Iris Localization using Parallel Computing

Mohammad Aknan

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

Iris Localization using Parallel Computing

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
Master of Technology
by
Mohammad Aknan
(Roll 212CS1086)
under the supervision of
Dr. Banshidhar Majhi

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

Dedicated to my best friend Subhadra Pal...

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

Dr.Banshidhar Majhi
Professor

May 30 , 2014

Certificate

This is to certify that the work in the thesis entitled *Iris Localization using Parallel Computing* by *Mohammad Aknan*, bearing roll number 212CS1086, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering*. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Dr. Banshidhar Majhi

Acknowledgement

This dissertation, though an individual work, has benefited in various ways from several people. Whilst it would be simple to name them all, it would not be easy to thank them enough.

The enthusiastic guidance and support of *Dr. Banshidhar Majhi* inspired me to stretch beyond my limits. His profound insight has guided my thinking to improve the final product. My solemnest gratefulness to him.
It is indeed a privilege to be associated with people like *Prof. S.K.Jena*, *Prof. S. K. Rath*, *Prof. D. P. Mohapatra*, *Prof. Sujata Mohanty*, *Prof. A. K. Turuk*, *Prof. S.Chinara*, *Prof. Pankaj Sa* and *Prof. B. D. Sahoo*, *Prof Ratnakar Dash.* They have made available their support in a number of ways.

Many thanks to my comrades and fellow research colleagues. It gives me a sense of happiness to be with you all. Special thanks to *Lokendra, Rajkamal, Anshuman, Nilay, Vijay, Abhishek, Dilip, Sandeep and Ankur* whose support gave a new breath to my research.
My Special thanks to Subhadra Pal for inspiring me in my odd days and moral support when i need. Her help can never be penned with words.
Words fail me to express my gratitude to my beloved parents who sacrificed their comfort for my betterment.

Mohammad Aknan

Abstract

In this thesis, we have proposed a parallel iris localization technique by implementing canny edge detection in parallel on Graphical Processing Units(GPU) with the help of Compute Unified Device Architecture(CUDA) plateform. The output of canny edge detector which is binary image transfer from GPU/Device to CPU/Host and it is given to serial circular hough transform as input that locate the iris region from image.

In this thesis, we follow the Wilde's approach of iris recognition in which he used the edge detector, and Circular Hough Transform for detecting iris region from an eye image.We processed canny edge detection part of iris localization on GPU in parallel manner and Hough transform serially on CPU. In edge detection, we processed a number of pixels in parallel that execute on cores of GPU in block and thread manner, that reduces the execution time.

The outcome of canny edge detector given to serial hough transform that locate the iris region from image. Then we compare the execution time of our parallel technique with existing serial one. In our case, execution time is reduced by 10 to 12 percent in comparison of serial approach. We use the 96 core NVidia GeForce GT 630 GPU for implementation.

***Keywords*:** **GPU, Grid, Block, Thread, CUDA, Canny Edge Detector, SMP, CHT**

Contents

List of Figures

List of Tables

Chapter 1

Inroduction

Nowadays, as computing becoming more and more fast, problems are also becoming more complex and people demand a better result in lesser time. Most of the problems are data intensive and require a lot of computations on large data set. Parallel Programming is an emerging computer science field that studies the opportunity of splitting data into small chucks and then processes them on multiple processors simultaneously, which provides a faster execution time.

In computational problem, initially we measure time and space complexity for program and try for making program more efficient in both ways. Now space complexity is generally not a big issue, due to availability of a lot of memory space. Nevertheless, the time constraint is continuing, working and becoming more important. So a lot of research is continuing for doing program in the efficient manner with an intense of getting the result in less time. So for getting the faster results, we use high-performance computing. However, the computing speed is constrained by serial execution of the program; that's become the main reason behind a lot of research in parallel computing from last few years on data intensive task.

The main reason for parallelization of most of the problem is getting speedup. Many different fields like Financial modeling, scientific computation, weather forecasting, biometrics, Statistics, Image processing, Medical imaging and diagnosis etc are application of parallel computing with nonlinear processing and massive data input.

Generally software written for serial computation and they run on a single computer having a single processor, a problem is divided into a discrete series of instructions and instructions are executed one by one and only one instruction may execute at a time.

Parallel computing is an area where we solve a computational problem, that run using multiple CPUs A problem is dividing into discrete parts that can be solved concurrently; each part is further broken into a series of instructions; instructions from each part execute simultaneously on different processors and an overall coordination mechanism is employed.

The basic motive behind the popular trend of parallel computing is speed up. Speedup is a measure that captures the relative benefit of solving a problem in parallel. Speedup is a basic unit of measurement that shows how much the parallel program is faster than the serial program. In a single Program, speedup depends on the number of processors used.

According to Amdahl's law, Speed up is defined as following:

$$S = T_s/T_p = \frac{1}{\left((1-f) + \frac{f}{n}\right)} \tag{1.1}$$

where,
S is speedup,
T_s is execution time when program execute on a single processor sequentially and
T_p is execution time when program execute on p processors in parallel.

The law assumes a program in which a fraction $(1 - f)$ of the execution time involves code that is inherently serial and a fraction f that involves code that is infinitely parallel with no scheduling overhead.

Ideal Speed up = p

Ideal speedup is obtained when time reduces by number of processors used means if we use p processor then execution time is also p time lesser than serial time. Speedup of $\frac{1}{p}$ on using p processor is not possible and it is ideal situation. Parallelism is affected by number of factors that are following:

1. Synchronization
2. Communication Latency
3. Data dependency
4. Load balancing

The main reason behind getting less speedup in parallel execution of a program is due to Communication between the task executed concurrently and control of parallel activities including scheduling of the task. Another factor of inefficiency is the synchronization between tasks that executed concurrently. In case of synchronized execution, it is required that all tasks execute concurrently completed before the next set of activities can proceed. Hence synchronization also leads to increase in computation time.

We can maximize the speedup by balancing load on processor and minimizing the cost of communication and other overhead.

1.1 Basics of Parallel Computing

1.1.1 Why Parallel Computing

The main reasons behind the preference of parallel processing in respect of serial processing are following:

1. **Resource Sharing** In parallel computing, we give more resources to a problem then less time required in completion and also save money.
2. **Solve Bigger Problems** Nowadays, various kind of program are either large in size or complicated that make it is difficult to solve them on a single computer due to space constraint and processing capacity.

3. **Limits of Serial Computing** physical and practical reasons show significant restrictions to simply making better and faster serial computers. Main constraints in building faster serial computers are following:
    (a) Transmission speed
    (b) Economic limitations

1.1.2 Architectural classification Scheme

There are three main architectural classification schemes:

1. **Flynn's Classification:**It is classifying on the basis of instruction sets and data sets in a computer architecture.
2. **Feng's classification:** It is classifying on the basis of serial and parallel processing.
3. **handler's classification:** It check the degree of pipelining and parallelism in a subsystem.

Flynn's classification is most popular in all classification scheme. Flynns Classification divides the computer system into following four categories on the basis of instruction streams and data streams in system.

1. **Single instruction stream single data stream:**A single sequential processor executes a single instruction set to operate on data stored in a single memory. Generally, instructions execute sequentially but they may be overlapped in the execution stages means we use pipelining in such a kind of system for getting speedup. These types of the system shown very low or no parallelism.
2. **Single instruction stream multiple data stream :**Many processors execute single machine instruction concurrently, and each processor has its own memory and its own data stream. Each processor has an associated data memory, so that each instruction is executed on a different set of data

by the different processors. Thats why its become the most important class of parallel architecture. [1]

3. **Multiple instruction stream single data stream:**A sequence of data stream is transmitted to a set of processors one by one, each of which executes a different instruction stream. The output of one processor work as a input for next processor. This kind of structure is not implemented commercially and do not exist in physical world.

4. **Multiple instruction stream multiple data stream:**A set of processors simultaneously execute different instruction streams on different data streams. Multiprocessor system belongs from this category. This organization can also be classified into two categories. One is tightly coupled systems when degree of interactions among processor is high and second is tightly coupled systems. Symmetric Multiprocessors (SMP), clusters and NUMA systems fit into this category.

1.1.3 Parallel Programming Communication Models

Inter processor communication is required for data exchange between different node of a parallel system. The speed up of a parallel program is greatly affected by inter processor communication. In some cases when inter processor communication is more than the computation then system performance is degraded even after parallelization. There are two primary kind of data exchange between parallel tasks accessing a shared memory space and exchanging messages between systems.

MessagePassing Platforms

Message passing system consists of p processors where each machine has its own exclusive address space. Every processor can be either a single processor or a sharedaddressspace multiprocessor. The interactions between processes running in different processors must be accomplished using messages; hence, the interaction is termed as messagepassing. Messagepassing paradigms support execution on each of the p processors.

SharedAddressSpace Platforms

A common memory space that is accessible to all processors who are working together for executing tasks in parallel is supported by shared Address space of a parallel platform. This shared memory space is interacted by processors to modify data objects. Memory in these platforms can be either local (exclusive to a processor) or global (common to all processors). The time taken by a processor to access any memory location in the system is equal is called the Uniform Memory Access (UMA. If the time taken to access different memory location varies, then the platform is called Non-Uniform Memory Access (NUMA).

1.2 Iris Recognition

Iris recognition is an automated method of biometric identification that uses mathematical pattern recognition techniques on image of an individuals eyes, whose complex random patterns are unique and can be seen from some distance. Iris recognition is a biological characteristic and considered as a form of biometric verification.

The iris generally has a form of circular ring surrounding the pupil of the eye with black, brown, blue, greenish or gray color that contained complex patterns that are in form of coronas, freckles, furrows, stripes, crypts and so on. The patterns of iris are visible to close inspecting.

In iris recognition, the identification is carried out by taking one or more detailed images of the eye with a sophisticated, high-resolution digital camera at infrared wavelengths, and then using a specialized computer program that also known as matching engine to compare the subject's iris pattern with images stored in a database. The matching engine can match tens of thousands of images per second with a level of precision comparable to other conventional biometric

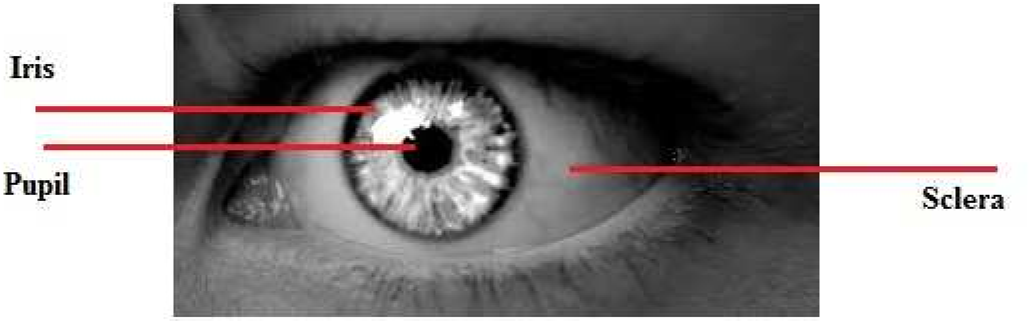

Figure 1.1: Anatomy of eye image

recognition like fingerprinting or digital finger scanning.

1.2.1 Steps in Iris Recognition

Iris Recognition system consists of several steps that shown in figure 2. First, we acquire an image of iris with the help of an infrared camera, then in next step we do some preprocessing on image and Iris Localization. After localization of the interested region from image, next step is feature extraction.

In features extraction step, we extract features from image that are different for every single individual. On the basis of extracted feature, in next step template are generated and stored in a database. For identification or verification of an individual, we compare generated template with all templates that are stored in database and verify or identify to him.

A number of biometrics are used for verification or identification of an individual, but iris is considered as a most accurate biometric in respect of other. Irisrecognition is entered very late in the group of biometric based recognition, but attracted a lot of attention from academia, government and industries.

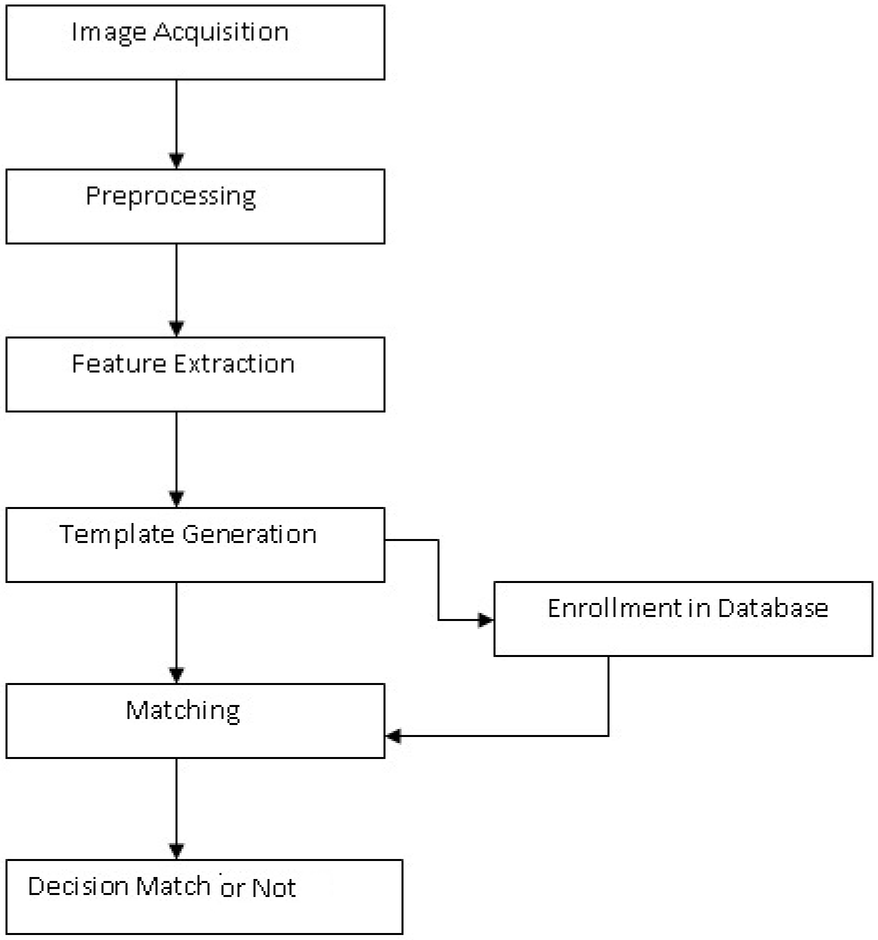

Figure 1.2: Steps in Iris Recognition

Iris patterns are unique in nature due to richness of texture details in iris images. Iris texture is stable throughout the life and after death, they deteriorate too fast so no extra attention required to check a person is alive or not. Another main quality of iris is a difficulty of live iris forgery because we required near infrared illumination to capture detailed iris texture in most cases. Iris is also well protected against damage because it is an internal organ. All these properties make iris to most suitable and secure biometric.

Many millions of people from different countries around the world are enrolled for iris recognition system, and it is used by several nations in various big projects. Iris recognition system is use in so many different areas like national border controls, computer login, secure access to a bank account at cash machines, ticket less travel, premises access control, tracing missing or wanted person, anti terrorism, biometric key cryptography and so many more.

Most of the Iris recognition systems are sequential in nature and processed on CPU. In an iris recognition system also like other biometric systems different stages (Localization, Feature extraction, template matching, etc.) are processed in sequence. However, each of the stage is computation intensive task, the efficiency of which can be improved using parallel algorithm and architecture. Low cost GPUs that have dozens to hundreds cores in respect of few core CPU are used extensively in dealing with a task of heavy computation.

1.3 Motivation

In last few years, so many algorithms developed on GPU architecture for Iris recognition with a motive of improvement in execution time. GPUs are becoming more programmers friendly day by day and also more capable in storage and processing power so it is used extensively by researchers for getting more speedup for computation intensive task that took a lot of time on serial computer. Iris recognition is a computation intensive task that fall in the SIMD category of Flynn's architecture and suitable for parallel computing. Iris localization is the first step of iris recognition and take a considerable amount of time, so for getting more speedup it is necessary to parallel iris localization.

1.4 Thesis Layout

Rest of the thesis is organized as follows:

Chapter 2 give basic information about the GPU and CUDA architecture that help in understand thesis in easy way. This chapter give basic detail about memory concept of GPU and thread, block structure.

Chapter 3 give the information about different technique of iris recognition that proposed and implemented on serial computers. It also show the parallel algorithm that recently developed for existing iris recognition technique.

Chapter 4 give the information about serial canny edge detector and proposed parallel iris localization with parallel canny edge detector.

Chapter 5 show the implementation of proposed algorithm in CUDA and also show output for serial and parallel algorithm.

Chapter 6 show conclusion.

Chapter 2

Graphical Processing Unit and CUDA Architecture

In this chapter we discuss about Basic concept of CUDA Architecture and GPU. In last two decade, Graphic cards capacity grown exponentially with a motivation of handling large amount of data in minimum time. Graphics cards are capable in handling large number of frames of a video in fast and efficient manner. Since a lot of work done in past years, that make graphic cards more powerful and they emerge with a name of Graphical Processing Unit.

GPU come with high computation capacity and visual rendering capabilities. They are capable in doing complex problem so quickly due to high speed data transfer, fast memory alteration and better computation capabilities in real time manner. These properties of Graphics Processing Units make them very popular in research and academic fields for solving complex problem efficiently and in high speed manner with the intention of getting better result for data intensive tasks. Researchers around the world start developing new algorithm with the intention of running them on GPU in parallel for existing algorithm that developed for CPU based system.

CPUs are general purpose devices with a single core to few cores but GPUs are specialized processor that available in hundreds of core that make solving problems easier in parallel. GPUs are used in different kind of electronic gadgets like Personal Computers, Mobile devices and devices that are developed for playing video games. GPUs due to such properties make a big space in market and now very popular in customer.

GPUs are able in handling at a time up to thousand threads due to more number of core or processor.

2.1 Basic of GPU hardware

GPU is consist of a large set of memory with few dozens to up to thousand cores or processor. So, the basic information about GPU hardware as following:

2.1.1 Processor

Another name for GPU is Multi Multi-Processor system due to nested structure of processor. First a number of core or processor are grouped and this group is known as a Symmetric Processor.Generally the number of core in a SMP are 8, but upto 512 core are possible. The basic unit for computation is thread that run on a core. Then the basic question that arises is what the meaning of Streaming Multiprocessor at all.

Streaming Multiprocessor are work as a barrier for threads that running on core of it. Threads that execute on a core of particular Streaming Multiprocessor cannot synchronize with threads those execute on other streaming multiprocessors core in a GPU. In basic it is looking like a disadvantage and we think at this stage that if synchronization is possible between all threads then we get more parallelism and speedup, but it is not true in real.

Main motive behind grouping of cores into SMPs for running different part of program that have no dependency at a similar time on different SMPs. This property make to GPU more parallel and scalable.

2.1.2 Thread

It is the minimum part of a program up to which we break to program for executing to it at same time on different processor.With a proper understanding of hardware and thread management, system give better result in respect of time. If we do not handle to thread in proper way on cores of a GPU, then expected speed up may not come.

Thread divergence is another big issue in parallelization of an application which includes some conditional statement. Like if a program included If -else condition then we do not run to them on different cores of a single streaming multiprocessor. This part run as serial and it is known as Thread Divergence.

2.1.3 Memory

GPU have different kind of memory with following main parameter for each kind of memory that are differentiate to them from each other.

- Memory Size
- Access time
- Scope
- Location

GPU memories are classified into following categories –

1. **Global Memory:** It is common for all SMP of GPU. All cores of GPU fetch the data from it or write on it. Also , global memory used for data transfer between GPU and Host/CPU. Global memory is available with a capacity of several GB. In size it is clearly bigger than shared memory but speed is slow in respect of shared memory.

2. **Shared memory:** It is available for each SMP of a GPU separately. Only threads that run on cores of this SMP access to it. They are better in speed but size is limited to few KBs.It is not accessible by CPU.

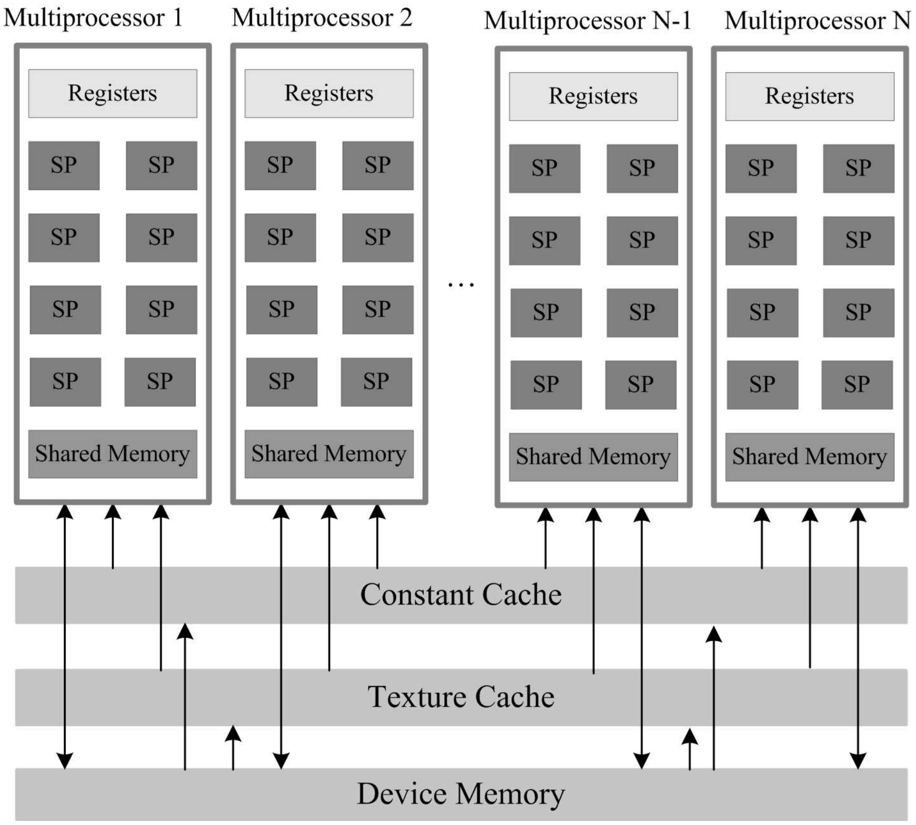

Figure 2.1: GPU Memory Architecture

3. **Register:** Each core also have its own local small size, high speed memory. Size of register are only few bytes.

4. **Constant Memory:** It is available for all threads that run on GPU at a time. It is read only and GPU code just access to it but not modified. CPU also access to it and modify data.

5. **Local Memory:** Each core have its own local memory and only thread that run on this core can access to it.

When we run a program on a GPU, then memory which we used for store/load data also give impact on the speed of program.

2.1.4 Limitation

GPU accelerate to those problems in excellence manner that are sequential in nature. But if a problem have inter-dependent and conditional code, then GPU is not able in parallelize and execute it in serial manner. Communication latency also another limitation of GPU. Sometimes we get degradation in speed if problem have more data transfer and less computation.

2.2 Basic of CUDA Architecture

CUDA or Compute Unified Device Architecture is a programming methodology developed by NVIDIA that solve problems in parallel with the help of GPUs.

Problems those have a big amount of independent data and computation intensive provide better better result when programmed in CUDA. For such kind of problems, GPUs based implementation with the help of CUDA show several times speedup in execution time in respect of serial CPU based implementation.

In general, CUDA is just an extension of C. CUDA support various computational interfaces, like DirectCompute and OPENCL. MATLAB, OPENCV, Python etc used thirdparty wrappers for running CUDA program.

2.2.1 CUDA Program Structure

Parallel program consist of serial and parallel part. Serial part is run on CPU that also known as a Host and parallel part is run on GPU/Device in parrallel manner. In CUDA, a part of program that run on GPU arranged in grid, block and thread manner. A grid have a number of blocks and a block have a number of threads. Threads are distributed between block in equal manner means each block have equal number of threads. Blocks are also grouped in similar manner and make grid. Total number of block in a GPU are calculated by dividing total number of threads by threads per block.

In GPU, threads are generally arranged in a group of 8. Maximum possible threads per block and blocks per grid are 1024. A streaming Multiprocessor does not hold more than 1024 threads due to limited size of Shared memory.

CUDA program are run on different GPUs that may be have different number of core. Thread scheduling in CUDA not done by programmer and Compute Engine of GPU automatically schedule to them on cores. If we have a program with 16 threads, it is automatically run on 4 core, 8 core or 16 core SMPs without any modification. This scheduling property of CUDA give more freedom to a programmer.

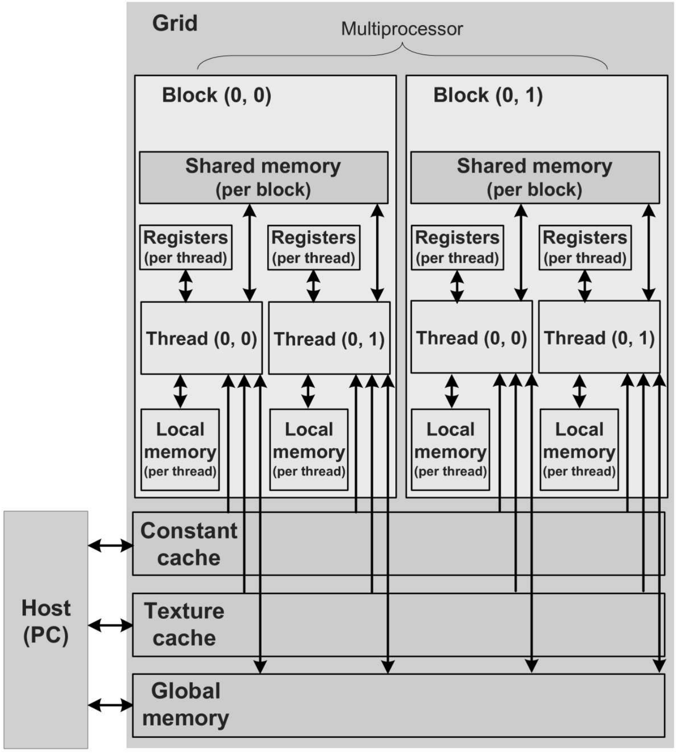

Figure 2.2: GPU in Grid, Block and Thread manner

CUDA program are generally included in a high level language to make a parallel program. If we used C, then CUDA program is same as C program with some additional code [21]. This additional code represent the kernel and when it run then define number of thread run in parallel on availbale core of GPU.If number on cores in GPU lesser than the number of threads for an application, then first number of threads equal to available cores run then later remaining cores run. In CUDA C program, a kernel is defined in following way [21]

This line is included in each CUDA C program, where global shown that it is run on device code globally, not on host. In CUDA program, with the help of following line we define the number of threads that execute in parallel.

Where X represent number of block and Y represent number of threads per block. If a GPU have X$*$Y number of core then X block each with Y number of threads execute in parallel. A block is executed on a single Streaming Multiprocessor and the threads that run on this SMP not coordinate with threads of other block. Each thread have its unique ID that represent to it om GPU. Thread ID is may be one, two or three dimensional. If it is one dimensional then it's represented by a single array and shown by threadIdx.x. Two dimensional thread id represented by threadIdx.x and threadx.y in two dimensional array and three dimensional represented by threadIdx.x, threadIdx.y and threadIdx.z [21]. In same way, CUDA arrange to block on GPU.

Chapter 3

Literature Review

The concept of automated Iris recognition system was proposed in 1987 by Leonard Flom and Aran Safir [2]. The eye of individual is first illuminated until the pupil reaches a predefined size, at which an image of the eye is captured. To account for variation in size of iris due to maximization and minimization of pupil, the illumination has been changed for make the pupil of predetermined size. Then captured image is compared with stored templates after doing some preprocessing. The stored templates were previously captured images of people that captured after the pupil reaches a same predefined size with the help of illumination. They also proposed pattern recognition tools to extract iris features and an initial method for detecting pupil using the static threshold. However, they don't make an operational Iris recognition system and then contact to Daugman.

The first operational iris recognition system has been developed at University of Cambridge by Daugman [2]. The digital image of an eye has been acquired using near-infrared light source so that illumination could be controlled for maximization or minimization of pupil, that remains unaffected to users. The next step is detection of iris in the image or iris localization. A deformable template is trained with some parameters and properties of the eye to improve the detection process [3]. Daugman suggests for getting the rich information of iris for making template; an imaging system should have at least of 70 pixels in radius of it.

After localization of iris region in captured image, next step is feature extraction in which we take the unique feature from localized iris region that is circular in shape and make a template. Daugman used multi−scale quadrature 2D Gabor wavelets for extracting texture information from iris a total of 2048 bit iris code. Many other iris recognition algorithms used different size template of iris. Then in matching step he compares, this iris code or template with stored templates of same size for identifying or verifying to an individual by applying logical X−OR operator who finds the Hamming Distance between them.

$$HammingDistance = \frac{\parallel (CodeA \oplus CodeB) \cap MaskA \cap MaskB \parallel}{\parallel MaskA \cap MaskB \parallel} \tag{3.1}$$

In the formula of hamming distance, denominator ensures that just significant bits give impact on Hamming distance. If the value of hamming distance is near to 0.5, then two templates that we are comparing are generated from different iris, but if both templates are generated from the iris of single person, then HD must be near to 0 since both are mostly correlated.

3.1 Iris Localization

Iris localization means to locate the inner and outer boundary of the iris. The image captured by the image acquisition system contains a larger portion of Image that includes data from immediately surrounding eye region [5]. Iris's image preprocessing is one of the most important steps in iris recognition system, and it also determines the accuracy of matching.

An eye is composed of three main components−sclera, iris and pupil. Sclera is white and biggest and out of the iris. Pupil is in the center of an iris, and its diameter is constantly changing, even under constant illumination. The iris, which contains texture information, is between pupil and sclera. The obtained eye image has to be preprocessed and localized to detect the iris and pupil.

The important steps in iris localization are outer boundary detection and inner boundary detection. Therefore, prior to calculating the features of iris and iris matching, it is very important to accurately segment and localize the iris from the acquired eye image because the overall performance of the iris recognition system

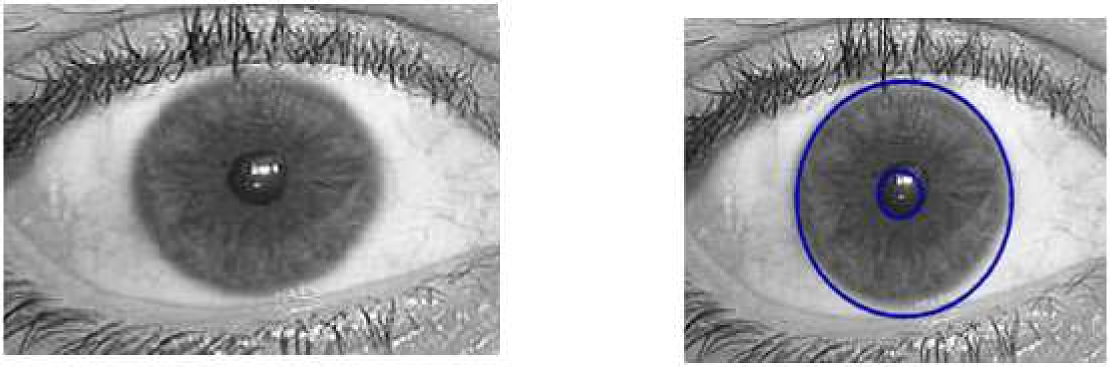

Figure 3.1: An Eye image before and after localization

is decided, firstly by the fact that how accurate iris is segmented and localized from an eye image and secondly by the resolution of an image [6].

Main methods of iris localization are following:

3.1.1 Iris localization using Integro differential operator

Daugman [2, 6–8] use Integro differential operator for detecting the circle shape iris and pupil region of captured image as well as the arcs of upper and lower eyelids. IDO is given in following equation:

$$Max(r, x_0, y_0) \mid G_\sigma(r) * (\frac{\delta}{\delta r}) \int_{(r,x_0,y_0} \frac{I(x, y)}{2\pi r} ds \mid \tag{3.2}$$

The operator is applied continuously on image with the intention of finding a maximum contour integral derivative with increasing radius at successively finer scales of analyzing through the three parameters, center coordinates and radius $(x0, y0, r)$. Eyelids are also localized in the same way. Daugmans algorithm seems like a variation of Hough transformation for circle because it used first derivative of the picture and also search for geometric variable. It does not face a threshold problem of Hough transform because it works with raw derivative information. However, sometimes it may be failed if noise in the captured eye image present due to reflection because it works on the local scale. After localization of Iris, a 2048 bit code is calculated and then compared with templates that present in the database. These patterns are encoded with the help of 2D Gabor demodulation. The IrisCode consists of 2048 bits or 256 byte of data plus 2048 masking bits producing an Iris Code of 512 bytes.

3.1.2 Iris Localization using Circular Hough Transform

Hough transform used in iris localization due to its property of isolating features of a particular shape from an image. Wildes [5] use the Hough transform for circle on image of an iris.

The equation of Hough transform for circle is:

$$H(x_c, y_c, r) = \sum_{j=1}^{n} h(x_j, y_j, x_c, y_c, r) \tag{3.3}$$

where

$$h(x_j, y_j, x_c, y_c, r) = \begin{cases} 1 & \text{if } g(x_j, y_j, x_c, y_c, r) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

with

$$g(x_j, y_j, x_c, y_c, r) = (x_j - x_c)^2 + (y_j - y_c)^2 - r^2 \tag{3.5}$$

Where x_c, y_c are the center coordinates of circle and r is the radius.

The process of identifying circle from an image using Circular Hough Transform is:

1. First we find all edges in an image by applying edge detector like Canny, Sobel or Morphological operation.
2. 2. After applying edge detector, at each edge point draw a circle with defined radius. We find Maximum intersected point by applying voting procedure and then this point become the possible center of circle.

3.1.3 Essams Approach

Essams [10] proposed three level morphological and threshold computation for quick processing of iris segmentation process with acceptable accuracy. They perform localization in two stages:

1. Coarse stage
2. Fine stage

In Coarse stage, acquired Iris image is processed with three level thresholding after converting to it in gray image. Then they apply morphological processing for detecting pupil after filling small holes and probable center of the pupil also detect. In fine refinement stage, they first reduce the image size to one-fourth. Then they applied Daugmans IDO for locating iris and pupil boundaries.

3.1.4 Other Approaches

El-Bakry [11] give a Neural Network based approach for iris localization. Bonney [12] detect the pupil in eye image by using LSB plane of image. Then they apply erosion and dilation operations for locating pupil. When they locate pupil region, then they compute the standard deviation in the horizontal and vertical direction for detecting limbic boundary. J. Cooper et.al [13] localize pupil by using active contour model.

3.2 Parallel techniques for Iris Localization

All the approaches that we discuss so far in this chapter are sequential in nature and implemented on CPU in serial manner. Main stages of Iris recognition is fall in Single Instruction Multiple Data (SIMD) category of Flynn architecture that is most suitable for parallelization [24]. However Iris recognition is computation intensive task but not so much work is done for parallelization of it and a lot of research is going on in this field.

Ryan N. rakvic et. al. [17] present a parallel implementation of iris recognition system using Field Programmable Gate Arrays (FGPAs) and provide an alternative in respect of CPU based system. A FGPA is an integrated circuit that configured by a designer after manufacturing. They deconstructed and directly parallelized the time taking steps of an iris recognition system. They parallelized the parts of iris localization, template generation and matching step and gain speed up of 9, 320 and 19 times in comparison of CPU based system.

In 2010, Nicholas A. Vandal [14] parallelized the template matching step of Iris recognition on Graphics Processing Units with CUDA programming model to achieve higher matching rates and they got $14X$ speedup in comparison of serial implementation.

. In 2011, Fatma Z. Sakr et. al. [15] parallelized the template matching and identification step of Iris recognition system using GPU. They gain speedup of around 16 and 11 times in template matching and identification step and get overall 1.3 times speedup. They parallelized the Hamming Distance approach of template matching step and also suggest that if GPU is able in holding 2048 bit Iriscode in shared memory then we achieve more speedup. Fatma Z. Sakr et. al. [16] parallelized the first two stages of Iris recognition system on GPU and gain speedup of 9.6 and 15 times in Localization and feature extraction respectively and total speedup of 12.4 times after merging it previous work [15].

In 2013, A. Sinha [15] come with the parallel iris localization and parallelize the Hough transform part of iris localization. This parallel iris localization shown, good performance even on low capacity GPU and he shown sppedup of more than 5 times that even more on high capacity GPUs.

In 2013, M Askari et. al. [23], come with two different algorithm for parallelization of hough transform part of iris localization and show 60 times more speedup for fully parallel hough transform algorithm in comparison of serial algorithm that implemented on CPU. They also used the CUDA architecture for implementation with GPU and also implement parallel algorithm on a several GPUs that have different number of core and show the impact of number of core on speedup.

3.3 Summary

Significant research has been done in field of iris recognition but most of the work is based on sequential system. As we seen iris recognition is a computation intensive task and research is continue from last few year in GPU based parallelization of such kind of task. So there is a lot of scope remain in full parallelization of different stages of iris recognition system.

Chapter 4

Parallel Iris Localization

4.1 Objective

Iris recognition is a computational intensive task that involves a lot of processing in various stages. After image acquisition, iris localization is the first step in which we locate the interested area that is used for extracting required information in later stages of iris recognition. So, in iris localization we work on every single pixel of image for tracing iris region. Most of the techniques for iris localization are sequential in nature. The main objective of our work is to parallel the existing iris localization techniques over GPU with the help of CUDA and compare with it with serial one. In Wildes approach, iris localization is consisted of two steps, edge detection in iris image and apply the circular Hough transform technique for locating iris. We parallelize the first step of iris localization technique and examine the result that we get from it.

4.1.1 Serial Canny Edge Detection and Circular Hough Transform

Iris Localization is used for locating the region of interest for further processing in Iris recognition system. We take the Wildes approach for Iris localization that is work in the serial manner. Canny edge detector is a step by step process that used for reducing the data with preserving the useful structural properties about image boundaries for further image processing [1]. The output of canny edge detection is given to Hough transform for circle, and then we get the localized image of iris. Canny edge detection algorithm consists of five steps that run separately one after one.

1. Smoothing.
2. Finding gradients.
3. Non maximum suppression.
4. Double thresholding.
5. Edge detection by hysteresis.

In Smoothing, noise is removed from acquired image because it may represent edge mistakenly. For removing the noise from an image, first image smoothed by Gaussian filter. Main work of a canny edge detector is to detect edges in an image where intensity change. In the image, the area where intensity changes sharply are detected by finding gradients of the image for each pixel. Gradients find out by applying Sobel operator. In sobel operator, first find the gradient in X and Y direction respectively by applying following kernels that shown in equation for horizontal and vertical direction.

$$G_x = \begin{vmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{vmatrix}$$

$$G_y = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}$$

Then next step in sobel operator to find out gradient magnitude by adding square of both direction kernel and then take square-root of it. In next step, by applying non maximum suppression just convert the blurred edged, that come after applying sobel operator are changed into sharp edges. It is come after retaining just local maxima and removing everything else from gradient image.

After applying non maximum suppression, next step in a canny edge detector is double thresholding. In double thresholding, two thresholds limit as low and high used. If pixel intensity is lower than the low threshold, then we consider to such a pixel as black pixel and set the intensity of them to 0. If pixel intensity is lies between thresholds, then we consider to it as a weak. Those pixels, their intensity is higher than the upper threshold then they come in a final image as a white pixel and consider as a strong pixel. Final edge detection is come by applying hysteresis on the image that came after double thresholding step. In final image, Weak pixels come in the final edge image only if they are connected to edges that consist by strong pixels.

The image that we gain after applying a canny edge detector is a binary image where all edge pixels are shown in white color and remaining in black. This binary image, give as an input image to Hough transformation for circle. Hough transformation is used for detecting edges, which are in linear or circular shape from an image. In eye image, shape of both iris and pupil are circular. So for locating iris, Hough transformation for circle[chapter 3] is applied on image which we get after applying a canny edge detector.

Hough transformation for circle applied on image from a selected minimum radius to a maximum radius for each pixel. Then for each radius size, we create an accumulator array that equal to a total number of this size radius. For each point, we detect that if the circle boundary represent an edge point, then we increment to accumulator location by 1. After applying same radius circle on all points, we find out the maximum value location from accumulator and put into another array of maxvalue which size is equal to a difference of max radius and min radius. We also store the radius value, x and y coordinates into three other array that also have same size as maxvalue. Then we increment the accumulator array by 1 and also increase the radius size by one and same procedure apply, till we reach the maximum radius[2].

After completion of following procedure up to a maximum radius, we find out the maximum value from maxvalue array and also retrieve the radius value and respective coordinates from other three arrays and these point show the possible circle in iris image.

4.1.2 Parallel Approach

In parallel approach, we parallelize the edge detector part and implement to circular Hough transform part in serial. We create a grid, in which numbers of blocks are equal to height of image, and in each block number of threads are equal to image width.

First we load the image data from Host/CPU memory to device/GPU global memory. Then a gaussian filter kernel fetched the data from global memory, apply Gaussian filter on each pixel in parallel and stored back output value in global memory of GPU. Then Synchronize to threads. Then sobel template applying on the output and result stored on different position in global memory and free the memory where output of gaussian filter stored. Synchronize to all threads. Then calculate gradient value and direction and stored back on global memory and free the memory where sobel template output stored. Synchronize threads. Then apply double thresholding by applying low threshold and high threshold value than given in start and divide to pixel based on intensity in three categories as discussed earlier in this chapter. Now detect edges in the image by suppression and load the result back to host memory. After that, Circular Hough transforms part apply on image in serial on CPU. In parallel approach, we synchronize to threads after each step of canny edge detection because before in next stage, we also require neighbor pixel for computation.

Algorithm 1

```
Serial Iris Localization (Image)

1. Load image and stored it as Img
2. Apply canny edge detection function on image.
3. Create an array of image size and show 1 where edge point in output image
   of step 2 otherwise put 0.
4. Define Iris Max and min radius as Rmx and Rmn
5. Make arrays Mxval, Xcoordinates, Ycoordinates and Radiusval of size
   $R_(mx) - R_(mn)$
6. n=72
7. j=0
8. for r= $R_(mn)$ to$R_(mx)$ do
9. $m = (height - r) * (width - r)$
10. Make an accumulator array Hspacej of size n and set at 0
11.       for Circle $C_i$= $C_1$ to Cm each have radius r do
12.           for Pixel $P_k$= $P_1$ to $P_n$ on $C_i$
13.             If point Pk is an edge point which is on center of circle of radius r
14.               Increase Hspacej at edge point
15.           end
16.        end
17. end
```

```
18. Detect max(Hspacej) and add in Mxval[j] and update Xcoordinates, Ycoordinates and Radiusval array also at same location.
19. J++
20. End
21. Max(mxval) show a circle and respective arrays Xcoordinates, Ycoordinates and radiusval array give the center coordinate and radius value
22. Draw circle
```

Algorithm 2

Parallel Iris localization (image)

```
1. Load image and stored as img
2. Set grid size= image height.
3. Set block size=image width.
4. Call Parallel Kernel(Image, lowthrsld, uprthrshld) for each thread.
5. Do step 3 to 22 of serial iris localization
```

Algorithm 3

Kernel(image,high threshold, Low threshold)

```
1. Load pixel from host to device global memory
2. Apply 3 * 3 Gaussian fliter and stored result in global memory
3. Synchronize threads
4. Apply horizontal and vertical sobel filter on output of step 2 and store result on different location in global memory
```

5. Synchronize threads
6. Free memory of step 2
7. Find out gradient from output of step 3 and store to it at different location
8. Synchronize threads
9. Free memory of step 4
10. Apply double threshold on step 7 output and store back on same location and categorized pixels into nil, weak or strong pixel
11. Remove the weak pixel that not connected to strong pixel of step 10 and set to them as a nil pixel(black pixel).
12. Transfer output to host memory

4.1.3 Time Complexity Analysis

We analyze the time complexity for an image . Let for smoothing and gradient checking, kernel size is S and G pixel respectively.

First, we apply smoothing and gradient filter on each pixel of image. We also detect gradient magnitude and direction on each pixel of image. After that double thresholding step compares each pixel of image with two thresholds. Then in hysteresis step of canny edge detector, only weak pixels compared with neighbor pixel.

For Hough transform part, time complexity in serial algorithm is equal to multiplication of image size, diagonal of image and number of point that we calculate for each circle.

So, the overall time complexity of serial iris localization algorithm is,

$$T_s = O(S * imagesize + 2G * imagesize + 2 * imagesize + imagesize + imagesize + imagesize * diagonalofimage * numberofpointthatwecalculateforeachcircle)$$

Where,

First and second part shows time complexity of applying Gaussian and sobel kernel on image, third part shows time complexity of gradient and gradient direction computation. Fourth and fifth part show time complexity of double thresholding and hysteresis computation and last part show the time complexity of circular Hough transform.

Time complexity for parallel algorithm is,

T_p=$O(S + 2G + 2 + 1 + 1+$ image size$*$ diagonal of image $*$ number of point that we calculate for each circle)

$= O(S + 2G+$ image size$*$ diagonal of image $*$ number of point that we calculate for each cicle)

In calculation of time complexity for parallel algorithm, we assume that GPU has an equal or more number of core than image size. If cores are less than image size, then time complexity is more and in multiple of Tparallel.

Chapter 5

Implementation and Result

The parallel algorithm which implemented on GPUs compared with implementation of CPU based serial algorithm for Iris localization. Serial algorithm is just an existing algorithm that we implement on CPU.

Then, after parallelization of canny edge detection part of a serial iris localization algorithm implemented to it GPU and compare the performance of both. Iris recognition is a computation intensive task with most of the computation at a pixel level. So, we implement to it on GPUs and reduce the execution time in comparison of an existing serial algorithm.

5.1 Result

For implemention of our serial algorithm, we used Intel Core i3 CPU, which has 3 GB system memory. Parallel algorithm is implemented on same system's 96 core NVIDIA GeForce 630 GPU,which has 2 GB device memory and compared with serial one. The time of execution for both parallel and serial algorithm with speedup shown in figure 1 for iris images of $320 * 280$ size [22].

In table, we have seen execution time of a parallel algorithm is ten percent less from serial one. We have seen, most of the time consuming step are the Hough

Image no	T_s (msec)	T_p (msec)	Speedup
1	6308	5715	1.10
2	6758	5985	1.12
3	6369	5692	1.11
4	6532	5763	1.13
5	6576	5832	1.12
6	6813	6168	1.10
7	6402	5637	1.13
8	6497	5746	1.13

Table 5.1: Comparison of Serial and Parallel Iris Localization with a image size of $320 * 280$ pixel

transform part that we execute in the serial manner but parallel edge detector take around 600 milliseconds less time than the serial edge detector. In 2013, A. sinha [20] gain speedup of around 5.5 times by implementing Hough transform in parallel. Our Parallel iris localization approach reduced the time, and if we implement to both canny edge detection and circular hough transform in single parallel algorithm, then it will give much more speed up. .

Chapter 6

Conclusion

Iris recognition is emerging as a best biometric for identification and verification purposes. In iris recognition, hundreds of millions samples are compared for individual verification in big projects like UIDAI, so time is a critical factor. In our approach, we implement edge detection part of iris localization in parallel on GPU using CUDA.

In GPU, computational cores are continuously increased, so in near future when the core in GPU in a number equals image size, then it gave the tremendous results. Also, when in future shared memory and SMPs internal memory size increase then performance of edge detection also improves because we will be able in doing most of the operation at local memory of SMP and not access to GPU global memory each time that have more communication latencies in respect of local memory.

In Iris localization step, edge detection part has less computational complexity in respect of Hough transform part, but it also shows good improvement. In our approach, we execute Hough transform in the serial manner on CPU. But when we consider edge detection and Hough transform part in combine for parallel implementation, then we get a result that will take so lessor time than present parallel implementation of iris localization. This parallel implementation of edge detection part of iris localization will help in making fully parallelized Iris recognition system.

Bibliography

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

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

[3] J. Daugman. Biometric personal identification system based on iris analysis. U.S. Patent No. 29160, 1994.

[4] 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.

[5] Y. Huang, S. Luo, and E. Chen. An efficient iris recognition system. In International Conference on Machine Learning and Cybernetics, volume 1, pages 40?44, 2002

[6] R.P. Wildes. Iris recognition: an emerging biometric technology. Proceedings of the IEEE, 8(9):1348?1363, 1997

[7] Rajesh Bodade and Sanjay Talbar, Novel approach of accurate Iris localization form high resolution eye images suitable for fake iris detection, International Journal of Information Technology and Knowledge Management July-December 2010, Volume 3, No. 2, pp. 68-690.

[8] J. G. Daugman: High confidence visual recognition of persons by a test of statistical independence. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 1, pages 1148 - 1161, 1993.

[9] J. G. Daugman. How Iris Recognition Works. IEEE Transactions on Circuits and Systems for Video Technology, Vol 14(1), pp. 21 - 30, 2004.

[10] J. G. Daugman. The importance of being random : statistical principles of iris recognition. Pattern Recognition, Vol 36(2), pp. 279 - 291, 2003.

[11] Essam M. An Efficient Iris Localization Algorithm. 29th National Radio Science Conference, Cairo University, Egypt, pages 285-292, 2012.

[12] Hazem M. El-Bakry. Fast iris detection for personal identification using modular neural networks. IEEE ISCS, pages 52-55, 2001.

[13] B Bonney, R Ives, D Etter, and Y Du. Iris pattern extraction using bit planes and standard deviations. 38th Asilomar Conference on Signals, Systems, and Computers, volume 1, pages 582? 586, Nov 2004.

[14] J. Cooper, Location of the pupil-iris border in slit-lamp images of the cornea. ICIAP, 1999.

[15] N. A. Vandal and M. Savvides, CUDA accelerated iris template matching on graphics processing units (GPUs), in Proceedings of Fourth IEEE International Conference on Biometrics: Theory Applications and Systems, pp. 1-7, 2010.

[16] Fatma Z. Sakr, M. Taher, A.M. Wahba. High Performance Iris Recognition System On GPU, IEEE ICCES, pp. 237- 242, 2011

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

[18] Ryan N. Rakvic, B. J. Ulis, R. P. Broussard, R. W. Ives and N. Steiner. Parallelizing Iris Recognition. IEEE transactions on IFS, Vol. 4, No 4, 2009

[19] J. Canny, A computational approach to edge detection, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. PAMI-8, NO. 6, NOVEMBER 1986.

[20] A. Sinha, GPU accelerated iris localization, www.nitrkl.ac.in, 2013

[21] NVIDIA Corporation, CUDA C Programming Guide, www.developer.nvidia.com/cuda, Version 4.3, 2013.

[22] Biometric Ideal Test, CASIA, www.http://biometrics.idealtest.org/aboutUs.jsp

[23] Meisam Askari, Hossein Ebrahimpour, Azam Asilian Bidgoli and Farahnaz Hosseini, Parallel GPU Implementation of Hough Transform for Circles, International Journal of Computer Science Issues, Vol. 10, Issue 6, No 2, November 2013

[24] Frank Willimore, Introduction to Parallel computing, Texas Advance computing center, 2012