Parallelization Research of Circle Detection Based on Hough Transform

Suping Wu[1], Xiangjiao Liu[2]

**[1]School of Mathematics and Computer Science Ningxia University
Yinchuan, 750021, China**

Abstract
There is a problem of too long computation time in Circle detection of Hough transform. In this paper, two paralleled methods are given based on Threading Building Blocks (TBB) and CUDA, by utilizing multi-core and GPU, the most time-consuming part of circle detection is coped with parallelization. Experimental results show that the circle detection algorithms proposed in this paper has extremely good result of acceleration.
Keywords: *Hough Transform, CUDA, Circle Detection, Threading Building Blocks (TBB).*

1. Introduction

Hough transform is an effective method of binary image detection on straight line, circle, ellipse and other graphics. Later, it was proposed that generalized Hough transform can detect arbitrary shape graphics. Hough transform has been applied not only to graphical boundary recognition identification, it also in biomedical, office document image processing, SAR/ the ISAR image processing and automatic interpretation of aerial images [1-2].

Traditional Hough transform can achieve very good results in the case of relatively simple image detection, but there is the drawback of long computing time and the large cost of memory space, which has been proposed many kinds of efficient solutions. Hough transform in a variety of computing models are suggested in the reference[3-15], reference[16-19] propose a variety parallel Hough transform algorithm based on special processor or programmable logic hardware . Although these methods can improve Hough transform efficiency, the higher hardware requirements make these methods inaccessible under normal circumstances. Paper [20] gives a Hough transform line detection method based on GPU, but the acceleration rate is less than 3 times. Paper [21] gives a multi-core parallelization method; the acceleration effect is not significantly obvious with paper [20].

In this paper, Hough transform circle detection parallel methods based on Threading Building Blocks (TBB) and CUDA are given and higher acceleration is achieved.

2. Sequential Algorithm of Circle Detection

Circular contour detection plays a very important role in the field of image pattern recognition [22]. In the Circle Hough Transform (CHT), the center and the radius of the circle are two basic parameters; CHT on the circle extraction is to use the accumulation of three dimensional parameter [23-24].

Under normal circumstances, the circle in the image space after Hough transform, which is mapped onto the parameter space, is three-dimensional; therefore, it is needed to establish a three-dimensional accumulation array A (a, b, r) in the parameter space for each edge point in the image to accumulate after the calculation. The algorithm steps are as follows:

1) The image edge detection and binarization.

2) To calculate the minimum and maximum values of parameters a, b, r according to the size of the image , and to establish three-dimensional discrete parameter space, the size of the parameter space is determined by the minimum and maximum values of a, b, r.

3) To create an accumulator array A (a, b, r) in the parameter space, and each element set to 0.

4)To act Hough transform on the boundary point of the image, that is to calculate the corresponding curve of the point (a, b, r) in three-dimensional grid by the formula (1), and the corresponding accumulator adds 1, namely,

$$A(a,b,r) = A(a,b,r) + 1 \qquad (1)$$

5) To find out the local maxima value of the corresponding image circle point accumulator, the value of the radius of the circumference point on the image plane, and the center of the circle parameters. This value provides the parameters of the radius and the center of the circle on the image plane.

Because of discretization of the image, and in order to reduce errors, the formula (1) can be rewritten into the formula (2) in the actual calculations.

$$\left|(a_0 - x_i)^2 + (b_0 - y_i)^2 - r^2\right| \le \xi \qquad (2)$$

Where ξ is the compensation of image quantization and digitization, r is set to be incremental variable in the calculation. Before every step iteration, r must be fixed, then to seek the point of the circumference of the center (xi, yi), these points are in the plane (a, b), which is perpendicular to r, and to accumulate points of the track and corresponding points of the three-dimensional accumulation array A (a, b, r) of the plane mapping. r progressively increases from 0 to the upper limit of the image plane, increment of r each time corresponds to a flat image, for the point(xi, yi), a and b vary between [0,2r][24].

When the Hough transform parameter space is less than two dimensional, the transformation can achieve the desirable results. However, once the parameter space is more than two-dimensional, it will need a lot of storage space and time, which makes the Hough transformation only in theory, can not be applied in practice. Only by reducing the dimension of the parameter space, can Hough transformation be achieved in practice. It is assumed that by a prior knowledge to determine the radius of the circle or the radius range [25], can effectively reduce the space and time overhead in the calculation process [24].

For the circle in the image, the center (a, b), the gradient direction of the edge point (xi, yi) is θ, and the gradient direction of each edge point points to the center of the circle. Easy to know the gradient information of the point is:

$$\begin{cases} g = \sqrt{{g_x}^2 + {g^2}_y} \\ \theta = \arctan(g_y / g_x) \end{cases}$$

Then the center of a circle formula can be expressed as the follow formula:

$$\begin{cases} a = x_i - r\cos\theta \\ b = y_i - r\sin\theta \end{cases}$$

Which $\sin\theta$, $\cos\theta$ can be directly expressed as: $\sin\theta = g_y / g$, $\cos\theta = g_x / g$.

Therefore, the local gradient information of edge pixels can reduce the computation [25], and can improve the accuracy of the center coordinates. First to detect edge of the original image , and to do gray value variation and gradient solution of the pixel, then to do gradient image binarization according to given gradient threshold. Recorded the pixel of the gradient amplitude is large than the threshold, and gradient direction of the pixel is set to 1, gradient direction of the pixel of the gradient amplitude is smaller than the threshold and is set to 0. Scan all nonzero pixels and along the gradient direction on the corresponding two-dimensional accumulation array unit to add 1, the local maximum value in the accumulation array is the center coordinates. The fact that this method makes the dimensional number of accumulation array in Hough transform circle detection algorithm is reduced to two-dimensional from the three-dimensional. And to change the traditional method of one-to-many mapping into one-to-one mapping, this method has been the default standard Hough transform circle detection method [24].

3. Parallel Circle Detection Method Based on TBB

TBB (Threading Building Blocks) is a set of C++ template
library. It provides the appropriate abstractions, and more content. In parallel programming, such contents the task concept, automatic load balancing features, mature, achievement of commonly algorithm, flexibility and scalability of no bound to the number of CPUs and so on.

TBB programming uses a template as the usual parallel iterative model. TTB is a paralleled programming model for the program scalability can be improved, and fully supports the nested. It can run on Windows, MacX and Linux, and Support Intel C++, VC7 / and gcc compiler. In order to build a large-scale paralleled program, allowing programmers to create their own paralleled components [26].

3.1 Algorithm Implementation

According to the circle detection serial algorithm, the most time-consuming in the detection process is the fourth step of the algorithm, and this process is loop iteration, we can parallel the step of the algorithm. In this loop iteration, every iteration is independent from each other, so we can make loop parallel. Parallel reduction of TBB template classes can be achieved. Because cycle accumulation in the program is reentrant, previous running results will not affect next results, so the template class paralleled for can be used.
For the loop parallelization, the first is the loop body converted into the form of a small space to operate.

The form is function object of the standard template library-style, known as the body object, an operator () deals with a little space. Note the iteration spatial parameter of the method operator(), blocked_range <T> is a template class defined in the library, which indicates the one-dimensional iteration space on the type T. iteration space of other types in class parallel for can also be used. Defined blocked_range 2d in the library is used to represent the two-dimensional iteration space. Opfor need to include a number of member variables to save the local variables defined outside the loop, but used inside the loop. Generally, these member variables are initialized in the constructor of body object, but parallel for does not need to care about the body object is how to be created. Template function parallel for requires that the body object has a constructor copy. This constructor is used to create a separate copy for each working thread, and the parallel for will call destructor to destroy those copies.

In most cases, the implicitly generated copy of the constructor and destructor can work correctly. If the design requires the destructor to perform some other operations, such as free memory, then you need to define an explicit destructor. If copies of the constructor And destructor are generated by default, then it can not be achieved these additional operations. However, if the destructor is explicit, the constructor copies usually need to be explicit. As the body object may be copied, therefore, it should not be modified in its operator () method, so parallel _for will declare operator () method of the body object as const.

3.2 Experimental Results and Analysis

Experiment has been performed on one platform with sequential program and parallel program so as to testify the efficiency of parallel algorithm. The specific environment is given below.

Software environment: Windows XP, Microsoft Visual Studio 2008，tbb30_20100915oss.

Hardware platform: Intel Core2 Quad Q9400 2.66GHz, memory of DDR3 4G.

Table 1 and figure 1 show the experimental results which gives the speed-up for different problems. Picture 1 is an office picture; Picture 2 is a personal photo, it contains less amount of information ; Picture 3 is 1/16 of 292M China map from Internet; Picture 4 is 1/8 of China map; Picture 6 is 1/4 of China map; Picture 6 is 1/2 of China map. The amount of information contained in the China map is relatively large, sequential processing required for a long time, Execution time is given in seconds.

Table 1: Experimental results

Example	Image size (M)	Image width and height (pixels)	Serial time (S)	Parallel time (S)	Speed-up
1	1.48	868*600	0.468	0.180	2.600
2	34.3	4000*3000	3.781	1.461	2.588
3	18.1	2772*2286	20.204	6.602	3.060
4	35.8	2795*4481	85.422	24.562	3.478
5	60.1	5973*4481	285.093	80.344	3.548
6	152	11922*4481	876.016	240.156	3.648

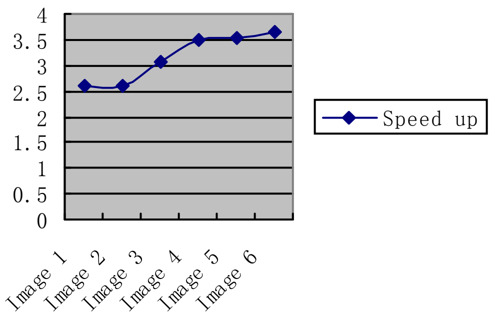

Figure 1 TBB circle detection speedup.

Based on the table 1 experimental results, some phenomena can be concluded.

1) Although the image size and the number of pixels of the picture 2 are larger than the picture 3's, the selected picture is relatively simple, and round number is much less than the picture 3's, so the execution time is less than the picture 3.

2) Though TBB task scheduling mechanism, the program is mapped to the four physical threads, the speed can, in
theory, be increased by four times. However, due to communication synchronization and thread overhead consumption, and the program still exists the part of the serial procedures and other system overhead, Lead to the improvement program running speed will not reach the theoretical value.

3) Start multiple threads, when the scale of the problem is smaller, switching between threads frequent and initializing multiple threads will take up more time, so the speedup is not very large. With the expansion of the scale of the problem, the CPU usage is also growing, and multi-core advantage is gradually reflected, The proportion of time to start multiple threads in the program and the total execution time is getting smaller and smaller, so the speedup increasing.

4. Parallel Circle Detection Method Based on CUDA

4.1 Algorithm Principle

For the most time-consuming part of circle detection of sequential program, recorded the pixel of the gradient amplitude is larger than the threshold, and gradient direction of the pixel is set to 1, gradient direction of the pixel of the gradient amplitude smaller than the threshold is set to 0. Scan all non-zero pixels and along the gradient direction on the corresponding two-dimensional accumulation array unit to add 1, the local maximum value in the accumulation array is the center coordinates. This part can do parallel processing on the GPU, The steps of the algorithm are as follows:

Step1: open space and pass data to memory for the graphics card.

//Open space
CUDA_SAFE_CALL
(cudaMalloc((void**)&d_edges_data_ptr, sizeof(uchar)*edges->height *edges->width));
// pass data to memory for the graphics card
CUDA_SAFE_CALL
(cudaMemcpy(d_edges_data_ptr,edges_data_ptr, sizeof(uchar)*edges-> height*edges->width,cudaMemcpyHostToDevice));

Step 2: set the thread number and the number of blocks:
int THREAD_NUM = 512;
int blockNum = (width*height)/THREADS_NUM + ((width*height)%THREADS_NUM==0?0:1);

Step 3: calling kernel functions Hough2, function prototype is as follows:
__global__ static void Hough2 (float idp,int rows, int cols,uchar* edges_data_ptr,int edges_step,uchar* dx_data_ptr, int dx_step, int dy_step,uchar* dy_data_ptr, int min_radius, int max_radius,int acols, int arows, int* adata, int astep)

Step 4: copy the execution result of the kernel function on the graphics card back to memory
//Pass the result to memory
CUDA_SAFE_CALL(cudaMemcpy(adata,d_adata, sizeof(int)*accum->height*accum ->width, cudaMemcpy-DeviceToHost));;

Step 5: free memory space. Such as:
CUDA_SAFE_CALL(cudaFree(d_edges_data_ptr));

4.2 Experimental Results and Analysis

Experiment has been performed on one platform with sequential program and parallel program so as to testify the efficiency of parallel algorithm. The specific environment is given below.

Software environment: Windows XP, Microsoft Visual Studio 2008，devdriver_3.2.

Hardware platform: Intel Core2 Quad Q9400 2.66GHz, memory of DDR3 4G, NVIDIA GeForce GTX 260, 896 MB graphics memory,192 processing cores, 16KB shared memory, 8192 available registers in each Block, 32 Warps, the maximum number of threads each block is 512.

Table 2 and figure 2 show the experimental results which gives the speed-up for different problems. The experimental pictures are the same picture with TBB experimental pictures. Execution time is given in seconds.

Table 2: Experimental results

Example	Image size (M)	Image width and height (pixels)	Serial time (S)	Parallel time (S)	Speed-up
1	1.48	868*600	0.468	0.818	0.572
2	34.3	4000* 3000	3.781	0.710	5.325
3	18.1	2772* 2286	20.204	2.317	8.720
4	35.8	2795* 4481	85.422	4.154	20.564
5	60.1	5973* 4481	285.093	6.236	45.717

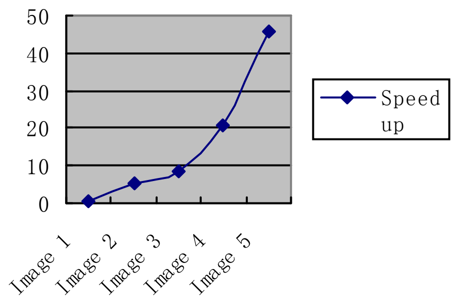

Figure 1 CUDA circles detection speedup.

Based on table 2 experimental results, some phenomena can be concluded. With the increase in the amount of data, the parallel algorithm speedup on GPU increasing, however, the table shows two strange phenomenon: 1) Picture 1 sequential program execution time is faster than parallel program; 2) Picture 2 sequential program execution time is slower than picture 1, parallel execution time is faster than the picture 1. The reason of the phenomenon 1: When the amount of data is small, Telecommunication transmission time of GPU and CPU is greater than the shortened time of the program executed on GPU.

The reason of the phenomenon 2: sequential program detects picture 2, the time consumption in CUDA parallel part, once this part uses the CUDA parallel execution, execution time is significantly shortened. While the parallel execution time of detecting Picture 1 is greater than the sequential detection time.

4. Conclusions

There is the problem of long computation time of Hough transform, This paper presents two parallel algorithm based on multi-core and GPU, Experimental result shows the maximum speedup can be achieved 3.648x based on 4-core system, the Maximum speedup can be achieved 45.7x based on GPU. Experiment results demonstrate the feasibility and efficiency of the parallel algorithms.

Acknowledgments

This research was supported by the National Natural Science Foundation of China (NSFC 60963004), and National Xinghuo project of China (No. 2011GA880001).

References

[1] J. Illingworth and J. Kittler, "The Adaptive Hough Transform", IEEE Trans, Pattem Analysis and Machine Intelligenee, 1987, 9 (5): 690-698.
[2] Wang Guohong, Kong Ming, He You, "Hough Transform and its application in information processing Hough", Bengjing: Ordnance Industry Press, 2005 (In Chinese).
[3] HAH, Kender J R, Shaw D E, "The analysis and performanceof two middle-level vision tasks on a fine-grained SIMD tree machine ", Binford T O. Proceedings IEEE Computer Society Confe rence on Computer Vision and Pattern Recognition[C]. Los Alamitos: IEEE Computer Society Press, 1985, pp. 387-393.
[4] Fisher A L, Highnam P T, "Computing the hough transform on a scan line array processor", IEEE Transactions on PAMI, 1989, 11 (3), pp. 262-265.
[5] Jolion J, Rosenfeld A, "An O (logn) pyramid Hough transform", Pattern Recognition Letters, 1989, 9 (5), pp. 343-349.
[6] Pan Y, Chung H Y H, "Faster line detection algorithms on enhanced mesh connected arrays", IEEE Proceeding-E, 1993, 2(140), pp. 95-100.
[7] Chung KL, Lin H Y, "Hough transform on reconfigurable meshes", Computer Vision and Image Understanding, 1995, 61(2), pp. 278-284.
[8] Kao TW, HorngS J, WangYL, "An O(1) time algorighmsfor computing histogram and the Hough transform on a crossbridge reconfigurable array of processors", IEEE Transactions on Systems, Man and Cyber-netics, 1995, 25(4), pp. 681-687.
[9] Lin S S, "Constanttime Hough transform on the processor arraywith reconfigurable bus systems ",computing, 1994, 52, pp. 1-15.
[10] Merry M, Baker J W, "Constant time algorithm for computing Houghtransform on a reconfigurable mesh", Image and Vision Computing, 1996, 14, pp. 35-37.
[11] Pan Y, "A more efficient constant time algorithm for computing the hough transform", Parallel Processing Letters, 1994, 4 (1/2), pp. 45-52.
[12] Pan Y, Li K, Hamdi M, "An improved constant time algorithm for computing the Radon and Hough transforms on a reconfigurable mesh", IEEE Transactions on Systems, Man, and Cybernetics, (Part A), 1999, 29(4), pp. 417-421.
[14] Pan Y, "Constant-time Hough transform a 3D reconfigurable mesh using fewer processors", Proceedings of 2000 Work-shops on Parallel and Distributed Processing, LNCS 1800 [C] Heidel-berg: Springer-Verlag, 2000, pp. 966-973.
[15] Pavel S, Akl S G, "Computing the Hough transformation on arrays with reconfigurable optical buse s [A]", Bosto n:Kluwer Academic Publishers, 1998, pp. 205-226.
[16] Li K, Pan Y, Zheng S Q, "Parallel Computing Using Optical Interconnections", Boston:Kluwer Academic Publishers, 1998, pp. 227-2 47.

[17]FISHER A L，HIGHNAM PT, “Computing the Hough transform on a scan line array processor”, IEEE Transactions on Pattern Analysis and Machine Intelligence，1989，11 (3), pp. 262-265.

[18] LI ZE-NIAN，TONG F，LAUGHLIN R G, “Parallel algorithms for line detection on a 1×N array processor”, Proceedings of the 1991 IEEE International Conference on Robotics and Automatio- n．Washington，DC: IEEE Press ，1991, pp. 2312-2318.

[19] CHUANG HYH，CHEN LING, “An efficient Hough transform algorithm on SIMD hypercube”, Proceedings of the 1994 International Conference on Parallel and Distributed Systems．Washing- ton，DC: IEEE Press，1994, pp. 236-241.

[20] ChenYu,Chen Jianhong,Xu Xiaohua, “A fast and efficient parallel algorithm for Hough transform”, Electronics Journal,，2004，32(5), pp. 759 -762．(In Chinese)

[21] Zhang Tong, Liu Zhao, Ouyang Ning, “Real-time line detection based on GPU”, Computer application.．2009, 5 Vol 29, No5, pp. 1359-1361．(In Chinese)

[22] CHEN Y-K，LIWEN-LONG，LI JIAN-GUO，etal, “Novel parallel Hough transform on multicore processors”, IEEE International Conference on Acoustics，Speech and Signal Processing: ICA SSP 2008．Washington，DC: IEEE Press，2008, pp. 1457-1460.

[23] Zhang Jie, Yang Xiaofei, Zhao Ruilian, “Precise positioning of the human eye based on Hough transform circle detection”, Computer Engineering and Application, 2005,41, pp. 43-44.

[24] Yang Quanying, “Image shape feature detection based on Hough transform”, Shandong University master degree thesis. (In Chinese)

[25] D．J．Kerbyson，T．J．Atherton, “Circle detection using Hough transforms filters”, In5th Int．Conf．On Image Processing and its Applications，Edinburgh，1995，410, pp. 370-374.

[26] Wang Qiang, Lu Zhimin, “A fast circle detection Hough algorithm”, Small micro-computer system. 2000, 21, pp. 970-973.

[27]James Reimnders, Nie Xuejun (translation) , “Intel Threading Building Blocks compile Guide”, Machinery Industry Press, January 2009, first edition.

Suping Wu received the computer soft and theory degree from Beijing Normal University. Currently, she is a professor at Ningxia University. Her interests are in parallel computer.

Xiangjiao Liu is a research student in computer soft and theory at Ningxia university . Her research interests in parallel computer.