Parallel GPU Implementation of Hough Transform for Circles

Meisam Askari[1], Hossein Ebrahimpour[2], Azam Asilian Bidgoli[3] and Farahnaz Hosseini[4]

**[1] Department of Computer Engineering, University of Kashan
Kashan, Iran**

**[2]Department of Computer Engineering, University of Kashan
Kashan, Iran**

**[3] Faculty of Electronic and Computer Engineering, Pooyesh Higher Education Institute
Qom, Iran**

**[4] Department of Computer Engineering, University of Kashan
Kashan, Iran**

Abstract
Hough transform is one of the most widely used algorithms in image processing. The major problems of Hough's transform are its time consuming and its abundant requirement of computational resources. In this paper, we try to solve this problem by paralleling this algorithm and implementing it on GPUs(Graphic Process unit) using CUDA(Compute Unified Device Architecture) . We have introduced two methods for parallelization, each of which has been implemented on four different graphic cards using CUDA. After executing the proposed methods on GPUs, we have compared our results with sequential algorithm execution on CPU and it is observable that we have about 65 times more speedup toward the sequential algorithm.
Keywords: *CUDA, Hough Transform, Image Processing, Parallel algorithm, GPU.*

1. Introduction

Hough's transform was presented by Pole Hough in 1962 [1]. This transform is a technique for determining the position of shapes at images. The main advantage of this conversion is that it can give the same results of template matching algorithms but very faster. It is reachable by changes to the formula of template matching based on Evidence Gathering Approach (where the evidences are counters in an accumulator array). The implementation of HT(Hough Transform) is a mapping from the image points into an accumulator space (Hough space). This technique especially has been used to extract lines, circles and ellipses. The present article discusses only extracting the circles.
Although HT needs fewer computational resources than template matching approach, it still requires significant storage and has high computational requirements. Therefore, a lot of efforts are being made to parallel HT. In [2] Hough transform was implemented using DCOM model, but its speedup is not still high and it has improved only twice the speedup of than sequential algorithm.
Mr. Chan has used a new algorithm for circular Hough Transform [3]. In that case, he has used two two-dimensional Hough space instead of using a three-dimensional Hough space and he has implemented this algorithm on a multiprocessor MIMD which has eight T800Transputer processors. Though, in this method, the circle detection rate comes down to less than 3 seconds, due to obsolescence Transputer processors, and because this algorithm uses gradient operator for determining circle's center and since this operator is very sensitive to noise, it is not used in practical applications. Mr. Rakvic has used FPGAs to determine circles in iris images [4]. One of the problems of FPGAs is they should be programmed with such hardware languages as VHDL, making programming very complicated. In addition, the number of ALUs used in FPGAs is a lot fewer than the number of GPU's cores, so its speedup is much less than GPU's speedup. In [5-7] other parallel platforms are used to speed up the Hough Transform but because of using special hardware, they are so costly and also their speedup toward the sequential mode isn't very high.
In [8] Wu has presented two implementation of Hough transform on CUDA and TBB . He has reached 45 times speedup more than sequential algorithm, by using CUDA on a GPU with 196 cores. In this research we have presented two new high performance parallel algorithms that improve speedup until 65 times with use a GPU with 96 cores. Using CUDA in parallel processing is so much useful because besides the availability of its hardware and its low cost; its speedup is very higher than sequential mode.

1.1. Finding the circle by HT

The Eq.(1) defines a circle in explicit form.

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \quad (1)$$

This equation represents the locus of points whose distance from an assumed point (x0, y0), namely the circle center, is a constant measure. This equation can be interpreted in two ways: locus of points (x0, y0) centered (x, y) with radius r. Fig 1 describes these two different interpretations. Each point on circle's edge in the left figure has been defined as

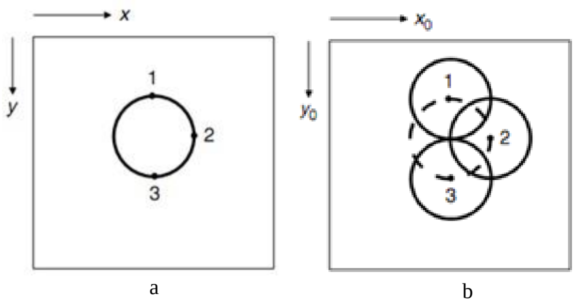

Fig. 1 (a) Image containing a circle, (b) Accumulator space for especial radius.

a circle in the accumulator space, a circle with the same center and all possible radiuses. In the right representation, circles have plotted for only an assumed radius [9]. Since circles with different radiuses should be drawn for each edge point, accumulator space must be three-dimensional. Therefore, each edge point is mapped to a cone of storage space. This mapping is shown in Fig 2 [10].

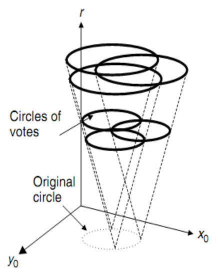

Fig. 2 Accumulator space in three dimensions.

After drawing all coins for all edge points, the point by most votes in accumulator space represents the circle's parameters in the main image. For the first time Kimme and Ballard showed how Hough transform can be used to detect circles and they used it in medical image processing [11]. One of the most widely usages of Hough Transform is in iris recognition. To identify iris, first the region containing the iris must be extracted from images. As seen in Fig 3, this area is located between two circles and using HT we can identify circles and consequently the area of the iris [12]. In addition to the above cases, Hough transform has many applications that Mr. Kittler has introduced them at [13].

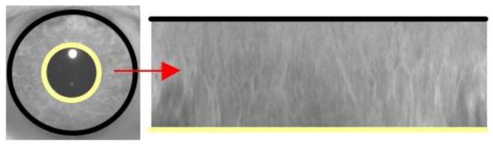

Fig. 3 Extract the region containing the iris.

1.2. Using CUDA in parallel processing

The future of computing science is parallelism, because on the one side increasing the number of transistors inside the CPU and then CPU speedup have been very hard, while the need to real-time and three-dimensional graphics capabilities, is, on the other hand, increasing daily. Using multi-core processors is an attempt to parallelism [14]. However, these processors are very expensive and their maximum increased efficiency is equal to the number of their cores. GPUs that have recently been receiving attention in most applications are useful tools for implementing parallel algorithms. The advantage of GPUs is their high performance and their availability. In Fig 4, below, a comparison between different models of GPUs and CPUs for floating–point operations has been presented [15].

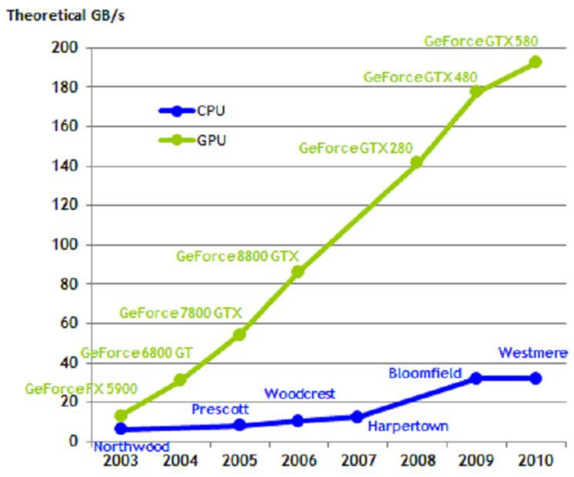

Fig. 4 Floating point operations per second for the CPU and GPU.

Each GPU includes a large number of cores, parallel cooperation of which enables GPU to perform an array of operations much faster than CPU.

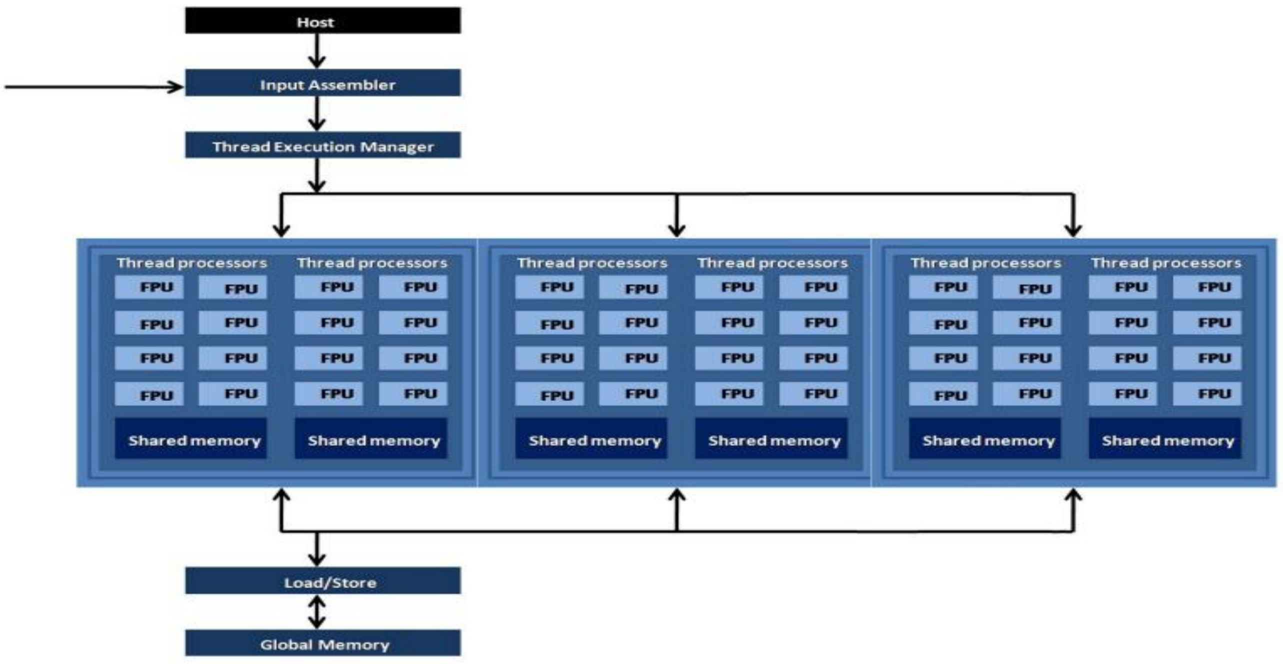

Fig. 5 NVIDIA GeForce 8800 architecture with 48 cores.

Fig 5 shows an internal architecture of a GPU with 48 cores [16]. As seen in this figure, GPU has its own memory and there is no shared memory between GPU and CPU.
Thus, at the program's launch, data is transferred from RAM to GPU memory and finally, results are transferred from GPU memory to RAM [15,17].
In 2006, NVIDIA Company introduced CUDA platform for implementing massive parallel computing with high performance on its GPUs. A software environment was presented along with CUDA that allows developers to write the programs to C language and run them on GPUs.
Each CUDA program is composed of two parts, Host and Device. Host is a program that is executed sequentially on the CPU and Device is a program running parallel on the cores of GPU.
In terms of software, each parallel program can include a number of threads. Threads are light weight processes that do an independent operation each. A number of threads make a Block and a number of Blocks create a grid. There are different types of memory in GPUs. Each thread has its own local memory. Each Block has a shared memory to which its threads have access. There is a global memory that is accessible to all threads. Moreover, there is another type of memory called texture memory that like global memory is accessible to all threads, but its addressing mode is different and it is used only for specific data like images.
In the Host part the number of threads, or in other words the number of lightweight processes that will run on GPU cores, should be determined. The code of Device runs for the number of defined threads in Host. Each thread can find its own situation by initial functions in CUDA, for example when we bind pixels of images to threads; each thread can know which pixel it is bound to and operates accordingly. Finally, the calculated results should be returned to main memory.
GPUs are suitable tools for implementing image processing algorithms. Because many of image operators are local, by allocating each pixel to a thread (of course, if required threads can be defined) the calculation time can be reduced to O(1). Olmedo and his coworkers implemented the number of typical image operators by CUDA in [18]. Similarly, in our previous work CUDA is used for space image processing [19] or Gray CUDA has been used for motion tracking [20].
In the following section, the program of finding circles by Hough transform has been implemented in two different methods on GPUs and finally the results obtained from each case are compared with sequential method.

2. Semi-parallel Hough transform

In this section, we have tried to make the Hough's transform parallel using MIMD architecture. For this purpose, we define a Block for each edge pixel in the main image. Each of these Blocks has R threads that R value is obtained from the Eq. (2).

$$R = r_{\max} - r_{\min} \tag{2}$$

In this equation, $r_{\max}$ & $r_{\min}$ are the largest and smallest radiuses for searching in the image, respectively. In fact, the three-dimensional Hough space is divided to R two-dimensional spaces that each thread has the duty of updating one of these R spaces for a particular edge pixel. Fig 6 shows that how the Blocks and threads are defined in this method. P is the Number of edge pixels in this figure.
Therefore, R threads update all of the R two-dimensional spaces for every edge pixel or put differently, for each Block, simultaneously. The pseudo code of each thread is given in Table 1.
In this pseudo code, result is a three-dimensional matrix

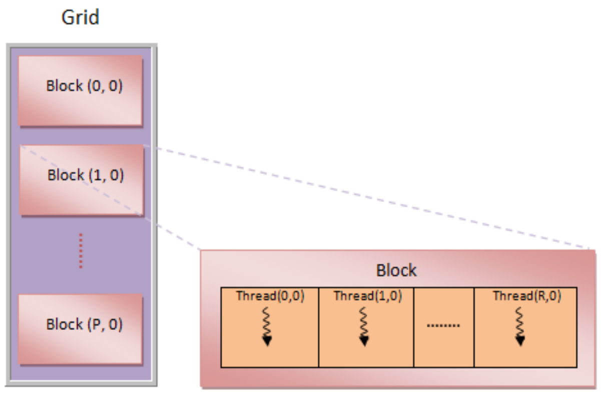

Fig. 6 Grid's structure in SPHD(Semi Parallel Hough Transform).

that makes up our Hough's space. After running this program, the Hough space is built for all pixels, and then we only need to find the maximum in this space to find the target circle. Also for searching in accumulator to find the maximum, we can use CUDA for speedup.

Table 1. Pseudo code of threads in SPHD.

```
Begin
  $x, y$ : cordinates of the edge pixel
    // calculate the radius for search
  $r$ : the thread Id $+ r_{\min}$

  For all pixels in radius $r_{\max}$ around the $x, y$
    If Euclidean distance between this pixel and $x, y == r$
      $result(r, x_{pixel}, y_{pixel}) \leftarrow result(r, x_{pixel}, y_{pixel}) + 1$
End
```

For this purpose we defined a Block with R threads, each of which should find the maximum of one of the two-dimensional spaces in the accumulator. Pseudo code of these R threads is given in Table 2.
Max in the above table is a vector of R elements. After running the above program, we only need to find the maximum between the max elements to find the final result.
If we assume that the original image dimensions are W*H, the cost of building accumulator space is in order O(WHR) that using the presented algorithm in this section, the cost cuts to order O(P). Since P<<W*H, the cost of calculation will be reduced significantly. The results of the program by this algorithm are given in the results section.

Table 2. Pseudo code for find maximum

```
Begin
  $i$ : the thread Id
  $\max[i] \leftarrow 0$
  For all $x, y$ in $result(i, :, :)$
    If $result(i, x, y) > \max$ Then
      $\max[i] \leftarrow result(i, x, y)$
End
```

3. Fully-parallel Hough transforms

As we observed in the previous section, we only could parallel operations done to build accumulator space for one pixel, but the operations corresponding to different pixels were done sequentially because we couldn't change a value in accumulator space simultaneously.
Here, we want to parallel the whole operation. To achieve this end, we should design an algorithm in which each value in accumulator space is obtained independently from other pixels. So instead of defining R threads for each edge pixel in main image, we should define a thread for each value in accumulator space.
To do so, we need to define R Blocks so that in every Block there are threads as many as image pixels. But since each Block cannot be defined with more than 512 threads, we should use a set of Blocks to accommodate all threads related to one image.
Suppose that the original image's dimensions are H*W pixels, then as many as m*n Blocks are needed for Blocking the whole of image, with m and n as Eq. (3).

$$m = \left\lceil \frac{W}{25} \right\rceil \quad n = \left\lceil \frac{H}{20} \right\rceil \tag{3}$$

In Fig 7, a method for blocking the image is shown. We defined Block size in 20*25 for simplicity; therefore, each Block contains 500 threads in two-dimensions. Therefore nm blocks should be defined for each value of R. So we used a two-dimensional grid with R*n*m Blocks.

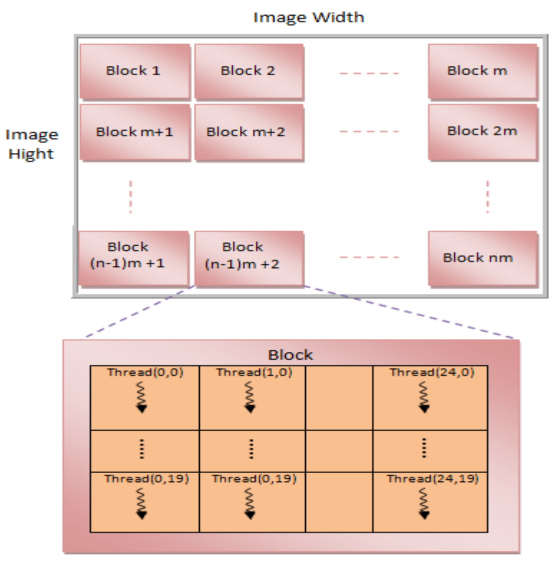

Fig. 7 Binding image's pixels to threads.

As seen in Fig 8, there are R rows, each of which contains n*m Blocks and their job is building accumulator space for

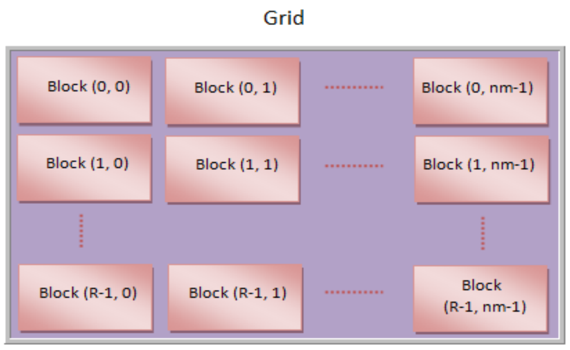

Fig. 8 Grid’s structure in FPHT (Fully Parallel Hough Transform).

a specific radius.

In this grid, each thread has the role of searching around a specific image pixel for special radius and setting its accumulator space entry with the number of corresponding edge pixels in this radius. Pseudo code of threads is brought in Table 3. In this pseudo code m and n are values that were defined in equations 3 and x, y are corresponding pixel coordinates.

In Table 3, result is a 3-dimentional matrix that is composed of the accumulator space. As can be seen, there is a thread for each cell of the result matrix, so with this grid configuration all accumulator space entries are calculated in a transaction time. Thus, the cost of filling the accumulator space is reduced to O(1). At the end, we should find the maximum according to Table 2 pseudo code to find the circle. The results of running the program on four different models of GPUs are given in the next section.

4. Results

As previously mentioned, one of the most popular applications of circular Hough’s transform is circle detection in iris images, which helps to extract the region of the iris tissue from the iris image. We have used CASIA database images [21] to test our programs. Fig 9(a) shows an iris image from this database. First, we applied the canny filter [22] on iris image for extracting the circle. Before applying the Hough transform, some pre-processing such as Gaussian filter to minimize noise, according to the method mentioned in Masek’s work [23], were done on the image. In Fig 9(b) the sample of pre-processed image is seen. Finally, two circles were identified applying Hough algorithm based on the previous methods and finding the first and second maximum, like Fig 9(c). We used four types of graphic CPU enabled card for implementing the intended methods. Their specifications together with the methods execution time are presented in Table 4.

Table 3. Pseudo code of threads in FPHT

```
Begin
  $Bx$ : Block Id at X cordinate;
  $By$ : Block Id at Y cordinate;
  $Tx$ : Thread Id at X cordinate;
  $Ty$ : Thread Id at Y cordinate;
  $x = mod(Bx, m) * 25 + Tx$;
  $y = div(Bx, n) * 20 + Ty$;
  $r = By + r_{min}$;
  For all pixels in radius r around the x, y
    If (Euclidean distance between the pixel and x, y == r) and (pixel's amount is 1)
      $result(r, x, y) \leftarrow result(r, x, y) + 1$;
End
```

Table 4. Parallel Hough transform execution time on difference graphic cards.

FPHT[1]		SPHT[2]		Sequential(ms)	GPU Cores	Model
Speed Up	Time(ms)	Speed Up	Time(ms)			
11	4808	9.5	5613	53606	8	GeForce 9300M GS
23.4	2285	9.4	5697	"	16	GeForce 9400 GT
57.6	930	8.1	6609	"	48	GeForce GT 220
65.4	819	9.8	5447	"	96	GT 430

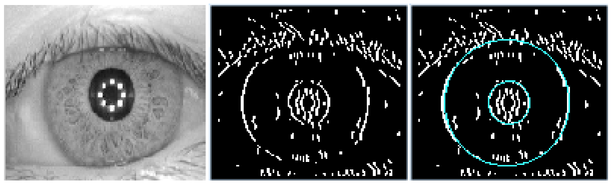

Fig. 9 (a) Iris image, (b) Edge image, (c) Identified circles by HT.

Time of the sequential mode is calculated using an Intel core i3 processor, which its cores processing at 2.8 GH speed. The chart of the speedup depending on the number of cores in GPUs is shown in Fig 10. As seen in Fig. 10, through increasing the number of cores, FPHT execution speedup increases linearly. Because, as mentioned the cost of this algorithm is of O(1), if we can define sufficient threads; cost time of this method can be equal to one transaction cost time. Since the number of threads required is much more than the number of cores in GPUs, by using a GPU with more cores we can define more threads and our speedup will increase. But from one place to the next, by increasing the GPU cores the speed does not change gradually because we cannot eliminate data transform time between CPU and GPU.
In SPHT we only need R threads and all the four graphic cards used, have the ability to define such numbers of threads. Thus, by increasing the GPU cores, only managing threads and allocating them to the cores becomes harder and the speed approximately remains unchanged.

5. Conclusions and further research

Using CUDA in image processing is rising every day. In addition to low prices and high availability, they have high performance in image processing applications. As seen in this study, Hough's transform, which is one of the time-consuming algorithms in image processing, was done in a fraction of a second. One of the problems of GPUs is their low memory; we cannot put the databases in GPU memory due to this problem. To solve it, we can use a cluster of CUDA enabled computers and share database on its nodes.

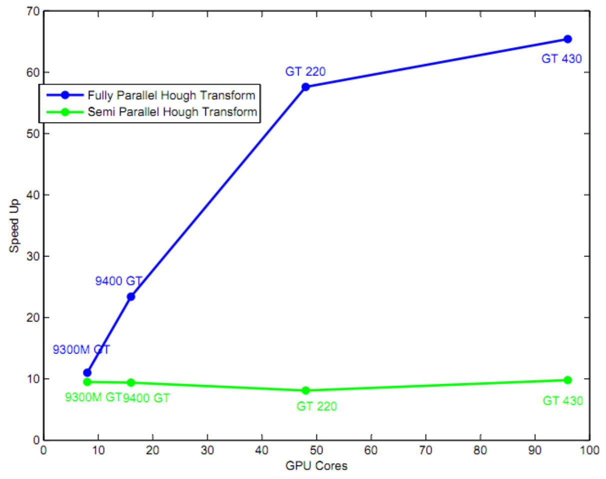

Fig. 10 Speed up depend on number of GPU cores for FPHT and SPHT

This way each node can load a part of database on its GUP's memory and we can have maximum parallelism with a low cost, which can be useful in real time applications.

References

[1] P. V. C. Hough, "METHOD AND MEANS FOR RECOGNIZING COMPLEX PATTERNS," US 3069654 United StatesFri Dec 11 15:05:56 EST 2009DTIE; NSA-17-008572English, 1962.
[2] S. KOSE, *et al.*, "PARALLEL HOUGH TRANSFORM ON DCOM ARCHITECTURE," presented at the Information Systems Analysis and Synthesis (ISAS'99),, Orlando, U.S.A, 1999.
[3] R. Chan, "New parallel Hough transform for circles," *Computers and Digital Techniques,* vol. 135, pp. 335 - 344 1991.
[4] R. N. Rakvic, *et al.*, "Parallelizing Iris Recognition " *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY,* vol. 4, DECEMBER 2009.

[1] Fully Parallel Hough Transform
[2]Semi Parallel Hough Transform

[5] Y.-K. Chen, *et al.*, "Novel parallel Hough Transform on multi-core processors " presented at the Acoustics, Speech and Signal Processing, 2008.

[6] M. Meribout, *et al.*, "Hough Transform Algorithm for Three-Dimensional Segment Extraction and its Parallel Hardware Implementation," *Computer Vision and Image Understanding*, vol. 78, 2000,pp. 177-205.

[7] C. Ming-Yang, "Design and Integration of Parallel Hough-Transform Chips for High-speed Line Detection," 2005, pp. 42-46.

[8] Suping Wu and X. Liu, "Parallelization Research of Circle Detection Based on Hough Transform," *IJCSI International Journal of Computer Science*, vol. 9, 2012,pp. 6.

[9] M. S. Nixon and A. S. Aguada, Eds., *Feature Extraction and Image Processing*. Academic Press is an imprint of Elsevier, 2008.

[10] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, 1972, pp. 11-15.

[11] C. Kimme, *et al.*, "Finding circles by an array of accumulators," *Commun. ACM*, vol. 18, 1975,pp. 120-122.

[12] C.-. Loc, *et al.*, "Person Identification Technique Using Human Iris Recognition," 2002,pp. 294-299.

[13] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer Vision, Graphics, and Image Processing*, vol. 44, 1988, pp. 87-116.

[14] J. D. Owens, *et al.*, "GPU Computing " in *Proceedings of the IEEE* 2008, pp. 879 - 899

[15] *Nvidia Cuda C Programming Guid v.4,* 2011.

[16] R. F. Anderson, *et al.*, "Applying Parallel Design Techniques to Template Matching with GPUs," 2009.

[17] "ATI Stream Computing user guide rev1.4.0a," 2009.

[18] Eric Olmedo, *et al.*, "Point to point processing of digital images using parallel computing," *IJCSI International Journal of Computer Science*, vol. 9, 2012,pp. 10.

[19] M.Askari, *et al.*, "Performance Improvement of Lucy-Richardson Algorithm using GPU " presented at the Machine Vision and Image Processing (MVIP), 2010,Esfahan.

[20] S. Grauer-Gray, *et al.*, "GPU implementation of belief propagation using CUDA for cloud tracking and reconstruction," presented at the Pattern Recognition in Remote Sensing (PRRS 2008), 2008.

[21] S. Mozaffari and H. Soltanizadeh, "ICDAR 2009 Handwritten Farsi/Arabic Character Recognition Competition," 2009, pp. 1413-1417.

[22] J. Canny, "A Computational Approach to Edge Detection," *PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 1986, vol. PAMI-8.

[23] L. Masek, "Recognition of Human Iris Patterns for Biometric Identification," The University of Western Australia, 2003.