For parallel computation, not only does the code execution needs to be considered but also communication between the various processes which can induce delays that are malignant to performance. Increase in number of processes, elevate the impact of communication delays on performance. Two methods exist for data communication between processing elements:

Shared Address Space Mechanism

In the shared-memory programming model, tasks share a common address space, which they read and write asynchronously. Various mechanisms such as locks or semaphores may be used to control access to the shared memory. The shared memory address space may be uniformly or non-uniformly accessed. In Uniform Memory Access (UMA), the time taken by a processor to access any memory location is the same whereas in Non Uniform Memory Access (NUMA), the time taken by a processor to access different memory locations may be different [3].

Message Passing

Processors exchange data through communications by sending and receiving messages. Each use their own local memory during computation. Data transfer usually

requires collaborative operations to be performed by all the processors. In a tightly coupled parallel computer, messages are prepared, sent, and received quickly relative to the clock speed of its processors, but in a loosely coupled parallel computer, the time required for these steps is much larger.

1.2 Iris Biometrics

Iris plays a significant role among various available biometric traits to provide a promising solution to authenticate an individual using unique texture patterns [9]. Iris recognition has drawn attention due to highly desirable properties.

Non-invasiveness: Iris recognition technology does not require direct contact between subject and camera. This characteristic also enables iris recognition for passive or covert personal identification.

Uniqueness: Texture details in iris images such as freckles, furrows, coronas and stripes bring out the uniqueness of iris pattern. The irregularly shaped and randomly distributed microstructures of iris patterns make the human iris one of the most informative and reliable biometric traits.

Scalability: Iris region images are normalized into fixed size rectangular region, which enables extremely fast feature detection and matching.

Stability: The morphogenesis of iris occurs during seventh month of gestation and which remain stable throughout life and are protected by body's own mechanisms.

Security: Compared to other biometric systems, iris recognition is more secure due to difficulty posed in live iris forgery.

Iris recognition uses camera technology with near infrared illumination to obtain images of the intricate, detail-rich structures of the iris. Digital template encrypted from these patterns using mathematical and statistical algorithms lead to identification of an individual. Enrolled templates stored in database are searched by matcher engines at speed of millions of templates per second per CPU, and with infinitesimally small false match rates.

Iris is the most significant and promising feature in the eye image as shown in Figure 1.1. The dark circle in the center inside the iris is pupil. The dilation and constriction of pupil and dim and bright light respectively is caused by the muscles

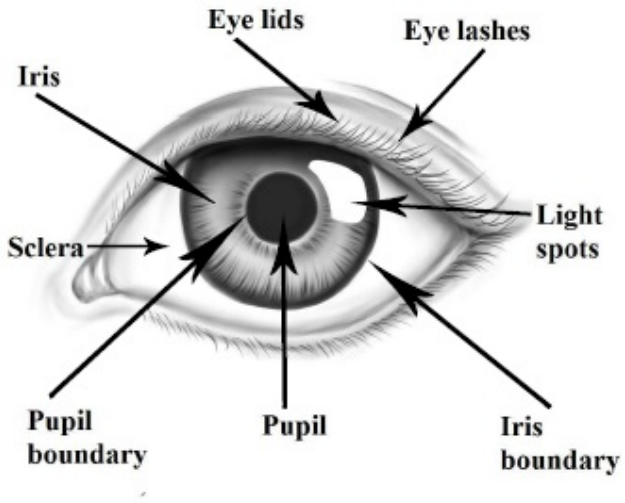

Figure 1.1: Anatomy of a human eye

in the iris. This pupillary motion checks the amount of light entering the eye. The circumference of iris and pupil is called iris and pupil boundary respectively. Sclera is the dense, white, fibrous membrane that forms the covering of eyeball. The eyeball is covered by lower and upper eyelids. The lower eyelid has a smaller degree of motion which is caused by deformation due to eyeball. The upper eyelid is a stretchable membrane having a great freedom of motion, ranging from wide open to close. It forms a cover over the eye [8]. The hairs that grow at the edge of the eyelid and protects the eye from dust are called as eyelashes.

Iris recognition is employed in many systems all over the world. But they use algorithm designed for CPU based system which is sequential in nature. As the number of processors on a CPU increased, the algorithm has been parallelized to take full advantage of multiple processors. Since the number of processors on a CPU are limited, utilization of GPUs might prove as an advance option to accelerate application.

Iris recognition process is basically divided into three steps, namely, iris localization or segmentation, feature extraction, template generation and template matching as shown in Figure 1.2. All these stages involve large amount of computation complexity and their efficiency can be immensely increased by employing parallel algorithms. GPUs initially used for Graphics visualizations and docking are now also applied for applications involving large amounts of computations. And thus,

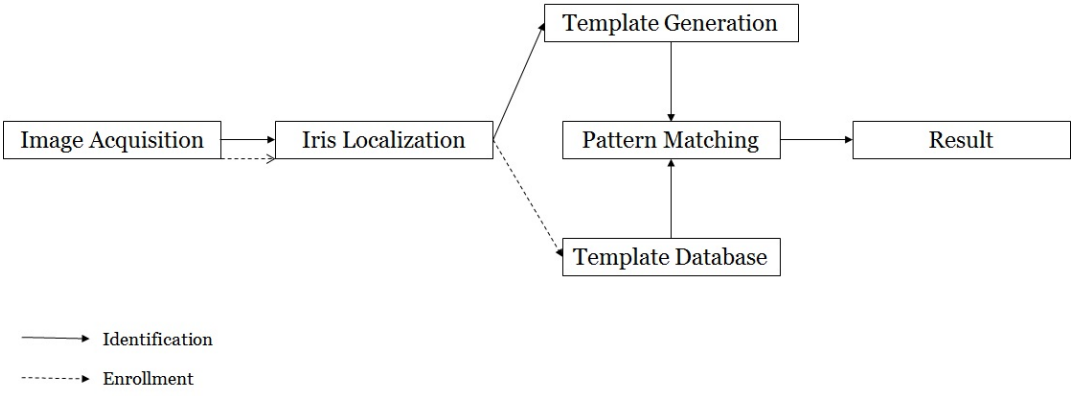

Figure 1.2: Iris Recognition Process

are now also known as General Purpose Graphics Processing Units (GPGPUs).

The altogether performance of iris recognition process depends on the iris segmentation process which is a highly computationally complex task. Iris localization involves finding iris and pupil boundaries in order to separate the region of interest in the acquired iris image from the area which would not be utilized in further processing. A sample of iris image and its localized image are shown in Figure 1.3. The portion of image required for further processing is the region between the boundary of pupil and outer boundary of iris. Also region occluded by eyelids has to be removed.

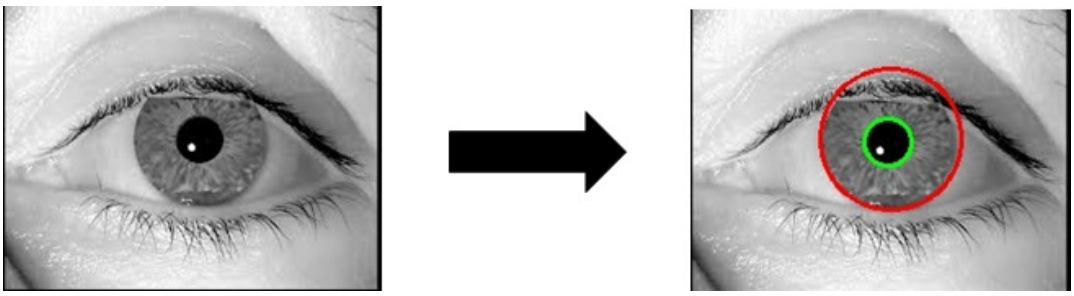

Figure 1.3: Iris segmentation without removal of portion occluded by eyelids

1.3 Motivation

Noteworthy research and efforts are being put in to develop parallel algorithms for iris recognition. This active research forms the basic motive of this project. Also, Graphics Processing Units have seen recent advancements expanding its area of usage to accelerate applications which otherwise demand considerable amount of processing time on conventional systems. Iris recognition is an image processing task which requires considerable amount of computation, and thus gives a prospect for parallel processing application. Parallelization of localization of iris, which is a crucial step in iris recognition, improves performance of the entire system.

1.4 Problem Statement

Each stage of the iris recognition process handles a large amount of processing, and thus takes substantial amount of time, which brings this application into focus. The algorithms developed for iris recognition until now have been sequential in nature. These constituents along with the development of parallel infrastructures like the Graphics Processing Units provide the base for this thesis. The segmentation of the iris is a significant element in the procedure of iris detection and its sequential implementation needs a sizable amount of time. The segmentation process is accelerated by implementing parallel hough transform, which is a sturdy method to encounter the presence of shapes in an image.

1.5 Iris Databases used in Research

The databases used in all the experiments in this thesis that are relevant to the research are CASIA version 1 [35].

1.6 Thesis Organization

Remaining part of this thesis is divided into following chapters:

Chapter 2: Literature Survey

This chapter summarizes the existing work done in iris segmentation and the sequential and parallel approaches developed so far.

Chapter 3: GPU architecture and CUDA

This chapter outlines the basics of GPU architecture and CUDA programming platform. It intends to help to better understand the parallel algorithms implemented using CUDA.

Chapter 4: Parallel Iris segmentation

This chapter introduces the parallelization of existing sequential algorithm implemented for iris localization.

Chapter 5: Implementation and Results

This chapter describes the implementation of CUDA based parallel algorithm and analysis of results acquired for iris localization.

Chapter 6: Conclusion

This chapter presents analytical results of overall achievement.

Chapter 2

Literature Review

Leonard Flom and Aran Safir originally originally proposed the concept of automated iris recognition in 1987, both ophthalmology professors, for which they afterwards obtained a patent entitled "Iris Recognition Technology" [13]. Later they approached John Daugman, a professor at Harvard University, to develop an algorithm build upon their concept [10, 11, 12]. Daugman created an algorithm, for which he got a patent few years later. After the expiry of patent, various other algorithms were created based on Daugman's method and some of them had a performance much improved than Daugman's algorithm.

2.1 Iris Recognition

The human eye iris comprises of complex random texture patterns that can be obtained from the digital image of the human eye, and then encoded into a template, employing image processing techniques. This template can then be stored in a database. The unique information stored in the iris is mathematically represented in the template, allowing the comparison of templates from the database.

The first operational iris biometric system was developed by Daugman at University of Cambridge in 1994 [10]. Near-infrared light source is employed to capture the digital images of eye to control the illumination. The algorithm used for image acquisition resolves focus of the system to maximize the spectral power, thus prov-

ing to be highly robust. Further, iris in the image is found that uses deformable templates. Some parameters and shape of the eye are used to train a deformable template to guide the detection process [14]. Daugman used multi-scale quadrature 2D Gabor wavelets to extract texture information in order to generate a 2048 bit pattern called IrisCode [11]. All the commercial systems in operation today are based on this method.

To sum up, Daugman's process is carried out through:

(i) Detection of the inner and outer boundaries using an Integro differential operator.

(ii) Extraction of texture information employing multi-scale quadrature 2D Gabor wavelets and stored in form of unique binary vectors constituting the IrisCode.

(iii) Detection of the hamming distance between the input IrisCode and the IrisCodes in the database using a statistical matcher (logical XOR operator).

In another significant work succeeding Daugman's work, Wildes *et al.* have represented the Iris texture by constructing laplacian pyramids with different resolution values [15]. In order to detect whether inpput and model image belong to the same class, they have used normalized correlation. R.Wilde's solution has following steps [3]:

(i) Iris localization using Hough Transform.

(ii) Laplacian pyramid(multi-scale decomposition) to represent distinctive spatial characteristics of the human iris.

(iii) Modified normalized correlation process for the matching step.

An iris biometric system was developed at Sarnoff labs which followed a different approach. The authors used a diffused source of light with low level light camera for the image acquisition. Iris and pupil segmentation was accomplished through Hough transform. The iris images were matched using Laplacian of Gaussian filter at multiple scales for producing templates and computing normalised correlation to act as a similarity measure [16].

In Robust encoding of local ordinal measures, proposed by Z.A. Sun, T.N. Tan, and Y.H. Wang, the textures are represented using Gaussian filter [17]. A local

orientation at each pixel is obtained by convolving the gradient vector field of an iris image with a Gaussian filter from normalised iris image. The angle is quantized into six bins. This method has been tested on 2255 images of CASIA database and was found to have a correct recognition rate of 100%. In Efficient iris recognition by characterizing key local variations, put forth by L. Ma, T. Tan, Y. Wang, and D. Zhang, dyadic wavelet transform has been applied on a sequence of 1-D intensity signals around the inner part of the iris to create a binary iris code [18]. The system achieved 100% correct recognition rate with an EER of 0.07%. In yet anpther approach, Modified Log-Gabor filters are used because unlike Log-Gabor filters Gabor filters are not bandpass filters [19].

2.2 Iris Segmentation

Iris preprocessing involves finding the iris and pupil boundaries from the eye image. These boundaries are presumed to be circular. To further improve the segmentation performance few authors have also worked on detecting eyelids and eyelashes [20]. Figure 2.1 shows localized iris image after removal of eyelids and eyelashes.

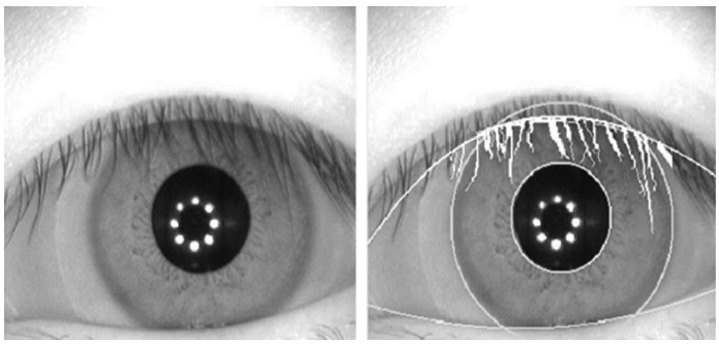

Figure 2.1: Removal of eyelids and eyelashes in iris localization

The iris recognition algorithms were originally designed to be deployed on CPU based systems and thus, were serial in nature. The work began with Daugman and various approaches have been developed ever since. As stated by Daugman's algorithm, iris recognition consists of three steps. First step is to obtain an iris image using digital video camera. Near infrared illumination is used for image acquisition so that a darker iris will show more details [11]. Then iris segmentation extracts the

region of interest from the iris image. Then Gabor wavelets are employed for feature extraction to generate IrisCode, a 2048 bit binary code. Then finally matching step is performed.

2.2.1 Daugman's Method

Daugman proposed an integro-differential operator which searches for the maximum in the blurred partial derivative of the image [12]. He used a model of iris where the iris and pupil boundaries are circular thus the boundary of circle can be described using three parameters: center of the circle x0, y0 and radius r. The operator is defined as:

$$max(r, x0, y0)|G_\sigma(r) * \frac{\partial}{\partial r} \oint_{r,x0,y0} \frac{I(x,y)}{2\pi r} ds| \tag{2.1}$$

where, I(x, y) is the image of the eye and G_σ(r) is a blurring function. Different values of σ provide different levels of smoothing. The operator searches the entire image domain (x, y) for the maximum in the blurred partial derivative with respect to the increasing radius (r) of the normalised contour integral of the image I(x, y) along a circular arc ds with centre coordinates (x0, y0) and of radius r. The operator in totality behaves as a circular edge detector. Absolute value of partial derivative is taken into account to address the issue that the pupil is not always darker than the iris.

The Iris recognition process does not take into account the iris color, although there is a considerable variation in the color of the iris ranging from blue to green and also black to brown , rather the process focuses on the texture patterns like the crypts, furrows, corona and rings. After the iris segmentation, features of the iris are extracted for comparison. The most considerable trouble faced in iris comparison is that, all the iris images are of various sizes. The iris features need to be invariant to change in scale, size, orientation, etc. The size of iris in an image is affected by the distance between the eye and the camera. The issue of linear deformation of the iris pattern caused by change in orientation of iris due to movement of eyeball, camera position, head tilt, etc, and alteration in illumination that causes dilation

or contraction of pupil has been tackled by Daugman by mapping the iris into a dimensionless polar coordinate system [12]. The matching score is determined by the similarity of the two iris representations.

2.2.2 Wilde *et al.'s* Stratergy

In Wilde's scheme, first the intensity information of the image is converted into a binary edge map [15, 21]. Then the edge points vote to manifest particular contour parameter values. Using a gradient based edge detection scheme the edge map is acquired. This is accomplished by thresholding the magnitude of the image intensity gradient.

$$\nabla G(x,y) * I(x,y), \qquad where \quad \nabla = (\partial/\partial x, \partial/\partial y) \tag{2.2}$$

and

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x0)^2+(y-y0)^2}{2\sigma^2}} \tag{2.3}$$

The voting procedure is implemented using Hough Transforms on the parametric definitions of the iris boundary contours [21, 22, 23]. Following equation represents the hough transform :

$$H(x_c, y_c, r) = \sum_{j=1}^{n} h(x_j, y_j, x_c, y_c, r) \tag{2.4}$$

where

$$h(x_j, y_j, x_c, y_c, r) = \begin{cases} 1, & \text{if g}(x_j,\ y_j,\ x_c,\ y_c,\ \text{r}) = 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

with

$$g(x_j, y_j, x_c, y_c, r) = (x_j - x_c)^2 + (y_j - y_c)^2 - r^2 \tag{2.6}$$

2.2.3 Other Schemes

The method proposed by Ritter *et al.* uses active contour models for pupil localization [24]. To further improve the time of localization, few authors have also worked

towards improving the localization performance by the recognition of eyelids and eyelashes [20]. The localization and normalization phases of Iris localization assume the pupil and iris boundaries to be circular in shape. Considerable work has also been done for the detection of eyelids and eyelashes.

Few authors have proposed thresholding based approaches for pupil localisation. The authors in *Fast algorithm and application of hough transform in iris segmentation*, take the pixels below a threshold as pupil and then find the circles using Hough transform and edge detection in the limited area [25]. Additionally, an iris localization based on adaptive threshold is proposed in [26]. In this proposition, the iris image is first dissected into rectangular regions and intensity means is acquired for each of the region. Then the minimum value of the mean is used as threshold for converting the image into binary. In another approach, split and merge algorithm is utilized to detect connected regions in the image. Yet another method was proposed where the concept for iris localization which was similar to Daugman. Firstly the irregularities are separated using bilinear interpolation, then the candidate locations are produce to provide the initial conditions for pupil and iris boundary. After that, pupil and iris parameters are recovered for each seed (x, y) [27] . The algorithm proposed by B. Bonney et al. finds the pupil using least significant bit planes [28]. The algorithm proposed in Real-time iris segmentation based on image morphology uses a novel sector based method for the iris segmentation [29]. Using an adaptive thresholding the iris image is converted into binary and the inner iris boundary is identified. The outer iris boundary is detected by first taking the sum of concentric circles of incrementing radii starting from the pupil radius. Then the difference of adjacent circles is obtained. The circle having the maximum difference is taken as the outer boundary of the iris. Also the noise caused by eyelids and eyelashes is removed by employing a sector based approach.

The technique suggested by Huang *et al.* improves the iris localization time by obtaining the outer iris boundary in the rescaled image [30]. Scaling the image for an optimum fraction of the original gives improved performance than the original image. Various other authors have also put forth methods to detect the eyelids and eyelashes. The detection of eyelids involves the search for two curves, which satisfy

the polynomial equation of the form:

$$x(t) = at^2 + bt + c, t \in [0, 1] \tag{2.7}$$

Kong and Zhang suggested a method for the eyelashes detection. He divided the eyelashes into two categories [31]. First one is separable eyelashes which can be easily separated from the image and the second one being multiple eyelashes, which are bunched together and are found overlapping in the eye image [3]. One dimensional Gabor filters can be applied to detect separable eyelashes since the gaussian smoothing of a separable eyelash leads to a low output value. Variation of intensity method is employed to identify multiple eyelashes. When the variance of the intensity values in a window is lower than a threshold value then the center of the window is a point on the eyelash.

2.3 Parallelization of Iris Recognition

Works in parallelization of iris recognition began quite late. It received attention with the progress of parallel computing infrastructure like multicore architecture and advanced technologies for the development of parallel programs. Daugman himself instigated the application of parallelism in the matching of the IrisCode generated from the image to the template IrisCodes stored in the database. The template can now be compared to a stored template using the fractional Hamming distance as the measure of closeness.

$$HammingDistance = \frac{||(CodeA \bigoplus CodeB) \cap MaskA \cap MaskB||}{||MaskA \cap MaskB||} \tag{2.8}$$

The $\bigoplus$ operator is the EXCLUSIVE OR operation to detect disagreement between the pairs of bits that represent the directions in the two templates, $\cap$ represents the binary AND function, and masks A and B identify the values in each template that are not corrupted by artifacts such as eyelids and eyelashes and specularities [32]. The denominator of equation 2.8 make certain that only the bits that are important are included in the calculation, after artifacts are deducted. Rotation mismatch(due to head-tilt) between irises is managed with leftright shifts of the coded one-bit template to choose the minimum Hamming distance. Lower the HD

value, higher the match between the two irises that are compared. The fractional Hamming distance of two templates is analogized to a predetermined threshold value and a match or nonmatch statement is made. The speed performance is highly affected by time-consumption of template matching, since it is related to the size of the database that it is being matched against.

Depending on the manner the computer handles the bitwise operations, the compuation of Hamming distance can be done very quickly. Bitwise operations are performed directly at the hardware level. It takes only 1.7 seconds to compare the IrisCodes of a million templates on a 2.2 GHz computer [3]. Such high performance levels enable the iris recognition system to be used for large scale applications where the comparison of calculated IrisCode needs to be done against millions of templates. Since the IrisCode is of size 2048 bits, the comparisons could be done in parallel in terms of 32 bit chunks.

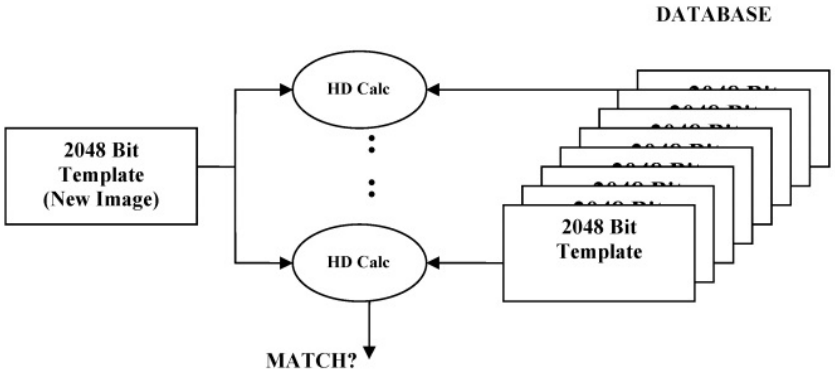

Figure 2.2: Comparison of new template with the templates stored in database

Recently, many authors have suggested new approaches towards parallelizing iris recognition. In 2011 Fatma Zaky Sakr *et al.* introduced an alternative technique by employing direct parallel processing, utilising the Graphics Processing Units [33, 34]. They proposed parallelization of iris recognition after the evaluation of the most tedious processes involved in Iris Recognition. Precisely they concentrated on the matching and identification phases since they need the maximum volume of computation when the size of the database to be compared against is extremely high. They have utilised Daugman's algorithm in every phase of the process. The speedup

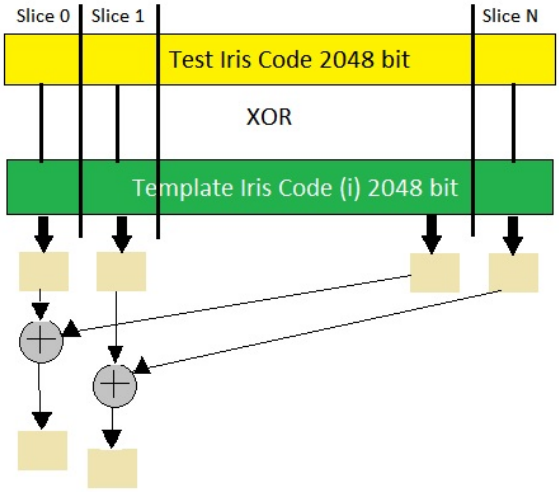

Figure 2.3: Parallel computation of Hamming Distance

obtained was around 10 to 15 times in the matching and identification phases and around 1.3 times when taking all the stages of iris recognition into consideration [33].

According to Daugman's methodology, identification process comprises of matching the 2048 bit IrisCodes for the images by evaluating the Hamming distance between the input code and the template codes. The method put forth by F.Z. Sakr *et al.* scrutinizes this fact by matching the codes in parallel [33]. The code is segregated into bit slices and then a single bit slice of 32 bits is analysed on a single processor by XORing the input code with the template codes as shown in Figure 2.3.

Later, they proposed another method where they focused on localization and extraction processes, considering these are the two highly time consuming steps compared to the matching step [34]. They realized a speedup of 9.6 and 12.4 for the two processes and a total speedup of 12.4 taking into account all the processes.

Ryan N. Rakvik *et al.* proposed a parallel processing alternative utilizing Field Programmable Gate Arrays (FPGAs), which accelerates the application to a higher level [35, 36]. They adopted parallel processing to speed up most time consuming phases involved in the process of iris recognition. Iris Segmentation, template generation and matching phases require a lot of computation and its has been parallelized on an FPGA system, giving an average speedup of 9.6 for the segmentation step,

324 for template generation step and 19 for matching step when compared to a CPU based system.

Chapter 3

GPU Architecture and CUDA

GPUs abbreviated as Graphical Processing Units are processors fashioned to reduce the workload on the CPU, short for Central Processing Unit, when working on video or graphic concentrated applications. Over the time, GPUs have evolved to be an essential component of computing platforms for interactive simulations, high-end 3D rendering, and video games. GPUs could be perceived as accelerators or co-processors. They do not preclude the demand for a CPU. Specifically, CPUs are devised to execute serial applications as speedily as possible. Latterly, CPUs have developed into multicore and are capable of achieving some extent of multithreaded parallelism, but are still optimised for serial execution. However, GPUs are devised for densely threaded parallelism instead of serial computation that can be executed on the CPU. Hence, an application developer may apply a heterogeneous execution model to carry out densely parallel sections of an application on the GPU and the serial parts on the CPU [37].

Recently, GPUs have drawn great attention for their use as a more general purpose computing element rather than just video and graphic accelerator [38]. Thus, they are also sometimes referred to as General Purpose Graphics Processing Units (GPGPUs). This course was launched with shader languages. It has lately evolved into an entire series of development tools distributed by the major GPU chipset developers to ease general purpose GPU computing. Prime vendors of GPUs are nVidia, AMD (formerly ATI) and Intel. While Intel supplies low-end graphics pro-

cessors desegregated into varied mother board chipsets, nVidia and AMD participate in the competition for the high-end GPU market. nVidia and AMD have alternative proprietary GPU platforms each of which is compatible only with their own hardware [37]. Precisely, AMD delivers the ATI Stream SDK while nVidia presents the CUDA Toolkit. Besides, a vendor independent standard, OpenCL, is being developed to execute HPC applications to be expanded independent of hardware [39].

3.1 GPU Architecture

The GPUs are highly parallel, multithreaded programmable devices capable of rendering realtime graphics applications. It has tens, hundreds or sometimes even thousands of cores with tremendous power capable of high precision floating point arithmetic which is the requirement of realtime video processing [3]. Graphics cards have a very high memory bandwidth of at least 64 bits in the modern processors which allows large amounts of data transfers in a single flow and a large number of on chip registers which allows it to hold several variable values while computation [40].

Compared to the number of floating point operations per second(FLOPS) for a high end CPU, FLOPS for GPU are exponentially higher. A standard GPU has a computation speed of few hundred gigaflops while for a high end CPU its few tens of gigaflops. High end GPUs have computation speeds in few teraflops. The reasoning for such high computation speed is that GPUs are proficient in performing compute intensive highly parallel tasks such as graphics rendering. It is fashioned in a manner such that large number of transistors are dedicated to data processing instead of flow control and data caching.

The GPUs are ideal for embarrassingly parallel applications *i.e.* applications where scant endeavour is required to divide the problem into number of parallel tasks and the arithmetic operations are much higher than memory operations. Flow of data between processors is very less since the same program is completed on different data sets on different processing elements. The memory latency is therefore not visible and it remains hidden under the heavy calculations that take place inside the processors [40].

Applications which require usage of large data sets can employ a parallel programming model for speed up. In 3D image rendering applications like computer gaming programs, large number of pixels are mapped onto processing cores for independent parallel processing. Similarly video encoding and decoding applications can map a subset of data like an image block onto a separate processor for execution [40].

3.1.1 GPU Processing Elements

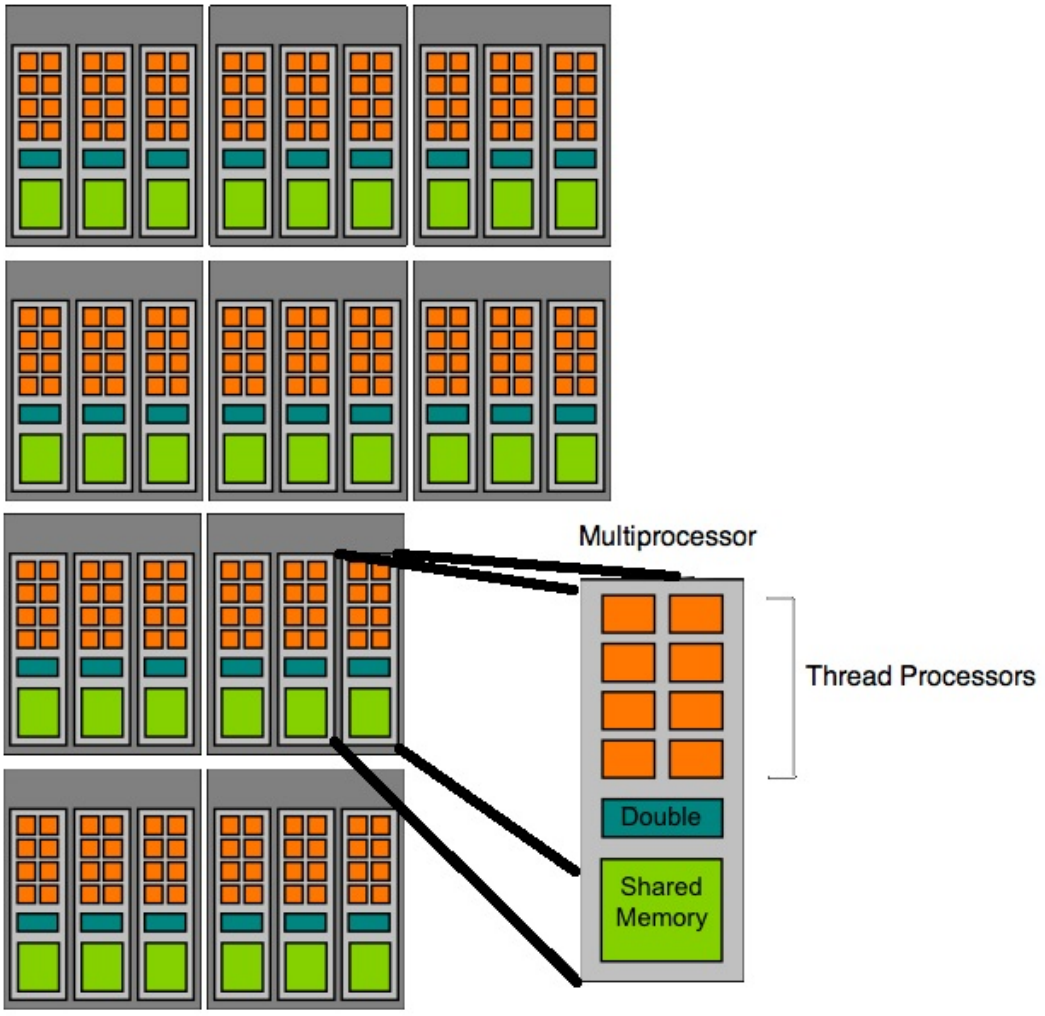

Figure 3.1: Diagram of multiprocessors in GPU [41]

The GPUs endure several Streaming Processing Clusters (SPC). All SPCs com-

prise many streaming multiprocessors (SM), every one of which contains streaming processors (also known as cores) that share admission to local memory. Every core contains a fused multiply-adder capable of single precision arithmetic (shown in Figure. 3.1). Alongwith single precision operations, all the SMs contain one 64-bit fused multiple adder for double precision operations. Therefore, a SM contains 8 single precision streaming processors for every double precision unit. SMs also contain 16KB of shared memory and 16KB of registers [41].

3.1.2 GPU Memory Organization

GPUs manufactured by nVidia have various memory spaces usable with its own advantages and constraints. Understanding the memory hierarchy and effectively utilizing it can result in better performance for GPU based applications.

Device registers are the fastest memory available, which is accessible without latency on each clock cycle, just as in CPU. Each streaming multiprocessor (SM) has 16KB of registers. Each thread residing on GPU has its own register. These registers cannot be shared among threads. Also, all the registers are dispensed among all the threads that reside concurrently on the GPU. Therefore, if CUDA kernels utilize oodles of registers, the device will not be able to complete as many threads simultaneously.

Each multiprocessor has *shared memory* space which is a 16KB region with very short access times. The purpose of shared memory is to act as a means for fast communication between threads. However, due to its speed, it can also be employed as a programmer controlled memory cache.

After this, GPUs have DRAM (Dynamic Random Access Memory), or *device memory*, accessible at relatively 150x latency in comparison to registers or shared memory. This memory is logically partitioned into four regions: *global memory, local memory, texture memory,* and *constant memory*. Global memory is handy to all threads and is relentless between GPU calls. It resides off chip from the multiprocessors, resulting in 100x access time compared to shared memory. Local memory, specific to individual threads can also be used as a substitute when the compiler is incapable to fit desired data into the device's registers. In such case registers are said to spill to local memory. Texture memory is read-only with a small texture cache optimized for manipulation of textures. Texture memory is

advantageous than global memory since the memory reads do not require an access pattern to obtain better performance and addressing calculations are done outside the kernel. Constant memory is also a read-only part which also has a small cache of 8K constant memory.

At the end, *host memory* (system's main memory) is accessible obliquely and comparatively slower to the GPU. Host memory space is handy only to the GPU when copied over the PCI-Express (Peripheral Component Interconnect Express) bus to the GPU's device memory.

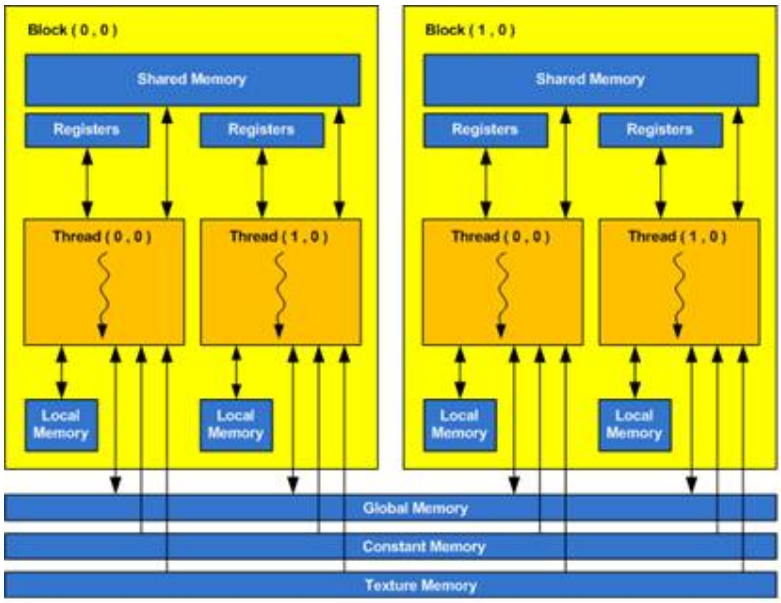

Figure 3.2: GPU Memory Hierarchy

3.2 CUDA

nVidia has released a complete software development toolkit for development of software for their GPUs, CUDA (Compute Unified Device Architecture). This tookit includes a GPU compiler (nvcc) and a kernel profiler. CUDA is only adaptable with nVidia devices. It is a parallel computing platform and programming model which employs the Parallel Compute Engine to handle the threads running on the GPU. It is the job of the Parallel Compute Engine to lineup the threads to be executed

on the GPU and map them to the vacant cores such that it complies with the CUDA architecture and specification. The CUDA platform provides a bedding for expansion to the C programming language alongside petty additions and limitation. To control hardware specific to the GPU, some additional language constructs are added [40]. Some constraints such as a lack of function pointers and recursion exist, because all functions are inlined by the compiler automatically. CUDA provides support for high level languages like C, C++, FORTRAN, DirectCompute and OpenACC, while third party wrappers are available for other languages like Python and Java [40].

3.2.1 Building Components

CUDA C allows to define C functions called *kernels* , which are executed in parallel by N different CUDA threads.

A CUDA kernel is defined as:

__global__ void kernel_function_name (parameter list)

__global__ declaration specifier indicates that code is executed on device and is invoked from host code.

A *thread* is the basic unit of a CUDA program and it serves as the main component of a CUDA program. GPU can handle thousands on concurrent threads, CUDA allows a kernel launch to specify more threads than the GPU can execute concurrently which helps to amortize kernel launch times. Threads are grouped into *blocks* which in turn are grouped into a *grid*. It is in the hand of the programmer to specify the number of blocks to be created for execution of the parallel application. The programmer also defines the maximum number of threads that can be created per block.

$$NumberofBlocks = \frac{Totalnumberofthreads}{Numberofthreadsperblock} \tag{3.1}$$

A CUDA kernel is called by following execution configuration syntax:

kernel_function_name <<<Number of blocks, Number of threads per block >>>

The number of CUDA threads that execute that kernel for a given kernel call is specified within <<<... >>>

Under CUDA, it is possible to have upto 8 blocks resident on multiprocessor at any time, as long as that multiprocessor has enough registers and shared memory to hold those blocks. However, there is a limitation on number of threads per block. CUDA arranges the threads and blocks in 1D, 2D or 3D array. Variables assigned to retrieve thread location are: *threadIdx, blockIdx, blockDim, gridDim.*

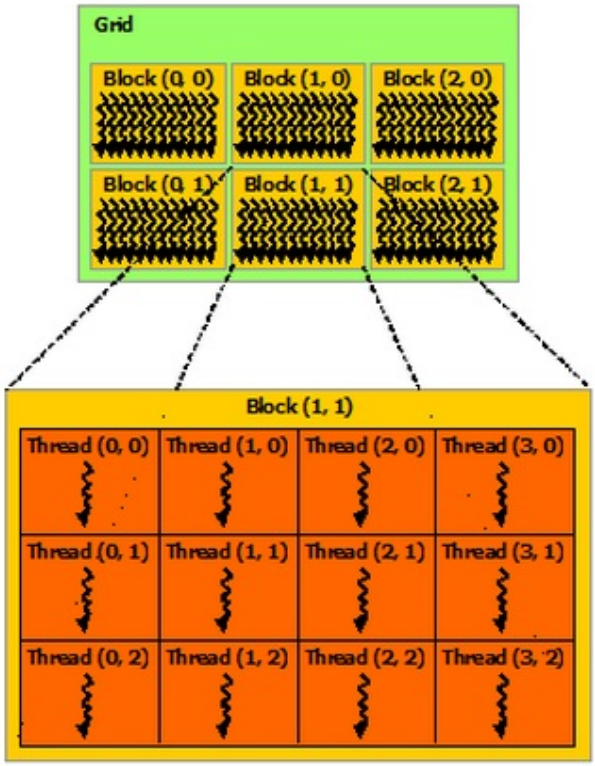

Figure 3.3: A thread of blocks [40]

3.2.2 Scalability

The programmer does not need to ponder upon the optimization of the program for the GPU being used. This accounts for automatic scalability. The same application

will successfully run on any GPU with any number of cores. It is the solitary responsibility of the GPU Compute Engine to lineup the threads appropriately on the device with a given architecture.

Blocks may execute in arbitrary order, sequentially or concurrently, and parallelism increases with resources. This forms the basis for scalability of CUDA programs. Blocks are p;aces one on each of the SMs. When all the threads in a block terminate, new block is placed on that SM.In case threads of some block needs to access the device's global memory, the multiprocessor will sit idle while the request is filled. To minimize the idle time encountered by processor, CUDA scheduler may place a second block on the same SM and execute the threads of the latter blocks. Since two blocks will be residing on a single multiprocessor, the processor keeps busy by switching between blocks, whenever threads of one block need to wait.

Thus a GPU with more number of multiprocessors will automatically run the program in less time compared to a GPU with less number of multiprocessors.

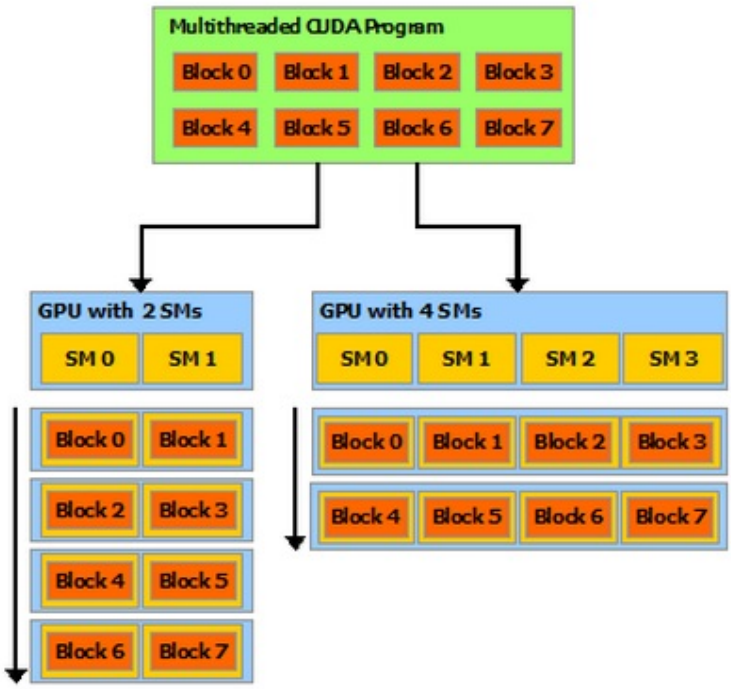

Figure 3.4: Automatic Scalability [40]

3.2.3 CUDA Programming Model

CUDA utilizes a SIMT (Single Instruction Multiple Thread) programming model, alike the SIMD (Single Instruction Multiple Data) model usual in vector processors. All the threads executing on the GPU carry out the same instruction set simultaneously, on different datasets. But, unlike time honored SIMD architectures, CUDA allows threads to bifurcate at a performance cost. When branches are encountered in the code the divergent threads will turn inactive till the conforming threads finish their separate execution. When execution unifies, the threads further proceed to operate in parallel. Full advantage of the GPU hardware can be realised if the code is written in a way that lessens thread separation.

Thus, the heart of CUDA's programming model lies in the thought that there exists abundance of data that has to be operated on in a consistent way by some instructions [37]. The model requires a host (or CPU) and one or more devices (GPUs) each of which has plenty of arithmetic execution units to accomplish many calculations in parallel. This model can be applied to various applications such as bioinformatics, physics simulations, image processing,etc. which incorporate very large regular arrays of data requiring regular processing.

GPU and CPU code co-exist within a lone source which also comprises of GPU kernels. The CPU executes host code till a GPU kernel is invoked as shown Figure 3.5. This is when, the GPU commences many, possibly thousands, of parallel threads, as a consequence, control returns asynchronously to the CPU. If some other GPU call is initiated which invokes another kernel or when a GPU memory transfer is started, the GPU waits until the first call finishes. Therefore only one kernel execute on current GPUs at a time.

Hence, a CUDA program consists of the following elements :

- Allocation of host memory for the required variables
- Allocation of device Memory for the variables
- Copying values from host to device
- Computation of the results on the device by launching CUDA kernels.
- Copying the computed values back to the host

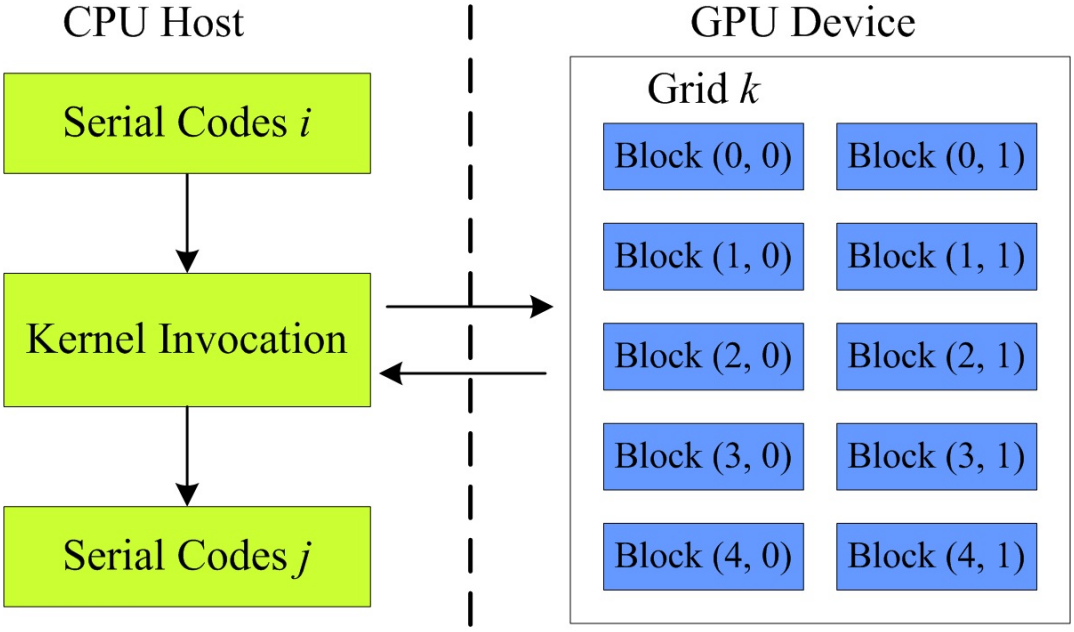

Figure 3.5: Kernel Execution

3.3 Strengths of GPU platform

Performance. The most apparent benefit of GPU computing is the usable performance at low cost. Some nVidia GPU cards provides 480 CUDA cores and can accomplish a peak GFLOP rate of 1788.48 for single precision. The fastest desktop CPUs have a theoretical peak rate of around 70 GFLOPs.

Inexpensive and Convenient. Compared to customary supercomputing platforms, such as clusters, GPU computing has several noteworthy advantages. GPUs are available at low costs compared to desktop systems with CPU cores that would implement calculations at an equivalent rate, presuming that communication costs could be kept unknown. Further, a GPU system does not need a specialized server room with additional energy and maintenance costs. Thus, GPU computing provides many advantages, in terms of space, cost, and convenience, over the cluster as an alternative platform.

Ubiquitous. CUDA compatible hardware is formerly available and set up on millions of machines. Even though the GPUs existing in older desktops and laptops are unable to deliver combative performance, the ubiquity of compatible hardware provides a chance for anyone to learn about and experiment with the technology. In

this sense, CUDA is the first readily available ubiquitous HPC platform [37].

3.4 Limitations of GPU platform

The memory constrictions. The CUDA platform has two memory constrictions. Firstly, chokepoint of communication between the device GPU and the host CPU. Transfer of data to and from GPU poses great difficulties. Similarly, while functioning on accelerated device registers, GPU behaves optimum, compared to while working with device memory. GPU conducts at extreme speed when it executes many operations for each memory access to hide this latency. Applications requiring random access to very large datasets are usually memory bound and are unable to attain near the peak capabilities of a GPU.

Programming Difficulty. In spite of the fact that GPU programming has progressed substantially well and GPU chipset developers made considerable progress in designing tools for producing GPU accelerated software promptly accessible, GPU programming is yet not an effortless task that is undoubtedly suitable for a novice programmer. Competent GPU programming demands deep understanding of a modern memory hierarchy and programming model with new terminology and concepts.

Precision. The prevailing generations of GPUs are mainly used with single precision. Although the current GPU generation supports double precision, there is a considerable performance penalty [37].

Chapter 4

Parallel Iris Localization

Most of the currently existing Iris segmentation methods are sequential in nature. The target of this thesis is to implement an existing Iris Segmentation method on a parallel system to speedup the task. The fidelity of the approach has already been proven by their developers. To accomplish the process of Iris segmentation, Hough transform method will be used for the detection of circular boundaries, and, GPU with CUDA architecture will act as the parallel system for implementation. The motive for using Hough transform is it's robustness and efficiency in dealing with noisy and incomplete data, unlike several threshold based approaches.

4.1 Hough Transform

Conventional Hough Transform is a very effective method for identification of basic shapes present in an image. Initially the Hough transform was employed to detect lines in an image but later it was extended for the determination of other shapes like circles and ellipses. The thought of Hough Transform was proposed by Richard Duda and Peter Hart in 1972, and they termed it as Generalized Hough Transform [21].

An edge detection algorithm finds out the points in an image where the brightness changes sharply or has discontinuities. The result of an edge detection algorithm is used as an input for the Hough transformation step. Succeeding algorithm then

detects the required shape present in the edge image. Hough transform is specially effective for noisy data where there are several imperfections in the image and also in the edge detection algorithm, but it is accompanied by a cost of computational complexity which is quite large.

Following equation represents the hough transform :

$$H(x_c, y_c, r) = \sum_{j=1}^{n} h(x_j, y_j, x_c, y_c, r) \tag{4.1}$$

where

$$h(x_j, y_j, x_c, y_c, r) = \begin{cases} 1, & \text{if g}(x_j,\ y_j,\ x_c,\ y_c,\ \text{r}) = 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

with

$$g(x_j, y_j, x_c, y_c, r) = (x_j - x_c)^2 + (y_j - y_c)^2 - r^2 \tag{4.3}$$

Hough transform uses the extracted edge features and groups them to a befitting set of lines, circles or ellipses. It makes this possible by performing grouping of edge points into object candidates by performing an explicit voting procedure over a set of parameterized image objects. Since the iris and pupil boundaries are assumed to be circular, Hough circular transform is used for their detection.

Hough Transformation can be described as a modification of a point in the x, y-plane to the parameter space. The parameter space is determined according to the shape of the object of interest. The circle is easier to represent in parameter space, when compared to other shapes, because the parameters of the circle can be directly transferred to the parameter space
The equation of a circle is:

$$r^2 = (x - a)^2 + (y - b)^2 \tag{4.4}$$

where,

r is the radius of the circle and,

a *&* b are the center of the circle in x *&* y direction respectively.

The parametric representation of circle is

$$x = a + rCos\theta \tag{4.5}$$

$$y = b + rSin\theta \tag{4.6}$$

Changing the value of θ from 0 to 360 degrees generates the complete circle. In the circular Hough transform we try to find a triplet (x; y; r) which has a high probability to be a circle in the image. Assuming that the radius is known. Every point in the image space will be equivalent to a circle in the Hough space, i.e. Parameter space. Rearranging the above equations we get:

$$a = x - rCos\theta \tag{4.7}$$

$$b = y - rSin\theta \tag{4.8}$$

These represent circle in the parameter space corresponding to a particular point (x; y) in the image space. To detect the iris and pupil boundaries, Hough transform works as follows, at each edge point result from previous edge detection step we draw a circle with center in the point with the desired radius. This circle is drawn in the parameter space. Figure 4.1 illustrates this process.

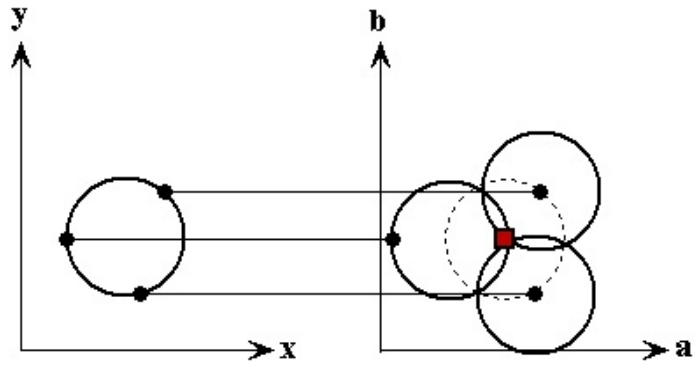

Figure 4.1: Image space to Hough space

The result of edge detector algorithm serves as an input to Hough transform. The maximum and the minimum values of radius r_{max} and r_{min} need to be defined over which to search for a circle. For each radius r such that $r_{min} \leq r \leq r_{max}$ an accumulator array houghspace is produced with a size equal to that of the image, i.e. $width * height$ of the image. The accumulator array is initialized to trivial value 0. For every edge pixel in the image, circle of the particular radius r is constructed

and votes are added to accumulator array cells for all the pixels lying on that circle. A high value in the accumulator shows that the pixel falls on the intersection point of various circles in the parameter space. This intersection point corresponds to a potential point for the center of a circle in the image space.

Since we do not know the radius of the circle we are targeting for, we search for all possible circles with radius up to the diagonal of the image being searched, since no circle with radius greater than that, be present in the image. A search for the circles with all possible radii is done, and accumulators are maintained for all the searches, i.e. One accumulator array for the search of a circle with a particular radius.The highest values from each accumulator cell are gathered in another list, and the highest values in this list represent the potential circles [3]. The process is shown in Algorithm 1.

4.2 Canny Edge Detection

There are many methods for edge detection, but one of the most optimal edge detection methods is Canny edge detection [45]. It gets a greyscale image and produces a binary map analogous to the identified edges. It begins by a blur operation followed by the generation of a gradient map for each image pixel. A non-maximal suppression stage sets the value of 0 to all the pixels of the gradient map that have neighbours with higher gradient values. Further, the hysteresis process uses two predefined values to classify some pixels as edge or non-edge. Finally, edges are recursively extended to those pixels that are neighbours of other edges and with gradient amplitude higher than a lower threshold.

4.3 Parallel Hough Transform

Algorithm 2 describes the mapping of Hough transform algorithm onto GPU. The initial steps are similar to the serial algorithm. For each possible radius value, each CUDA block contains the maximum number of allowable threads and thus the grid size becomes $width*height/blocksize$. Each thread evaluates the position for a pixel in the hough space, then loads the value of the image pixel from the circle edge image and updates the houghspace accumulator array for the particular radius.

Algorithm 1 SerialSegment (Image)

1: *img* ← *LoadImage(Image)*
2: *edgeImg* ← *Canny(img)*
3: Define maximum and minimum values of radius, r_{min} and r_{max}
4: Define **Point***center*, *radius* and *maxval* an initialize to 0
5: Define array *imgValue* of size *Image*
6: Initialize *imgValue* with value 255 for all edge pixels and 0 for others
7: **for** $r \leftarrow r_{min}$ to r_{max} **do**
8: Define array *houghspace* of size *Image* and initialize it to 0
9: **for** every *pixel* in *imgValue* **do**
10: **if** *pixel* represents an edge point **then**
11: Draw circle of radius r and increment houghspace cell value for all the pixels lying on that circle
12: *max* ← Maximum value of a *pixel* in *houghspace*
13: **if** *max* > *maxval* **then**
14: *maxval* ← *max*
15: *center* ← *pixel*
16: *radius* ← r

Algorithm 2 ParallelSegment (Image)

1: *img* ← *LoadImage(Image)*
2: *edgeImg* ← *Canny(img)*
3: Define maximum and minimum values of radius, r_{min} and r_{max}
4: Define **Point***center*, *radius* and *maxval* an initialize to 0
5: Define array *imgValue* of size *Image*
6: Initialize *imgValue* with value 255 for all edge pixels and 0 for others
7: **for** $r \leftarrow r_{min}$ to r_{max} **do**
8: Define array *houghspace* of size *Image*
9: *blocksize* ← *1024*
10: *gridsize* ← *(width*height)/blocksize*
11: Call **parallel** *initializeHoughspaceKernel(cudaHough, height, width)*
12: Call **parallel** *HoughKernel(imgValue, r, cudaHough, height, width)*
13: *max* ← Maximum value of a *pixel* in *houghspace*
14: **if** *max* > *maxval* **then**
15: *maxval* ← *max*
16: *center* ← *pixel*
17: *radius* ← *r*

Algorithm 3 initializeHoughKernel (cudaHough, height, width)

1: *pixel* ← Position in *houghspace* corresponding to current thread
2: *houghspace[pixel]* ← 0

Algorithm 4 HoughKernel (imgValue, r, cudaHough, height, width)

1: *pixel* ← Position in *houghspace* corresponding to current thread
2: **if** *pixel* represents an edge point **then**
3: Draw circle of radius r and increment houghspace cell value
for all the pixels lying on that circle
4: Synchronize threads

4.4 Time Complexity

In the serial algorithm, the extreme cases are $radius = 1$ (minimum radius value) and $radius = d$ (maximum value) where d is the diagonal of the image, given by $d = \sqrt{width^2 + height^2}$. Hence the total number of possible radius values is equivalent to the diagonal of the image, d.

For each radius value radius, we analyse $width * height$ number of pixels and draw circles around the edge pixels which again leads to calculation of 72 points. This time complexity of serial algorithm is proportional to $d * (width * height) * 72$

$$T_s = O(d * width * height) \tag{4.9}$$

As for the parallel algorithm, we again handle d number of radius values. Again for each radius value we create blocks with maximum number of threads and grid with $(width * height)/blocksize$ number of blocks. Each thread deals with a single pixel and the 72 pixels on its circle if its an edge pixel.

Since $width * height * 72$ number of threads run in parallel, the time complexity can be given by:

$$T_p = O(d) \tag{4.10}$$

The above equation holds if there are sufficient number of cores to handle each thread independently. For lesser number of cores the time complexity will be a multiple of T_p.

Chapter 5

Implementation and Results

This chapter provides the details of the results obtained for serial and parallel iris segmentation and its analysis. The localization from proposed parallel algorithm is compared against the serial algorithm. Since iris segmentation is the fundamental and most significant step in iris recognition, there should not be any errors. The basic idea behind the parallel implementation of Iris Localization algorithm is the fact that the data can be decomposed into smaller parts that can be executed in parallel.

5.1 Implementation Environment

For parallel implementation of the algorithm, a low end GPU was chosen which still provides appreciable results in terms of speedup of the application in comparison to CPU based serial algorithm.

Total number of CUDA cores affect the total execution time of the algorithm. More the number of cores, faster is the execution and lesser is the time consumed, but it comes with limitations. The bandwidth of memory interface present between the GPU cores and GPU memory plays a significant role.

The details of the environment and hardware used for the application are shown in the Table 5.1.

Item	**Details**
GPU	NVIDIA Quadro K1000M
GPU Specifications	192 CUDA cores, 2048 MB GPU Memory
Processor	Intel Core i7 3720QM, 2.60GHz
System Memory	8GB
Operating System	Windows 7

Table 5.1: Details of Implementation Environment

5.2 Results

Table 5.2 shows the execution time for the detection of iris and pupil boundaries in the random eye images taken from the standard CASIA database. The average speedup obtained on the GPU is 9.228x for sample from the CASIA database [28]. Even though the GPU used has 192 processor cores, the Speedup obtained is not 192 due to a lot of factors like amount of sequential code, idling and communication latency as addressed in Chapter 1. This is a significant improvement, provided the fact that the speedup realized will be much higher on the actual dedicated systems for the purpose of iris recognition, since the implementation environment is affected by various factors like hundreds of processes executing on the processor at any given time, and many other factors. This comparison is shown in Figure 5.1

Sample No.	$\mathbf{T_{serial}}$ **(sec)**	$\mathbf{T_{parallel}}$ **(sec)**	**Speedup**
1	27.648	2.917	9.47
2	28.603	3.026	9.45
3	21.560	2.590	8.32
4	22.698	2.793	8.13
5	25.802	2.792	9.24
6	29.172	3.010	9.69
7	27.893	2.887	9.66
8	29.766	3.011	9.88
9	27.831	3.151	8.83
10	29.090	3.026	9.61

Table 5.2: Execution times of serial and parallel algorithm on sample images of size 320 x 280 taken from CASIA database

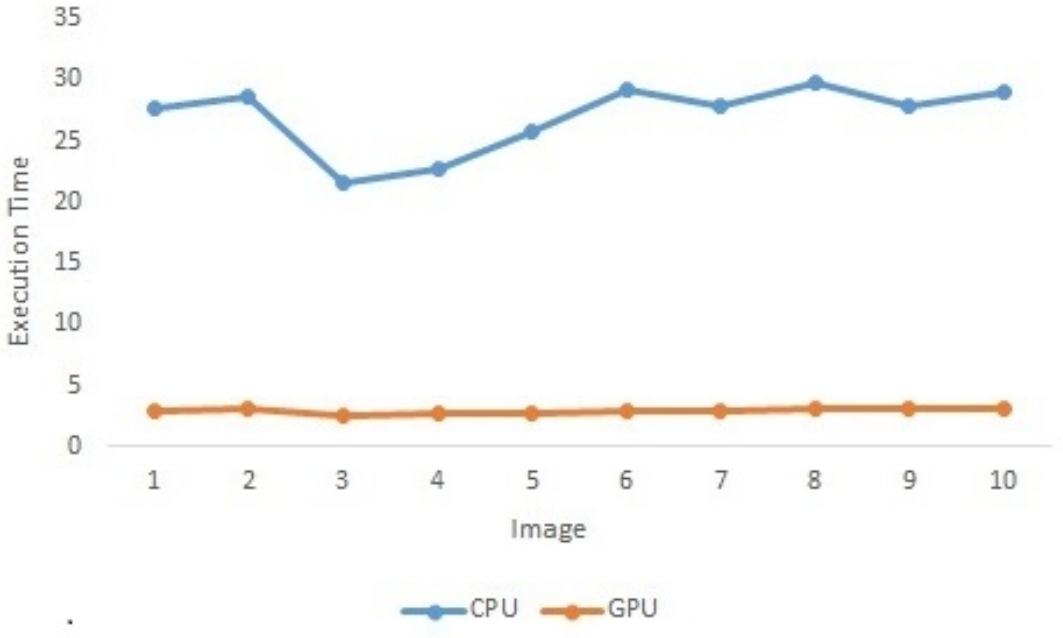

Figure 5.1: Execution time of serial algorithm on CPU and parallel algorithm on GPU

Chapter 6

Conclusion

In this thesis, a critical part of Iris recognition, i.e. Iris segmentation is parallelized on Graphics Processing Unit(GPU), using NVIDIA's CUDA architecture. CUDA provides a platform to express a complex parallel algorithm in a standard high level language like C. The expressiveness and flexibility of CUDA eases the parallel implementation of Iris segmentation, that can be incorporated in parallel iris recognition systems.

If all the steps involved in Iris recognition are accelerated using GPUs, the speedup will be greatly increased and thus the overall performance of the system will improve. With the recent projects such as UIDAI which has images of millions of people enrolled, the benefit of parallel implementations increase, while searching for a record in the database. GPUs have overturned the way computing is done, and this revolution will continue with development of more and more parallel versions of existing sequential algorithms being created for execution on the GPUs.

Bibliography

[1] Pathak.A. *Recognition using SIFT and its Variants on Improved Segmented Iris*, Bachelor's Thesis, NIT Rourkela, 2013.

[2] Grama.A., Gupta.A., Karypis.G., and Vipin. *Introduction to Parallel Computing*. Pearson Education, Second Edition, 2007.

[3] Sinha.A. *GPU Accelerated Iris Localization*, Master's Thesis, NIT Rourkela, 2013.

[4] Standards Coordinating Committee 10, Terms and Denitions . The IEEE Standard Dictionary of Electrical and Electronics Terms , J. Radatz , Ed. IEEE , 1996.

[5] Flynn.M.J. *Very high - speed computing systmes* . Proceedings of the IEEE . 54 (12), pp.1901–1909 , 1966.

[6] Gebali.F. *Algorithms and Parallel Computing*. Wiley Publications.

[7] Hossain.M.A. , Kabir.U. , Tokhi.M.O. *Impact of data dependencies for real-time high performance computing*. Journal of Microprocessors and Microsystems. 26 (6), pp.253-261, 2002.

[8] Sirohey.S. , Rosenfeld.A. , and Duric.Z. *Eye tracking*. Technical Report CAR-TR-922, Center for Automation Research, University of Maryland, College Park, 1999.

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

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

[11] Daugman.J. *High confidence visual recognition of persons by a test of statistical independence*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 15, pp.1148–1161, 1993.

[12] Daugman.J. *How Iris Recognition Works*. IEEE Transactions on Circuits and Systems for Video Technology. 14(1), pp.21–30, 2004.

[13] Flom.L. and Safir.A. *Iris Recognition Systems*. US Patent 4641349, 1987.

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

[15] Wildes.R. , Asmuth.J. , Green.G. , Hsu.S. , Kolczynski.R. , Matey.J. , McBride.S. *A Machine-vision System for Iris Recognition*. Machine Vision and Applications. 9, pp.1–8.

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

[17] Sun.Z.A. , Tan.T.N. , and Wang.Y.H. *Robust encoding of local ordinal measures: A general framework of iris recognition*. In ECCV Workshop on Biometric Authentication. pp.270-282, 2004.

[18] Ma.L. , Tan.T. , Wang.Y. , and Zhang.D. *Efficient iris recognition by characterizing key local variations*. IEEE Transactions on Image Processing. 13(6), pp.739-750, 2004.

[19] Yao.P. , Li.J. , Ye.X. , Zhuang.Z. , and Li.B. *Iris recognition algorithm using modified log-gabor filters*. In Proceedings of the 18th International Conference on Pattern Recognition. pp.461-464, 2006.

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

[21] Duda.R.O., Hart.P.E. *Use of the Hough Transformation to Detect Lines and Curves in Pictures*. Graphics and Image processing, Communications of the ACM. 15(1), January 1972.

[22] Hough.P. *Method and Means for Recognizing Complex Patterns*. US Patent No. 3069654, 1962

[23] Illingworth.J. and Kittler.J. *A survey of the Hough transform*. Computer Vision, Graphics and Image Processing. 44, pp.87–116, 1988.

[24] Ritter.N. *Location of the pupil-iris border in slit-lamp images of the cornea*. International Conference on Image Analysis and Processing. 1999.

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

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

[27] Camus.T.A. and Wildes.R. *Reliable and fast eye finding in close-up images*. 16th International Conference on Pattern Recognition. 1, pp.389-394, 2002.

[28] Bonney.B. , Ives.R. , Etter.D. , and Yingzi.D. *Iris pattern extraction using bit planes and standard deviations*. Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers. 1, 2004.

[29] Mehrotra.H., Bakshi.S., and Majhi.B. Real-time iris segmentation based on image morphology. Proceedings of the 2011 International Conference on Communication, Computing & Security, ICCCS 11. pp.335-338, New York, NY, USA, 2011. ACM.

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

[31] Kong.W. and Zhang.D. *Accurate iris segmentation based on novel reaction and eyelash detection model*. International Symposium on Intelligent Multimedia, Video and Speech Processin., pp.263–266, May 2001.

[32] and Steiner.N *Parallelizing Iris Recognition*. IEEE transactions on information forensics ans secutiy. 4(4), DECEMBER 2009.

[33] Sakr.F.Z. , Taher.M. , Wahba.A.M. *High Performance Iris Recognition System On GPU*. IEEE International Conference on Computer Engineering & Systems (ICCES). pp. 237 242, 2011.

[34] Sakr.F.Z. , Taher.M. , Wahba.A.M. *Accelerating Iris Recognition algorithms on GPUs*, IEEE CIBEC, Cairo, Egypt, 2012.

[35] Chinese Academy of Sciences Institute of Automation. Database of 756 Greyscale Eye Images. www.sinobiometrics.com

[36] Rakvic.R.N. , Ulis.B.J. , Broussard.R. and Ives.R.W. *Iris template generation with parallel logic*. 42nd Annual Asilomar Conference on Signals, Systems and Computers. Pacific Grove, CA, Nov. 2008.

[37] Lionetti.F *GPU Accelerated Cardiac Electrophysiology*. Master's Thesis. UNIVERSITY OF CALIFORNIA, SAN DIEGO

[38] Owens.J.D., Luebke.D. , Govindaraju.N. , Harris.M. , Kruger.J. , Lefohn.A.E. , and Purcell.T.J. *A survey of general-purpose computation on graphics hardware*. Computer Graphics Forum. 26, pp.80–113, 2007.

[39] Demmel.J. , Dongarra.J. , Eijkhout.V. , Fuentes.E. , Petitet.A. , Vuduc.R. , Whaley.R.C. , and Yelick.K. *Self-adapting linear algebra algorithms and software*. Proceedings of the IEEE. 93(2), pp.293–312, 2005.

[40] NVIDIA Corporation, CUDA C Programming Guide, www.developer.nvidia.com/cuda, Version 3.1, 2010.

[41] nVidia. Optimizing cuda. http://www.sdsc.edu/us/training/assets/docs/NVIDIA-04-OptimizingCUDA.pdf, 2009

[42] Silberstein.M. , Schuster.A. , Geiger.D. , Patney.A. , and Owens.J.D. *Efficient computation of sum-products on gpus through software-managed cache*. 22nd annual international conference on Supercomputing. pp.309–318. ACM New York, NY, USA, 2008

[43] http://cuda.ce.rit.edu/cuda_overview/cuda_overview.htm

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

[45] Canny.J. *A Computational Approach to Edge Detection*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 8, pp.679-698, 1986