Fast Hough Transform on Multiprocessors: 
A Branch and Bound Approach

N. Guil and E. L. Zapata

Department of Computer Architecture, University of Malaga, Campus de Tectinos P.O. Box 4114, 29080 Malaga, Spain

The algorithm of the Fast Hough Transform presents a strong irregularity which makes its parallelization difficult, especially if we introduce certain improvements into the sequential algorithm, such as the elimination of straight lines. In this work we approach this algorithm as a branch and bound problem, developing three different parallel algorithms. The first of these algorithms is based on an initial distribution of the problem that guarantees a good balance of computations. The other two algorithms perform a dynamic distribution of the load using different policies. In one of them, the mechanism is distributed and rebalancing is carried out at the demand of the processors left without load. In the other, one processor acts as a central processor and decides when to carry out the rebalance and what the correspondence between overloaded and underloaded processors will be. © 1997 Academic Press

1. INTRODUCTION

In order to reduce the requirements of the Hough transform (HT) [1] several techniques have been applied to the sequential algorithms. Most of these techniques employ geometric features based on the figures to be detected [2, 3]. In addition, the production of parameters is usually uncoupled so that the detection process is decomposed into a series of stages [4, 5]. This way, a smaller number of parameters have to be detected at each stage, which greatly improves the requirements of the algorithm. Other sequential algorithms, in addition to applying the decomposition of the parameter space, have employed fast algorithms in order to implement some of the stages into which the algorithms have been divided. These implementations are based on the modification of the fast Hough transform algorithms (FHT) [6], which is, from a computational viewpoint, more appropriate than the traditional transform [7–9].

However, even after these improvements the execution times of the algorithms may be too long, especially in industrial environments where it is necessary to make fast decisions. For this reason, better execution times have been sought by applying parallel techniques to the algorithms.

The parallelization of the classical formulation of the Hough transform does not present great conceptualization problems, and may be addressed using loop projection techniques. Some of these solutions have been applied to multiprocessors using coarse grain in order to distribute the processes [10]. Others have been applied to array processors using fine grain [11–14]. The main problems of both implementations are due to the use of a common parameter space which generates many contention problems in machines with shared memory, and produces a large number of messages in distributed memory machines.

The parallelization of improved algorithms based on focalizations, such as the multisolution Hough transform [15], has also been obtained using pyramid architectures. However, the parallelization of irregular sequential algorithms such as the FHT presents great problems due to the need to adequately distribute the load and implement a mechanism that allows us to focus as fast as possible without useless computations.

In this work we present three different algorithms for the parallelization of the FHT algorithm using different strategies for distributing the points of the image and balancing the load. In order to perform the parallelization, we have formulated the algorithm of the FHT as a branch and bound problem and have solved the implementation of the rules applied to algorithms of this nature in several ways with the aim of comparing the results obtained.

The work is divided as follows. In the next section we present the modified FHT algorithm and its characterization as a branch and bound algorithm. In Section 3 we present three parallelization strategies for the algorithm. In one of them we do not employ dynamic rebalancing, whereas in the other two we use it. In the latter two, we establish different mechanisms for distributing the load for comparison. In Section 4 we evaluate the different algorithms using a real image as an example and we compare the results.

2. FAST HOUGH TRANSFORM: BRANCH AND BOUND APPROXIMATION

As has been shown in [8, 9], the algorithms for the detection of circles, ellipses, and arbitrary figures may be decomposed into series of stages. The stage that uses more execution time
in these algorithms has been implemented using a new focalization algorithm derived from the FHT [6]. Although this algorithm presents some problems [9], it is shown to be a fast and efficient method for detecting the figures.

In this work the FHT algorithm is applied to two-dimensional parameter spaces, and thus the solutions originate at the crossing point of a line beam (generated lines). The focalization to the solution is carried out by means of a division and deepening process in the parameter space. To this end, a square parameter space is generated and divided into four quadrants. A given quadrant is again divided if a noticeable number of lines traverse it (greater than a threshold value). The recursive application of this process will approach a solution.

With the objective of reducing calculations, we define a presence vector that will indicate which lines of the parameter space traverse a given quadrant. When a solution is reached, the lines belonging to this solution may be eliminated so that they are not considered in the search for the rest of the solutions.

The way in which each object generates the beam depends on its shape. In the case of segments, circles, and arbitrary shaped objects, each edge point generates a line. However, for the case of ellipses, the generated lines are obtained by means of a pairing point process. This process tries to pair edge points that belong to the same ellipse [4, 9].

Analyzing the fast algorithm, we may establish the similarities to a generic branch and bound problem, as shown in Fig. 1. Thus, during the execution of the focusing process, an irregular computation tree is generated. Each tree node is associated with one parameter space quadrant. The node cost is defined as the inverse of the number of lines that traverse a quadrant. In this way, the quadrants with a larger number of lines will present a smaller cost. There is an implicit relationship between the cost of a child node and a parent node. As the child node is contained in the parent node, it will have at least the same cost as the parent (the same number of lines) although, in general, this cost will be larger.

3. PARALLELIZATION OF THE FHT ALGORITHM

In this section we propose two schemes for carrying out the load distribution that will lead to two different algorithms, which we will call the static balance algorithm for the FHT (SB) and the dynamic balance algorithm for the FHT (DB). On the other hand, we have assumed that the lines have been previously calculated. Note that this calculation is independent from the FHT application. The problem of the parallel computation of the line coefficients will not be dealt with in this paper, although different approaches are shown in [21].

3.1. Static Load Balancing

In the SB algorithm, the lines are distributed among the different processors so that each processor collaborates in the computation of each node. The total voting of the number of lines that traverse a node will be found from the values of the partial sums calculated by each processor.

In order to do this we will apply a global sum process which will obtain, for all the processors of the parallel computer, the total of the partial sums. This way, the number of nodes computed in the parallel algorithm is going to be the same as in the sequential algorithm, as the route followed is exactly the same. Therefore, from the point of view of the computation of nodes, the parallelization may seem correct. However, within each node, the computation carried out by each processor may differ, as each processor has a different set of lines.

There are two effects that may negatively influence the balanced distribution of the computations among the processors:

- The first is the typical problem when the generated lines are not distributed in a balanced way among the processors. It is then clear that all the processors are not going to perform the
same number of computations, and therefore, the performance of the system will degrade.

- Even if we manage to distribute the same number of pixels to every processor, this will not guarantee a balanced computation. As we have already indicated, in the FHT algorithm the edge points of an object generate a line beam that crosses at a point (solution). In order to reach a solution, the algorithm must go deep, rejecting lines that do not belong to the solution. Therefore, if we want to obtain a balanced computation, the generated lines must be distributed so that all of the processors have the same number (or as close as possible) of lines that belong to the same object.

Here we propose a new strategy for distributing the lines of the image among the processors. Let us assume that as the generated lines of the image are obtained, they are stored in a linear array of the form \{\(r_1, r_2, \ldots, r_R\)\}, where \(R\) is the number of generated lines obtained in the image. This way, if we assign consecutive lines to different processors we will guarantee that all of them have a very similar number of lines for a given figure.

With this distribution we will achieve good load balances, and so this may seem to be a good method for parallelizing the algorithm. However, it is important to examine the impact of communications on the execution of the parallel code in order to establish the effectiveness of the implementation. The number of communications to be carried out in the algorithm will depend on the policy for finding the solution.

The best breadth and depth search policy [9] is the most suitable for solving our problem. It is advisable to allow several nodes to branch at each level and to choose among the child nodes those that present the largest values. In addition, this will have a positive influence on the number of communications of the parallel algorithm. Thus, when several nodes of the same level are branched simultaneously, the number of communications to be carried out decreases, thus permitting larger packets also.

Every time a set of nodes have been branched in a level, they are eliminated from the list of nodes of this level. In the same manner, every time the nodes branch, the level at which the new child nodes are located must be checked, because if they belong to the bottom of the tree they will lead to a solution. In this case, we must proceed to eliminate lines from the parameter space, a process that will modify the values of the votes calculated up to that instant of time on all the levels.

### 3.2. Dynamic Load Balancing

The previous static distribution algorithm presents the problem of necessarily performing a synchronization (global sum) every time we want to expand new nodes. This process generates a large number of computations and even though we have managed to pack and reduce their number, the performance may degrade when the number of processors is relatively large.

As an alternative to the previous distribution, we propose a new strategy that dynamically distributes the nodes to be computed among the different processors. Other rebalancing mechanisms, for image processing, have been introduced in [19, 22]. In our case, as the processors must work on different nodes, it is necessary for each to have information about all the lines of the parameter space. In this way, the computation is carried out in an independent manner. This implies that the coefficients of the lines must be replicated in all the processors. Even though it may seem an unacceptable proposition regarding memory expense, it is important to observe that the number of lines is usually kept at acceptable levels, so this replication of the information is not very costly resourcewise.

In order to conveniently evaluate the possibilities of dynamic load balancing we have employed two different control strategies: distributed and centralized. The first uses a simple mechanism that introduces very little overhead in the total computation time of the parallel program. The second is designed by means of a centralized balancing mechanism which, although adding more computational complexity to the algorithm, saves computations in the solution search process. In both strategies the node is the load measurement unit.

#### 3.2.1. Distributed Dynamic Algorithm

Initially, the parameter space is divided among all the processors so that each processor is assigned an equivalent space (block distribution). Each processor, therefore, works with this subspace assigned as root node of its initial tree and from there on it applies the sequential algorithm, without any need of performing communications, until it is left without any nodes to compute. In the distributed control parallel algorithm for the dynamic distribution of the load (DDB), the stages of the rebalancing process are the following [20]:

**Evaluation of the processor load and determination of the benefit in the load distribution.** When a processor becomes idle, it asks the rest of the processors for a new load. Thus, rebalancing is carried out whenever a processor demands it. This policy is useful in this case because the information to be transferred is not very large and it simplifies the used rebalancing mechanism.

**Task migration strategy.** In our case, the processor that starts the rebalancing is the destination processor. In this way, when a processor that had initially been assigned some nodes to be computed finishes its computations, it sends a request to all the processors of the parallel computer for them to send it new nodes to compute. We have decided to transmit only the coordinates of the center of the node (quadrant), the size, and the presence vector. In this way, for each node received, the destination processor must initially calculate the distances of this node without any help from the distances calculated for its parent node. However, it will be able to use the value of the presence bits so as not to carry out some of these initial computations.
FIG. 2. Distributed management of the load distribution.

Task selection strategy. As we have already mentioned, as a processor deepens its search through a series of branches, it leaves others open over which it must go back later. Therefore, it will have a series of nodes belonging to different levels available for transference. Out of all these nodes, the algorithm will choose the one belonging to the lowest level as, in principle, it will lead to the largest number of computations. If there is more than one node in the lowest level, the one with the largest voting will be transferred.

In Fig. 2 we display an outline of the load distribution process. Processor P4 is momentarily left without load and so sends at instant 1 a request for load to the remaining processors. These processors are evaluating their particular tree: the computation is being carried out at the deepest nodes, although some nodes of lower levels have not branched yet. These nodes are the ones transferred to processor P4 in instant 2.

3.2.2. Centralized Dynamic Algorithm

In presenting the features of the Centralized Dynamic Balancing algorithm (CDB), we will indicate only the implementation of the stages which are different from the previous one.

Evaluation of the processor load. In this algorithm, the evaluation of the processor load is carried out in a central processor, and this processor is also in charge, depending on the results obtained, of ordering the migration of tasks between processors of the parallel computer.

Determination of the profit of the load distribution. It is convenient to restrict the updates of the central processor. This may be achieved, for instance, by not carrying them out if the level of the tree being updated is higher than a threshold level. This does not cause severe imbalance problems, because the deepest nodes require fewer computations, and, in addition, the migration time of these nodes does not compensate for the computations they are going to generate.

Task migration strategy. A band around the mean value is established, so that a processor is assigned to one of the two groups (overloaded or underloaded) if it is outside this band. After this, the processors belonging to one of these two groups are ordered according to their load.

Task selection strategy. Due to the application of the task migration strategy in the central processor, it decides which source processor must send a task to which destination processor and what this task is. This information is sent to the source processor that selects the particular task and sends it to the destination processor. It may happen that due to the delay between the decision by the central processor of the processors that have to send data and the time the message arrives at the processors, the node it had to transmit has already been processed. In this case, the task will not be sent.

In Fig. 3 we present the dynamic load distribution scheme. We have assumed, in order not to make the example complicated, that the central processor, P4, does not perform computations of the transform algorithm, and is thus exclusively used for balancing tasks. The processors update the central processor every time a change in the contents of their particular trees occurs. With this information, P4 calculates which processors need to rebalance their load. These calculations reach the con-
clusion that processor P3 must send a node from level 3 of its particular tree to processor P1 (processor P1 is carrying out computations of nodes belonging to higher levels). The central processor sends this order to P3 which executes the transference process.

4. EVALUATION

We have implemented the code in both a Fujitsu AP-1000 (MPI library) and a Cray-T3E (PVM library) parallel computer. Two different images have been used to test the behavior of the algorithms. Figure 4a is made up of four ellipses. The edge operator detected 715 points. After pairing points more than 24,000 lines were obtained. The value for the threshold is 1000. Figure 4b is a very noisy image which has circles of different sizes. At the end of the edge detection 8932 lines are generated, but the number of edge points of several circles is very low as a consequence of noise elimination. Thus, the threshold value in this example is 50 so as to detect all the shapes.

We now discuss the improvements that have been carried out on the proposed algorithms.

4.1. Static Load Balancing Algorithm

Taking into account that several children are evaluated on each tree level as the parent nodes are expanded, an overlap between the communications and the computations can be applied. Thus, while the partial voting for a quadrant (node) is being calculated in the processors, the communications for the global voting calculation of the previous node can be happening simultaneously. In this way, if the number of local computations is high enough, most of the communications can be hidden.

4.2. Dynamic Distributed Load Balancing Algorithm

An important aspect when selecting in the distributed algorithm is the number of requests that must be sent by a processor when it is left without load, and which must be the recipients of these requirements. For a large number of processors, the request must be limited for two reasons: the number of communications may be excessive, with consequent congestion of the interconnection network and the load received by the requesting processor may also be too large. We have chosen a scheme in which the processor sends limited requests in a random way to the rest of the processors. Although the communications will be a little slower (as they are not carried out between neighbors) we guarantee a better exchange of the load. In this example we have allowed that while the number of processors is less than or equal to 8, load requests are carried out by all the processors. In the case of 16 processors, we will only allow requests to eight processors, and for higher number of processors the number of requests is 4.

4.3. Dynamic Centralized Load Balancing Algorithm

The updating of the central processor tree has been limited to the lower levels of the local trees. Of course this limitation produces a negative influence on the load balancing, though keeping low the number of communications. Furthermore, the deepest nodes on the local trees contain a low number of lines and so an eventual migration is not worthwhile due to their low computational cost.

4.4. Experimental Results

In Table I the execution times for the three proposed balancing strategies in the AP1000 are shown. In both examples the behavior of these algorithms is the same. Thus, the CDB and SB obtain the best values for a number of processor lower
TABLE I
Execution Times, in Seconds, for Different Numbers of Processors Using Three Load Balancing Strategies in the AP-1000

<table>
<thead>
<tr>
<th>P</th>
<th>Ellipses</th>
<th>Circles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DDB</td>
<td>CDB</td>
</tr>
<tr>
<td>1</td>
<td>285.7</td>
<td>285.8</td>
</tr>
<tr>
<td>2</td>
<td>162.7</td>
<td>201.0</td>
</tr>
<tr>
<td>4</td>
<td>64.1</td>
<td>70.0</td>
</tr>
<tr>
<td>8</td>
<td>45.8</td>
<td>35.5</td>
</tr>
<tr>
<td>16</td>
<td>29.8</td>
<td>20.1</td>
</tr>
<tr>
<td>32</td>
<td>21.0</td>
<td>15.9</td>
</tr>
<tr>
<td>64</td>
<td>18.0</td>
<td>12.6</td>
</tr>
<tr>
<td>128</td>
<td>10.7</td>
<td>29.5</td>
</tr>
</tbody>
</table>

than 64, mainly due to the number of communications being low with respect to the computations. However, the DDB algorithm achieves the best values for a high number of processors because it needs fewer communications than the other ones. In fact, in the DDB algorithm, the processors compute without carrying out communications until they become idle.

The speed-up of the algorithms can be compared in Figs. 5a and 5b. We can see that superlinearity is achieved with two and four processors. The explanation of these results is related to the cache of the processors. In the static algorithm, the data over which they operate are distributed among the processors, and for this reason, as the number of processors grows, the number of data per processor decreases. This way, the datas may be kept in the cache, generating a lower number of misses. This reduction in access times will achieve gains that are greater than expected. However, as the number of processors continues to grow, the positive effect of the spatial locality of the data is going to be reduced by the increase in the number of communications.

It can also be observed that the SB speed-up is the worst in circle detection because the number of nodes that it is necessary to evaluate is greater than in the ellipse detection, and the computation of these nodes is clearly lower (the ellipse detection threshold is 20 times greater than for circle detection).

On the other hand, the speed-up of the CDB algorithm is better than the SB one, in most cases. As can be observed in Fig. 5, the gain of the distributed dynamic algorithm becomes worse as we increase the number of processors. However, as these algorithms try to keep the number of communications as low as possible, its results are the best of all for a high number of processors.

The speed-up values for the Cray parallel machine are shown in Fig. 6. The execution times in the Cray machine are lower than in the AP-1000 mainly because of the computational power differences between the processors of both machines. For example, ellipse and circle detections in the Cray-3TE take 18.00 and 63.75 s, respectively, in one processor.

In the Cray-T3E, the SB algorithm obtains the best results during ellipse detection due to the computations being overlapped with the communications. Furthermore, superlinearity is achieved for a wide range of processors because of the on-chip two levels cache size that allows exploiting the locality of the problem. The circle detection results are less good due to the low computation that is carried out in each node. This does not allow overlapping among communications and computations in the SB algorithm and produces bad speed-up values for the dynamic algorithms.

![FIG. 5. Speed-up for the ellipse image (a) and circle image (b) using DDB, CDB, and SB algorithms in the AP-1000.](image)
FIG. 6. Speed-up for the ellipse image (a) and circle image (b) using DDB, CDB, and SB algorithms in the Cray-T3E.

5. CONCLUSIONS

Three parallel algorithms for the execution of the fast Hough transform have been presented. The characteristics of these algorithms have been adapted to different data distribution and load balancing strategies. Thus, a static load balancing based on a cyclic data distribution and two dynamic load balancings based on independent tree node computations have been presented. The scalability of these algorithms have been also studied.

REFERENCES


NICOLAS GUIL received the B.S. in physics from the University of Sevilla, Spain, in 1986 and the Ph.D. in computer science from the University of Malaga in 1995. He worked in the R&D department of a computer enterprise
from 1986 to 1990. During 1990–1997 he has been an assistant professor in the Department of Computer Architecture at the University of Malaga in Spain. His research interests include image processing and parallel computation.

EMILIO L. ZAPATA received the B.S. in physics from the University of Granada, Spain, in 1978 and the Ph.D. in physics from the University of Santiago de Compostela in 1983. During 1978–1981 he was an assistant professor at the University of Granada, and during 1982–1991 he successively was assistant, associate, and full professor at the University of Santiago de Compostela. Currently, he is a professor with the Department of Computer Architecture in the University of Malaga. His research interests are in the area of parallel computation architectures, parallel algorithms, numerical algorithms for dense and sparse matrices, and VLSI digital signal processing. Dr. Zapata has published more than 40 papers in refereed international journals in his area of interest and about 60 refereed international conference proceedings.

Received April 7, 1997; revised June 30, 1997; accepted July 1, 1997