A Parallel Pipelined Hough Transform *

N. Guil and E.L. Zapata

Dpt. of Computer Architecture. University of Málaga
Campus de Teatinos, P.O. Box 4114
29080-Málaga, Spain
E-mail:{nico,ezapata}@atc.ctima.uma.es

Abstract. The algorithms based on Hough Transform techniques to detect complex shapes, like circles and ellipses, require excessive computing time. In order to obtain better execution times we propose a new procedure to parallelize the detection process in a distributed memory multiprocessor. The sequential algorithm splits the detection of parameters into several stages and uses a focusing algorithm to implement the most complex ones. In this work, each stage is parallelized, in a pipelined fashion, solving the different problems that arise, specially the best load distribution to obtain a good balancing.

1 Introduction

The Hough Transform (HT) [1] is a method that permits recognising different shapes in an image. This technique has a great inmunity to noise but needs strong computational and memory requirements. In order to reduce these requirements different techniques have been applied to the sequential algorithms [2, 4]. However, after applying these improvements, the execution times may still be too high, specially in industrial environments where the decisions have to be taken in a fast way. In these cases, the use of parallel computers is necessary.

The parallelization of the classical HT can be carried out using loop projection techniques. These solutions have been used both in multiprocessors with a coarse grain [6] and in array processors with fine grain [7, 8, 9, 10]. The main problem in these implementations is the use of a common polling parameter space that causes contention problems in shared memory parallel computers and a large number of messages in distributed memory parallel computers.

The parallelization of algorithms based on a focusing process, i. e. Multiresolution Hough Transform [11], have been achieved using pyramidal architectures. However, the parallelization of irregular algorithms, like the Fast Hough Transform (FHT) [12] brings several problems related to the load distribution and the necessity of a mechanism that allows focusing on the solution as soon as possible eliminating, this way, useless computations.

In this paper, we present a new algorithm for a distributed memory multiprocessor that parallelizes the different stages of two fast algorithms for circle

* The work described in this paper was supported by the EC under project BRPR-CT96-0170

and ellipse detection called Fast Circle Hough Transform (FCHT) and Fast Ellipse Hough Transform (FEHT). These algorithms solve two kinds of problems: First, the problems that arise in the stages implemented with the classical HT method, during the polling process in a common parameter space. Second, the load distribution to obtain a good balancing in the computation of stages where a focusing algorithm is applied. We have organised the rest of the work as follows: in the next section we present two new fast sequential algorithms for circle and ellipse detection. In section 3 we show the different techniques we have performed to carry out the parallelization of the different stages. Finally, in section 4 we evaluate the proposed parallel algorithms with a real image.

2 Sequential algorithms

In this section we present the sequential algorithms for circle and ellipse detection. The circle and the ellipse algorithms are divided into two and three stages, respectably. The first stage, in both algorithms, finds the center of the shapes. In order to detect the center, we apply a new focusing algorithm derived from the FHT, that employs a new focusing strategy to save computations.

2.1 Focusing algorithm

Focusing algorithms perform a hierarchical approximation to the solution, starting from a coarse description of the parameter space [5]. In order to do this, a square parameter space is generated and divided into four quadrants. A given quadrant is divided again if an appreciable number of lines (larger than a threshold value) traverse it. The recursive application of this process will approximate us to the solution. Hence, during the execution of the FHT algorithm an irregular tree is generated. The tree nodes are associated to the quadrants. A parent node generates four children nodes if the number of lines than traverse it is higher than the threshold value.

2.2 Circle and ellipse detection

The fast sequential algorithm for the detection of circles we propose in this work, Fast Circle Hough Transform (FCHT), performs the detection of the circle in two stages. In the first stage the center of the circle is detected and the points are labelled using the FHT algorithm. In the second stage the radius is obtained with the help of this labelling.

The fast algorithm for ellipse detection, Fast Ellipse Hough Transform (FEHT), developed in this work uses three stages:a) detection of the center by means of a focusing algorithm that finds the point where a beam of lines intersect; b) production of the orientation angle and of the semiaxes ratio using information from previous stages and from the gradient vectors of the figures; c) calculation of the semiaxes.

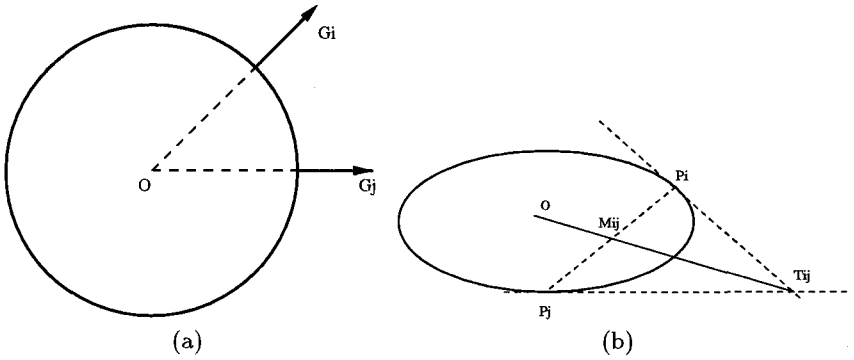

Fig. 1. (a) The lines that contain the gradient vectors (Gi and Gj) cross the circle center (b) The line that passes through the intersection point of the tangents generated by Pi and Pj and their middle point crosses the center of the ellipse.

Detection of the center The center detection for the circle a ellipse shapes is based on geometric properties of these shapes [3], as Figure 1.a and Figure 1.b show. In both cases, a beam of lines crossing the center is generated and the FHT can be used to find the intersection point.

Detection of the radius. Once the center of a circle is detected we can calculate its radius using the values for the distance between the center of the circle and each one of the points of the image. The distance values vote over a one dimensional space.

Detection of the orientation. We can use a two dimensional space where each point of the ellipse votes for different orientation angles. The maxima obtained after performing the process for all the points will indicate the possible orientation values and the semiaxis ratios of ellipses in the image.

The polling process is not carried out for all the points of the image. It is restricted only to those points that have participated in finding the center using a presence vector associated to the lines [5].

Detection of the semiaxis For all the points of the image that have collaborated in a maximum of the previous stage, we perform a polling process. The maxima found in the space will coincide with the values of the semiaxis of the detected ellipse. In the case where concentric ellipses are found in the image, there would be one maximum per ellipse.

3 Parallelization of the stages

From the execution of the sequential program, we know that the most complex stage in both algorithms is the center detection due to the high number of straight lines that are generated and the spread of these lines with respect to the real center of the shape (the spread will make the focusing process more difficult). Furthermore, during the execution of this stage, an irregular tree is generated. The paralellization of this tree is, clearly, the most difficult task in the algorithm if a good load balance is desired.

Before describing the parallel implementation of the stages we will discuss some considerations about the parallelization of the irregular algorithm. There are two possible approximations to the irregular tree parallelization in order to obtain a good load balancing [13, 14]. In the first one, a static distribution for the initial load is employed. In the second one, the load distribution is modified during the execution of the program to obtain a good balancing.

Each approximation will cause a different partition of the problem. In the static load balancing approximation, the initial data, that is the coefficients of the straight lines, are shared among the processors. Thus, all the processors will collaborate to calculate the computation in each node of the tree. However, in the dynamic load balancing only one processor will carry out the computation of a node. In this case, all the lines have to be stored in each processor.

In the static load balancing strategy it is necessary to guarantee an adequate initial distribution for the lines that generates a very similar number of computations in the different processors. In the dynamic load balancing strategy several mechanisms have to be established in order to evaluate the load of each processor, the load rebalancing criteria and the task migration implementation. These strategies have been evaluated in [15]. The conclusions indicate that the static strategy is more scalable than the dynamic one, due to the overhead generated, in communications and computations, by the dynamic balancing mechanisms.

Basing on these results we have choosen the static implementation for the center detection stage in the parallel circle and ellipse algorithm. Now, the main problem is to make all the processors perform a similar number of computations. The next condition will guarantee this behaviour: all the processors have to contain, approximately, the same number of lines from the different shapes in the image. This way, all the processors will collaborate, roughly, with the same number of computations in each node of the tree.

Next, we discuss, in detail, the parallelization of the different stages of the algorithms.

3.1 Edge detection and point pairing

The operations to know if an image point (x,y) is an edge point have to be performed within the neighbour points that lie in the window WxW centered at (x,y). In order to avoid communications among processors, these computations have to be carried out, as far as possible, in a local way. Therefore, a block distribution of the image is applied, previous to the edge point computation, so

as to restrict the communications to the points in the border of the blocks. At the end of this process, each processor will obtain the edge points that belong to its image block.

Before applying the center detection process in the FEHT algorithm, the pairing of points, to calculate the straight lines, has to be performed. In order to balance these computations all the edge points have to be known by all the processors. The overhead in communications and storage for this distribution is not very important because the rate of edge points in a image is usually very low (1%). At the end of these communications, all the processors will have an edge point list:

$$EL = \{E_1, E_2, ..., E_n\}$$

where n is the number of edge points in the image.

Using this list, each processor will search for a pairing subset. In fact, processor j will find the pairings, taking into account the constraints indicated in the previous section, between point E_i and E_k $(k > j)$ only if $jmodP = i$, where mod indicates the module operation and P is the number of processors. This assignment will permit achieving these three aims: a) the processors will perform a very similar number of computations to calculate the straight lines, b) the number of lines obtained in the different processors will be very close, too and c) each processor will have lines that belong to different objects in the image.

For the circles, like in the ellipse method, a previous distribution of all edge points in each processor will be done. The edge points will be stored in a list in each processor. Using this list, each processor p_j computes the line generated by EP_i only if $imodP$ is coincident with the index j of the processor.

3.2 Center detection

As we have seen, the lines have been distributed among the different processors. This way each processor will collaborate to the computation in each node. In order to do this, each processor will calculate the local number of lines that transverse a quadrant (node). Then, these local numbers are added to find the total number of lines that cross the quadrant. In this way, global number of computations of the parallel algorithm will be coincident with the sequential one. However, to evaluate the efficiency of the parallel algorithm the communications must be taken into account.

The number of communications in the parallel algorithm will depend of the focusing policy used to find the solution. In the references two basic methods have been studied to perform the searching: Depth-First and Best-First. In a first approximation, the Depth-First policy is the most adequate one to solve a general problem since it searchs the solution in a fast way, allowing line elimination and saving, in this manner, computations. However, as indicated in [15], this policy can be too restrictive when several shapes appear in the image. In this case, we recommend expanding several nodes per level and choosing the best ones.

This new policy will improve the execution time, allowing a faster focusing, and reduce the number of communications since the total sum for several nodes can be calculated and broadcast at the same time (the number of packages will be smaller but their sizes will be larger).

Whenever a center is found, the subsequent stage starts to calculate the rest of the parameter for this solution, while a new center is been searching. So the different stages can be working simultaneosly in a pipeline fashion.

3.3 Orientation and semiaxes rate detection

After applying the center detection, the lines that have contributed to find the shape centers are labelled with the presence vector. Using the information of the presence vector, each processor obtains the points that have collaborated to generate these lines. In this situation, it is important to realise that the information of the presence vector is local. Thus, the edge points obtained in each processor can be different. However, the same point or point group can be obtained in different processors due to the fact that a particular point may generate lines in several different processors.

The computation in this stage is performed using two nested loops that scan all the labelled points for a rank of angles within 0° and 180°. The parallelization could project each loop over the dimension of a mesh using a block distribution for the data. This distribution would allow the parallelization of the polling process, but it would need a large number of communications to obtain the global sum of the partial Hough spaces that have been calculated. To avoid this problem, we propose a different distribution of data. In this distribution, all the labelled edge points have to be shared among the processors (this situation is already a fact) and only the angles will be distributed through assigning different angle rank to each processor (block distribution). Hence, at the end of the polling process, all the processors can apply, in parallel, the maximum searching process.

Two communications will be necessary in this stage. First, all the labelled edge points have to be known by all the processors. This information is not very large since it is going to be restricted to the edge points that have generated lines that cross the center found. Second, at the end of the maximum detection, the local maxima found have to be broadcast in order to find the global maximum.

3.4 Semiaxis detection

To parallelize this stage, each processor uses a subset of the labelled edge points in order to perform the polling on the parameter spaces. Thus, each processor will calculate a local Hough space. All the local Hough spaces have to be added before applying the searching maximum process to find the semiaxes values. However, the computational complexity of this stage is usually very low and the communication time can be longer than the computational one. On the other hand, the parameter space votes are distributed in a sparse way because they tend to accumulate in a restricted number of positions. Then, we can limit the

information that is going to be transmitted in this stage to only the positions, in the local Hough spaces, which have a value different from zero.

4 Evaluation

In this section we indicate the result obtained in the Paramid multiprocessor with 16 nodes (one i860 and one transputer per node). The message passing routines have been written using PVM. On the other hand, the image that we have used is shown in the Figure 2.a. In this image, four ellipses appear: two are concentric and two are partially overlapped. The size of the image is 512x512 pixels (256 grey levels).

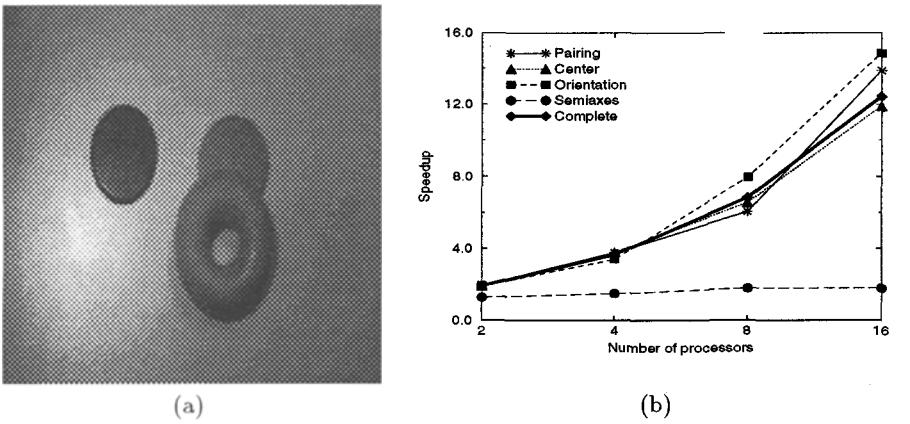

Fig. 2. (a) Original image with ellipses (b) Speedups for the different stages

In the Figure 2.b we show the speedups for the completed algorithm (including the execution times for all the stages). The speedups for the pairing points, and center, orientation and semiaxes detection are also indicated. The best values are achieved by the pairing and orientation stages. The speedup for the center is not so good due to the communications needed to calculate the node votes. The worst values are obtained for the semiaxes detection because the complexity of the computations are very low. So that, when the number of processor increases and, so, the communications spend more time, the speedup remains almost constant. In any case, the semiaxes time detection is not significant in the complete algorithm.

5 Conclusions

In this work we have shown the complete parallelization of two algorithms for circle and ellipse detection based on Hough Transform. The parameter calculation has been uncoupled using several pipelined stages. Furthermore, each stage has been parallelized using different data distributions. These techniques have permited achieving good values for the speedup.

References

1. P.V.C. Hough. "Method and Means for Recognizing Complex Patterns". U.S. Patent 3069654, 1962.
2. C. Ho and L. Chen. "A Fast ellipse-circle detector using symmetry". Pattern Recognition 28 (1), p. 117-124 (1995).
3. H.K. Yuen, J. Illindworth y J. Kittler. "Detecting partially occluded ellipses using the Hough Transform". Image and Vision Computing, 7 (1), 1989, pp. 31-37.
4. H.K. Muammar y M. Nixon. "Tristage transform for multiple ellipse extraction". IEE Proceedings-E 138 (1), 1991, pp. 27-35.
5. N. Guil, J. Villalba and E.L. Zapata, "A Fast Hough Transform for segment detection", IEEE Transactions on Image Processing, vol. 4, no. 11, Noviembre, pp. 1541-1548,1995.
6. A.N. Choudhary and R. Ponnusamy. "Implementation and evaluation of Hough Transform algorithms on a shared-memory multiprocessor". Journal of Parallel and Distributed Computing 12, pp. 178-188, 1991.
7. D. Ben-Tzvi, A. Naovi and M. Sandler. "Syncronous Multiprocessor Implementation of the Hough Transform". CVGIP 52, pp. 437-446 (1990).
8. A. Rosenfeld, J. Ornelas and Y. Hung. "Hough transform algorithms for mesh-connected SIMD parallel processor". J. Computer Vision Graphics Imagen Processing, Vol. 41, pp. 293-305, 1988.
9. M. Ferreti. "The Generalized Hough Transform on Mesh-Connected Computers". Journal of Parallel and Distributed Computing, 19, pp. 51-57, 1993.
10. S. Kumar, N. Ranganathan and D. Golgof. "Parallel algorthims for circle detection in images". Pattern Recognition 27 (8), pp. 1019-1028 (1994).
11. M. Atiquzzaman. "Multiresolution Hough Transform. An efficient method of detecting patterns in images", IEEE Transactions on PAMI 14 (11), 1090-1095 (1992).
12. H. Li, M.A. Lavin and R.J. Le Master, "Fast Hough Transform: A Hierarchical Approach", CVGIP 36, pp. 139-161, 1986.
13. D. Gerogiannis and S.C. Orphanoudakis. "Load Balancing Requirements in Parallel Implementation of Image Feature Extraction Tasks". IEEE Transactions on Parallel and Distributed Systems, Vol. 4, 9, 1993.
14. M.H. Willebeek-LeMair and A.P. Reeves. "Strategies for Dynamic Load Balancing on Highly Parallel Computers". IEEE Transactions on parallel and distributed processing (4), 9, 979-993, 1993.
15. N. Guil, "Hough Transform in Multiprocessors", PhD dissertation, Dept. Computer Architecture, Malaga University, December, 1995.