Pattern Recognition Letters 15 (1994) 253–259
North-Holland

March 1994

PATREC 1181

# A parallel Hough transform algorithm for nonuniform images

Fevzi O. Ozbek and Meghanad D. Wagh *

*Department of Electrical Engineering and Computer Science, Lehigh University, Bethlehem, PA 18015, USA*

Received 3 September 1992

*Abstract*

Ozbek, F.O. and M.D. Wagh, A parallel Hough transform algorithm for nonuniform images, Pattern Recognition Letters 15 (1994) 253–259.

A Distributed Hough Transform Algorithm (DHTA) is proposed and its performance is analysed. It is shown that by distributing the algorithm rather than the image or transform space, one can avoid the asynchronism overheads. Mapping of DHTA on real architectures is discussed and results on linear array, mesh and hypercube are presented.

*Keywords.* Hough transform, load balancing, parallel algorithms.

## 1. Introduction

Hough Transform is rapidly becoming indispensable to industrial automation, diagnostic health care and national defense as a powerful tool to recognize parametrically described curves. Pattern recognition using Hough transform is a two-step process. In the first step, Hough transform is used to translate a binary image consisting of black curves on light background into a parameter space. Each point of this parameter space represents a curve with the corresponding parameter attributes. The translation proceeds by examining each image pixel and for every pixel and for every black pixel, computing the parameters of all possible curves to which that pixel may belong. The cells corresponding to these curves in the parameter space are appropriately marked. At the end of this translation process, the parameter space holds information about the number of image pixels on each curve. The second step of pattern recognition simply searches the parameter space for cells containing the highest number of marks identifying the curves present in the image.

Hough transform computation is very intensive. Curves such as straight lines or circles are parametrized by two independent parameters each of which is discretized into $M$ levels giving an $M \times M$ parameter space. Each black point in the image space can then potentially generate $M$ points in the parameter space which are determined by varying one of the parameters through its $M$ values and computing the other each time. Assuming that this computation takes a constant time, Hough transformation of an $N \times N$ image into an $M \times M$ parameter space implies a $O(N^2M)$ complexity. Naturally, substantial research effort in recent years is devoted to improving the computation of Hough transform.

One approach that has been very successful in improving Hough transformation speed is the use of

* *Corresponding author.* Email: mdw0@lehigh.edu

This research was partially supported by a grant from the Center for Manufacturing Systems Engineering at Lehigh University.

parallel architectures. Since the computations corresponding to each image pixel are independent, they may be carried out concurrently. Similarly the computations corresponding to different parameter regions are independent and can also be carried out concurrently. The parallel algorithms for Hough transform therefore distribute either the binary image or the parameter space, or both to multiple processors. A variety of parallel architectures have been employed for this purpose. These include systolic arrays by Chuang and Li (1985) and Li et al. (1989), meshes by Rosenfeld et al. (1988) and Kannan and Chuang (1990), SIMD trees by Ibrahim et al. (1986), scan line array processors by Fishburn and Highnam (1987), shared memory MIMD machines by Thazhuthaveetil and Shah (1991) and Choudhary and Ponnasamy (1991), and hypercube multicomputers by Ranka and Sahni (1990).

There are several unforeseen overheads in parallel Hough transform implementation. The costs of data distribution and communication in parallel machines may sometimes limit the algorithm performance. Shared memory architectures present the *locking* overheads. In addition, if the pattern distribution within the image is nonhomogeneous, the load distribution between processors is not equal. The speed of computation is then limited by the speed of the processor that has the largest task. This overhead, called the *asynchronism overhead* is one of the important factors limiting the performance of parallel Hough transform algorithms. Partial elimination of asynchronism overhead by shifting loads between neighboring processors is possible (Ranka and Sahni, 1990), but it implies the costs of running a load balancing algorithm and of the additional communication to shift the loads. Most other parallel algorithms in literature fail to eliminate asynchronism overhead and some state that it is impossible without a priori knowledge of the pixels distribution (Thazhuthaveetil and Shah, 1991).

In Section 2 we develop a Distributed Hough Transform Algorithm (DHTA) by partitioning the algorithm, rather than the data space. This algorithm, suitable for distributed memory MIMD multiprocessor architectures, greatly reduces the asynchronism overhead. The results obtained from the implementation of DHTA on different MIMD architectures are described in Section 3. Finally Section 4 presents the conclusions from this work.

## 2. The distributed Hough transform algorithm

We illustrate the Distributed Hough Transform Algorithm (DHTA) for the simple application of straight line detection in a binary image. By using parameters $r$, the distance from the origin, and $\theta$, the angle with the positive $x$-axis, any straight line may be expressed through parameterized equations

$$x = r\cos\theta, \quad \text{and} \quad y = r\sin\theta. \tag{1}$$

By restricting $r$, this parametrization leads to a finite-sized parameter space with a one-to-one correspondence with all possible lines in the image. Under (1), a black image pixel at $(x, y)$ may belong on any line whose parameters satisfy the relation

$$r = x\cos\theta + y\sin\theta. \tag{2}$$

Consequently, the count in each $(r, \theta)$ cell in parameter space satisfying (2) is incremented to indicate that there is possibly a line with these parameters in the image. After the entire image is scanned, the parameter space holds the Hough transform of the image. The cell in the parameter space containing the highest count corresponds to the line to which the maximum number of black pixels fitted.

DHTA separates the task of scanning the binary image from the task of generating the parameter space. It is useful for distributed memory MIMD architectures with many different geometries. DHTA assigns to a set of processors (*image nodes*) the tasks of scanning the image, extracting the coordinates of black pixels and communicating these coordinates to the remaining processors. Since in distributed memory MIMD computers, communication set-up time is significant, DHTA generates messages only after examining successive $(1/Q)$th portions of the image for some appropriate $Q$. Each of these $Q$ messages contains the total number of black pixels found in that region and the coordinate pairs of all those pixels. A second set of processors (*parameter nodes*) are assigned to update the parameter space which is distributed amongst them. The data communication proceeds in a pipeline fashion: when a processor gets a message, it sends a copy to the 'next' processor be-

fore using the data. This algorithm is shown in Figure 1. [1]

Mapping DHTA to a given MIMD architecture requires a one-to-one correspondence between the algorithm nodes and the processors. The number of processors to be used as *images nodes*, $\rho_I$, versus those as *parameter nodes*, $\rho_H$, is governed by the relative complexities of image scanning and parameter space generation tasks. The image scanning task involves finding all the black pixels in the image, and generating and transmitting messages containing their coordinates. Its complexity, $T_I$, depends upon the image size and its average gray level as well as on the communication set-up time. The task of parameter space generation, of complexity $T_H$, involves receiving the messages containing the coordinates, and for each coordinate pair, evaluating (2) for every $\theta$ in the parameter space. The processor partitioning should be such that the average load on each processor is approximately the same, i.e.,

$$(T_H/\rho_H) \approx (T_I/\rho_I) .$$

[1] The 'type' qualifier added to each message of Figure 1 allows each node of the MIMD architecture to distinguish between the messages it receives from the same source.

In our studies, we found that for realistic images and reasonable parameter spaces, $\rho_I \ll \rho_H$.

The mapping of DHTA should also ensure that the source and destination nodes of a node are its topological neighbors so as to minimize the communication overheads. For architectures other than a linear array, a DHTA implementation may be configured as several concurrent pipelines. A DHTA implementation has time complexity

$$T = (T_I/\rho_I)/Q + Q \cdot T_s + (L_{max} - 1) \cdot T_c + (T_H/\rho_H) , \quad (3)$$

where $L_{max}$ is the maximum length of a pipeline, $T_s$, the communication set-up time and $T_c$, the total communication time between neighboring processors. As shown by (3), the time complexity of DHTA increases with the maximum length of a constituent pipeline. Thus within the constraints of the architecture, a mapping should minimize this quantity. Figure 2 shows a typical map of DHTA on a 16-processor hypercube having $L_{max} = 4$ and multiple pipelines {1, 3, 7, 15}, {1, 3, 11}, ..., {4, 12}, {8}. In Figure 2, the 0th processor scans the image and messages to multiple destinations are arranged in a left-to-right order. Thus, from processor 0, a message goes out to processors 1, 2, 4 and 8 in that order. This ensures

**Distributed Hough Transform Algorithm.**

**if** *image node* **then do**

Determine destination nodes

**for** $q = 1$ to $Q$ **do**

examine $(1/Q)$ image, noting the black pixels and creating message

send message of type $q$ to all destinations

**if** *parameter node* **then do**

Determine source and destination nodes

**for** $q = 1$ to $Q$ **do**

receive message of type $q$ from source

send message of type $q$ to all destinations

update parameter space with the coordinates of black pixels in the message

Figure 1. The Distributed Hough Transform Algorithm (DHTA).

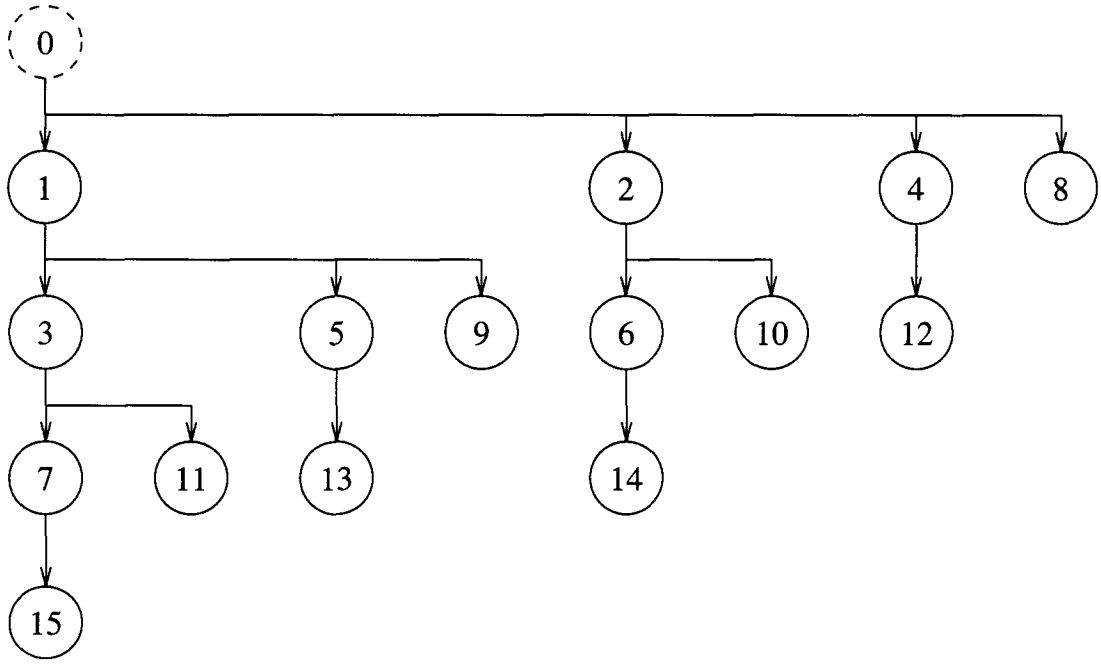


Figure 2. DHTA mapped on a degree-4 hypercube configured as several concurrent pipelines.

that the messages reach longer pipelines earlier than the shorter ones.

The value of $Q$ should also be judiciously chosen based on (3). A small $Q$ implies a large time lag (first term of (3)) before any parameter space updating may begin. On the other hand, a large $Q$ causes a large number of messages, increasing the second term of (3).

Thus the exact implementation of DHTA greatly depends on the topology and the communication/computation characteristics of the available MIMD architecture.

## 3. Implementations and results

This section describes the results of DHTA mappings on a linear array, wrap-around mesh and hypercube topologies all of which were implemented as subsets of the same NCUBE 10 hypercube machine. This ensured that they all would have identical computational and communication characteristics and the results obtained would thus characterize the performance of the chosen topologies.

NCUBE 10 is a multiprocessor MIMD architecture configured as a hypercube. Each processor, equipped with 128K local memory, has a peak performance of 0.5 MFLOPs. The operating system of NCUBE 10 supports time measurement in terms of 'ticks', a tick being approximately equal to 0.171 ms for our 6 MHz system. Communication between processors takes place in a DMA fashion and the time required to move $m$ bytes between neighboring processors is given by

$$T_c = T_s + \tau m ,$$

where $T_s$, the communication set-up time is 4 ticks and $\tau$, the incremental communication time is 0.017 ticks/byte.

A multitude of binary images characterized by parameters *image density* and *image nonuniformity* was used. Image density of a binary image, varied between 5% and 25%, is the fraction of its pixels that are black. Image nonuniformity was intended to describe the distribution of black pixels within the image. Since such a distribution can be varied in a variety of continuous ways, we constrained our experimentation to a somewhat artificial setting in

which both left an right halves of an image may have uniform but different densities. The extra fraction of black pixels in the right half (as compared to the left) was chosen as the measure of nonuniformity. Thus images with 0% nonuniformity have a uniform distribution of black pixels in the complete image, whereas 100% nonuniform images have all their black pixels in the right half and those with $-100\%$ uniformity, have all of them in the left half.

Our experiments consisted of evaluating DHTA on the chosen topologies for both speed and insensitivity to image nonuniformities. The speed improvement of a multiprocessor implementation is characterized by the Speed-Up, $S_P$, defined as

$$S_P = \frac{\text{time complexity using one processor}}{\text{time complexity using } P \text{ processors}}.$$

Speed-up of DHTA on the three architectures is shown in Figures 3–5.

In each case, the number of processors varied between 4 and 32. Exactly one processor was assigned the task of image scanning and the rest, to parameter space updating. The $64\times64$ images (with uniform density) were scanned from left to right and a new message bearing the coordinates of black pixels was generated after every 8 columns were examined.

A desirable characteristic of $S_P$ is $S_P=O(P)$. [2] Such *linear speed-up* implies that the algorithm implementation is *scalable*, i.e., its time complexity may be kept

[2] $S_P$ cannot be $P$ because of the communication and problem partitioning overheads.

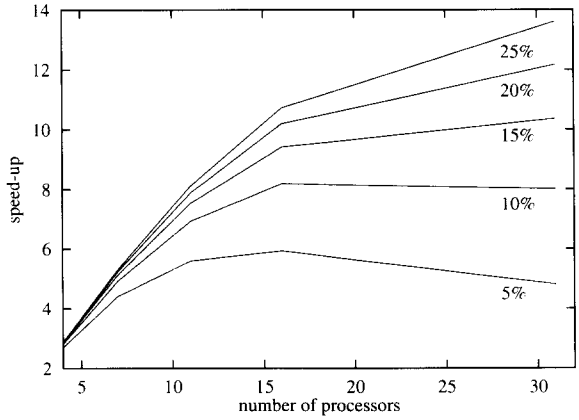


Figure 3. Speed-up on a linear array for various image densities.

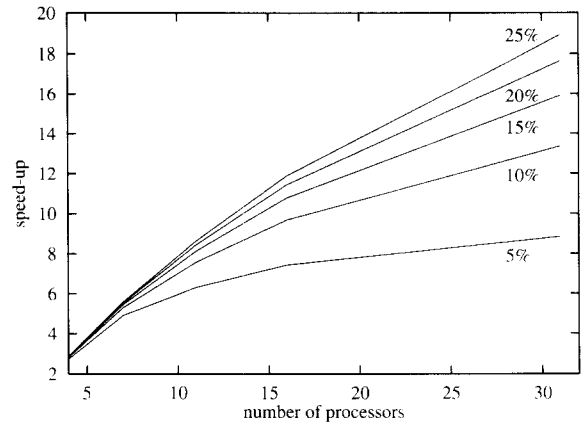


Figure 4. Speed-up on a mesh for various image densities.

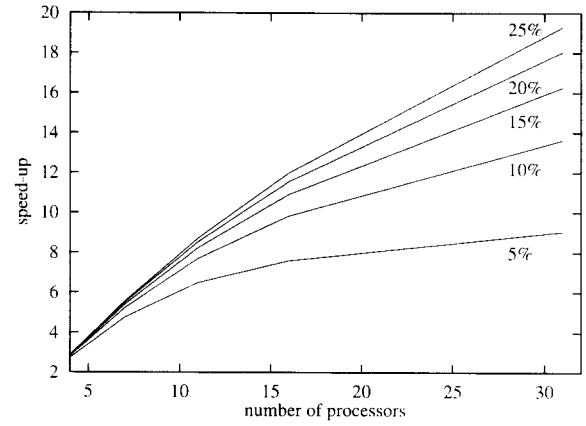


Figure 5. Speed-up on a hypercube for various image densities.

the same even if the problem size is increased by proportionate increase in the size of the architecture.

In case of DHTA whose behavior is modeled by (3), an increase in $P$ keeping $Q$ constant results in a proportionate decrease in $(T_I/\rho_I)$ and in $(T_H/\rho_H)$. The second term of (3) is a constant that becomes negligible in the presence of large $(T_H/\rho_H)$. However, the third term of (3) increases with $P$, and in particular, is proportional to the *diameter* of the architecture. For the linear array, mesh and hypercube architectures, the diameters are $P$, $\sqrt{P}$ and $\log_2 P$, respectively. One can thus conclude that the DHTA implementation on all these three architectures is scalable and provides $O(P)$ speed-up for large problems. This conclusion is supported by the results plotted in Figures 3–5.

One should however note that if the problem size is small and the architecture size is increased unbounded, one may actually get a lower speed-up because the increase in the overheads outpaces the decrease in the computations per processor. Thus, for example, in case of linear array computing the Hough transform of a low density (5%) image, the speed-up actually decreases as $P$ increases beyond 16. For higher densities, the amount of computation is higher and the speed-up curve does not decline till a much higher processor cardinality. In case of both mesh and hypercube, similar declines are not observed (within the range of $P$ plotted) because their diameters are much smaller than a linear array implying a much lower overhead represented by the third term of (3).

The results of Figures 3–5 were obtained with uniform image densities. When nonuniform images were used, these results were negligibly affected. Figures 6 and 7 show the DHTA complexity for nonuniform images. Even though we have presented here results for a 32-processor linear array and hypercube, other architecture sizes and the mesh behave similarly. The insensitivity of DHTA to image nonuniformity can be understood by noting that in our implementation, the average time to create a new message, $((T_I/\rho_I)/Q)$, is chosen to be a little less than the average time to digest that message, $((T_H/\rho_H)/Q)$. This implies a load imbalance as the $\rho_I$ nodes working on the image have a smaller load than $\rho_H$ others working on the parameter space. However, since $\rho_I \ll \rho_H$, the proportion of less efficient nodes is small. On the other hand, a new massage arrives at the $\rho_H$ processors before (an average size) old message is completely digested. Thus even if the old message contained fewer coordinate pairs than average (because of image nonuniformities), the probability of the $\rho_I$ processors becoming idle because of the lack of new coordinate pairs is fairly small.

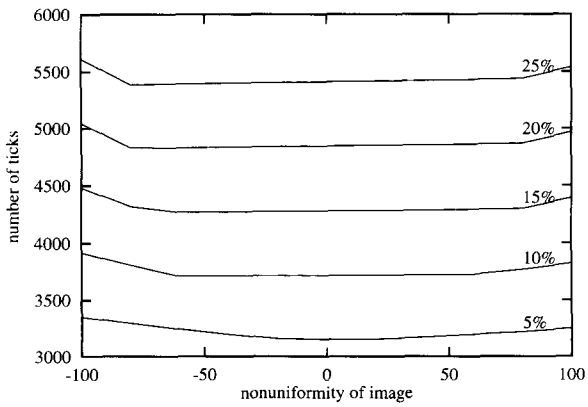


Figure 6. Time complexity of DHTA on a 31-processor linear array for various image densities.

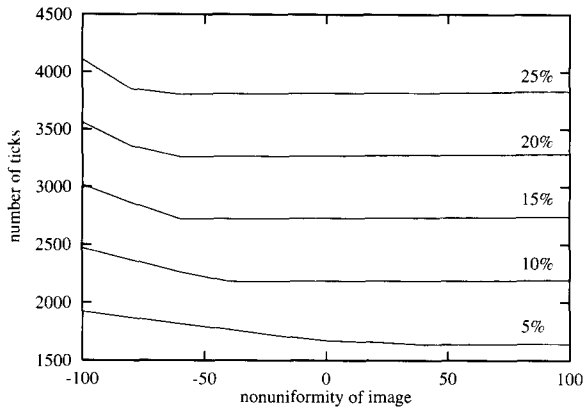


Figure 7. Time complexity of DHTA on a 32-processor hypercube for various image densities.

The results presented in Figures 6 and 7 show that the asynchronism overheads are greatly reduced in DHTA except in extreme image nonuniformities ($-100\%$) which are rare in real images.

## 4. Discussion and conclusion

This correspondence describes the Distributed Hough Transform Algorithm (DHTA) and presents results relating to its implementation on a linear array, a wrap-around mesh and a hypercube. DHTA is based on two principles: separating and distributing the tasks of image examination and parameter space updating, and mapping of these tasks on concurrent pipelines.

DHTA speed-up on standard MIMD architectures is seen to be linear unless the computational task is very small with respect to the number of processors used. In such cases, the task size is smaller than ideal, the overheads increase disproportionately and the speed up declines with increasing number of processors. DHTA implementation uses concurrent pipelines carved out of an architecture and its speed-up is

related to the maximum length of any such pipeline. Therefore, a smaller diameter architecture such as a hypercube provides a much higher speed-up than a linear array with a larger diameter.

An important consequence of distributing the algorithm rather than the data is the reduction of asynchronism overhead in DHTA implementations. This does not require additional tasks of load shuffling between processors; rather, it is achieved by the very nature of DHTA. The load imbalance between $T_I/\rho_I$ and $T_H/\rho_H$ allows one to ensure that the large fraction of processors engaged in parameter space updating are rarely idle even if the image is not uniform. Our results in this respect are promising but more accurate modeling of image nonuniformity and its relation to the 'best' load imbalance still is an open problem which needs to be addressed to make most effective use of DHTA in a practical situation.

## References

Choudhary, A.N. and R. Ponnusamy (1991). Implementation and evaluation of Hough transform algorithms on a shared-memory multiprocessor. *J. Parallel and Distributed Computing* 12 (2), 178–188.

Chuang, H.Y.H. and C.C. Li (1985). A systolic processor for straight line detection by modified Hough transform. *IEEE Workshop on Computer Architecture, Pattern Analysis and Data Base Management*, 300–303.

Fishburn, A. and P. Highnam (1987). Computing the Hough transform on a scan line array processor. *IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, 83–87.

Ibrahim, H., J. Kender and D.E. Shaw (1986). On the application of massively parallel SIMD tree machines to certain intermediate level vision tasks. *Computer Vision, Graphics, and Image Processing* 36 (3), 53–75.

Kannan, C.S. and H.Y.H. Chuang (1990). Fast Hough transform on a mesh connected processor array. *Inform. Process. Lett.* 33 (5), 243–248.

Li, H.F., D. Pao and R. Jayakumar (1989). Improvements and systolic implementation of the Hough transformation for straight line detection. *Pattern Recognition* 22 (6), 697–706.

Ranka, S. and S. Sahni (1990). Computing Hough transforms on hypercube multicomputers. *J. Supercomputing* 4 (0), 169–190.

Rosenfeld, A., J. Ornelas and Y. Hung (1988). Hough transform algorithms for mesh connected SIMD parallel processors. *Computer Vision, Graphics, and Image Processing* 41 (3), 293–305.

Thazhuthaveetil, M.J. and A.V. Shah (1991). Parallel Hough transform algorithm performance. *Image and Vision Computing* 9 (2), 88–92.