An Improved Constant-Time Algorithm for Computing the Radon and Hough Transforms on a Reconfigurable Mesh

Yi Pan, Keqin Li, and Mounir Hamdi

***Abstract*—The Hough transform is an important problem in image processing and computer vision. Recently, an efficient algorithm for computing the Hough transform has been proposed on a reconfigurable array [12]. For a problem with an $\sqrt{N} \times \sqrt{N}$ image and an $n \times n$ parameter space, the algorithm runs in a constant time on a three-dimensional (3-D) $n \times n \times N$ reconfigurable mesh where the data bus is $N^{1/c}$-bit wide. To our best knowledge, this is the most efficient constant-time algorithm for computing the Hough transform on a reconfigurable mesh. In this paper, an improved Hough transform algorithm on a reconfigurable mesh is proposed. For the same problem, our algorithm runs in constant time on a 3-D $n * n \times \sqrt{N} \times \sqrt{N}$ reconfigurable mesh, where the data bus is only $\log N$-bit wide. In most practical situations, $n = O(\sqrt{N})$. Hence, our algorithm requires much less VLSI area to accomplish the same task. In addition, our algorithm can compute the Radon transform (a generalized Hough transform) in $O(1)$ time on the same model, whereas the algorithm in [12] cannot be adapted to computing Radon transform easily.**

***Index Terms*— Hough transform, image processing, reconfigurable mesh, time complexity.**

I. Introduction

The Radon transform of a gray-level image is a set of projections of the image taken from different angles. Specifically, the image is integrated along line contours defined by the equation

$$\{(x, y) : x\cos(\theta) + y\sin(\theta) = \rho\} \tag{1}$$

where

θ	angle of the line with respect to positive x-axis;
ρ	(signed) distance of the line from the origin (see Fig. 1).

This parameterization of line contours is used rather than slope-intercept parameterization so that vertical lines can be represented with finite values of parameters. It is fair to assume that $0 < \theta \leq \pi$, because projections are the same for θ and for $\theta + \pi$ regardless of the value of θ. The original definition of the Radon transform involves a line integral. For digitized pictures, this integral is replaced by a weighted summation. Normally, it is assumed that each line contour is approximated by a band that is one pixel wide. All of the pixels that are centered in a given band will contribute to the value of that band. Because each band is one pixel wide, there are at most $\sqrt{2N}$ values of ρ for each value of θ. In general, each band could be several pixels wide and is determined by R_{res}, the resolution along the ρ direction.

Manuscript received November 1, 1997; revised November 22, 1998. This work was supported in part by the National Science Foundation under Grants CCR-9211621 and CCR-9540901; the Air Force Avionics Laboratory, Wright Laboratory, Dayton, OH, under Grant F33615-C-2218; NASA/University Joint Venture (JOVE) in Research Program of National Aeronautics and Space Administration; Research Foundation of State University of New York; NASA/ASEE Summer Faculty Fellowship Program; and the Hong Kong Research Grant Council under Grant RGC/HKUST 619/94E.

Y. Pan is with the Department of Computer Science, University of Dayton, Dayton, OH 45469-2160 USA (e-mail: pan@cps.udayton.edu).

K. Li is with the Department of Mathematics and Computer Science, State University of New York, New Paltz, NY 12561-2499 USA.

M. Hamdi is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.

Publisher Item Identifier S 1083-4427(99)03327-5.

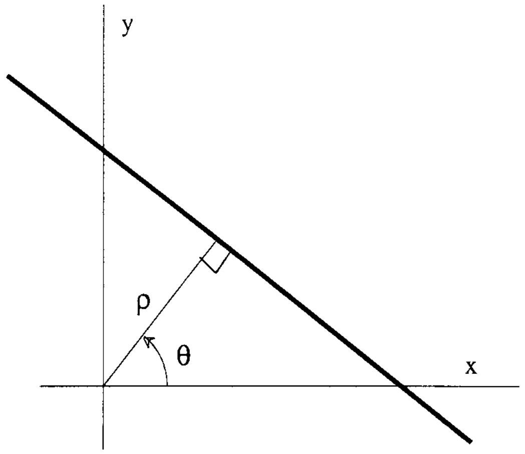

Fig. 1. The normal parameters for a line.

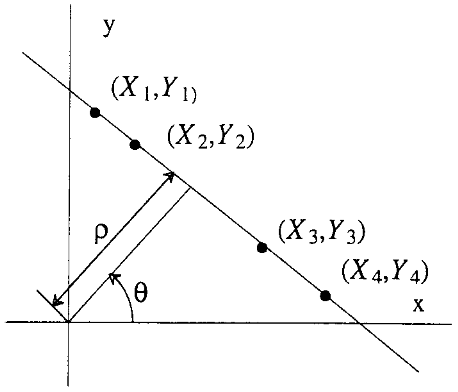

Fig. 2. Projection of collinear points onto a line.

The parameter n will be used to denote the number of projections (values of θ) that are to be calculated.

The Hough transform is just a particular case of the Radon transform, where the input image is binary. Both transforms have many uses in computer vision and image processing [26]. The Hough transform is an important curve-detection method based on a relation between points lying on a line or curve in the image space and the parameters of that line or curve [3], [6]. Under this transformation, collinear edge pixels (feature pixels) are mapped into the same point in the parameter space, as illustrated in Fig. 2. If the parameter space is quantized into n θ values, $\theta_0, \ldots, \theta_{n-1}$, and n ρ values, $\rho_0, \ldots, \rho_{n-1}$, a (θ, ρ) pair would correspond to a linear band of edge pixels, approximating a line. Thus, detecting lines reduces to detecting points in the parameter space to which a large number of pixels are mapped. Obviously, detecting (θ, ρ) pairs corresponding to lines is easier than detecting lines directly in the image space.

The computation of the Radon and Hough transforms on a sequential computer can be described as follows. We use an $n \times n$ array to store the counters which are initialized to zero. For each of the N pixels in an $\sqrt{N} \times \sqrt{N}$ image and for each of the n values of θ, the value of ρ is computed based on (1) and the sum corresponding to the particular (θ, ρ) is accumulated as given in the following algorithm.

In the algorithm, ρ_{res} is the resolution along the ρ direction; and gray-value(x, y) is the intensity of the pixel at location (x, y)

```
for each pixel at location $(x, y)$ in an image do
  for $\theta = \theta_0, \theta_1, \ldots, \theta_{n-1}$ do
  begin
     (* parameter computation *)
   $\rho := (x \cos\theta + y \sin\theta)/\rho_{\text{res}}$
     (* accumulation *)
   sum$[\theta, \rho]$ := sum$[\theta, \rho]$ + gray-value$(x, y)$
  end.
```

Obviously, for an $\sqrt{N} \times \sqrt{N}$ image, and n values of θ, a sequential computer calculates the Radon (Hough) transform in $O(Nn)$ time. The computation time is too long for many applications, especially for real-time applications, as N and n can be very large.

Since most image processing problems such as the Radon and Hough transforms require real-time computation, fast parallel computation for these problems is the natural way. Many parallel image processing algorithms and their VLSI implementations have been proposed in the literature [12], [17]–[19]. In particular, a number of parallel Hough transform algorithms have been designed for different parallel architectures, including tree [7], systolic array [1], [16], line array [4], pyramid [9], mesh [2], [5], [10], [22], [27], [30], reconfigurable mesh [8], and hypercube [23].

Recently, several constant-time algorithms for computing the Hough transform have been proposed for the reconfigurable mesh model [11], [24]. In the above algorithms, the data buses in the reconfigurable mesh model are $O(\log N)$-bit wide. More recently, Kao *et al.* [12] improved these constant-time algorithms by using fewer processors. Their algorithm runs in a constant time on a three-dimensional (3-D) $n \times n \times N'$ reconfigurable mesh where N' is the number of edge pixels and the data bus is $N'^{1/c}$-bit wide. Here c is a constant and $c \geq 1$. In their algorithm, a feature extraction algorithm is needed to process the binary image to extract edge (feature) pixels. Clearly, in the worst case, an image may contain all feature pixels; that is, $N' = O(N)$. Hence, the number of processors used in their algorithm is $n * n * N$. To our best knowledge, this is the most efficient constant-time algorithm for computing the Hough transform on a reconfigurable mesh.

In this paper, we will propose a new constant time algorithm for computing the Radon and Hough transforms on a reconfigurable mesh. The purpose of this paper is to show that the bus width of the reconfigurable mesh can be reduced while achieving the same time complexity and using the same number of processors as in [12]. Because bus width occupies VLSI area, our algorithm represents an improvement over that of Kao *et al* [12]. In addition to the above advantage, our algorithm can also compute the Radon transform (a generalized Hough transform) in $O(1)$ using the same number of processors, whereas the algorithm in [12] cannot be adapted to computing Radon transform easily.

II. The Reconfigurable Mesh Model

A reconfigurable mesh consists of a bus in the shape of a mesh which connects a set of processors, but which can be split dynamically by local switches at each processor. By setting these switches, the processors partition the bus into a number of sub-buses through which the processors can then communicate. Thus the communication pattern between processors is flexible, and moreover, it can be adjusted during the execution of an algorithm. There are many reconfigurable parallel processing systems such as the bus automaton [28], [29], the reconfigurable mesh [20], the polymorphic-torus network [13], the processor arrays with reconfigurable bus system (abbreviated to PARBS) [32], and the cross-bridge reconfigurable array of processors (abbreviated to CRAPS) [12] which are all functionally equivalent.

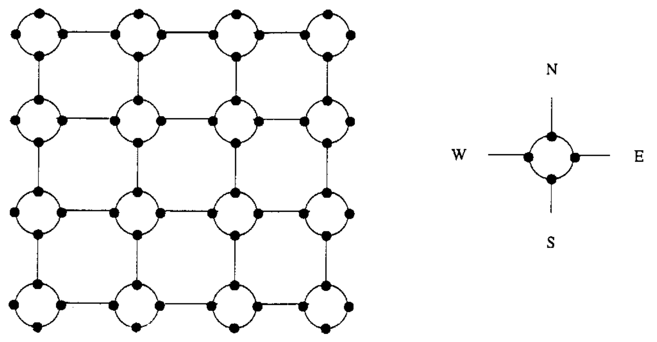

Fig. 3. Two–dimensional 4 × 4 reconfigurable mesh.

The reconfigurable mesh has begun to receive a great deal of attention as both a practical machine to build, and a good theoretical model of parallel computation. On the practical side, reconfigurable meshes have been constructed for use in vision and VLSI applications [14]. On the theoretical side, the reconfigurable mesh has been shown to be able to solve some fundamental theoretical problems very quickly. In fact, the reconfigurable mesh can solve some problems faster than the PRAM, the model most often used for theoretical parallel computation. For instance, an N processor reconfigurable mesh can find the parity of N bits in constant time [15], while an N processor PRAM requires $\Omega(\log N/\log\log N)$ time.

A two-dimensional (2-D) reconfigurable mesh consists of an $N_1 \times N_2$ array of processors which are connected to a grid-shaped reconfigurable bus system. Each processor can perform arithmetic and logical operations and is identified by a unique index (i, j), $0 \leq i < N_1$, $0 \leq j < N_2$. The processor with index (i, j) is denoted by *PE*(i, j). The configuration of the bus system is dynamically changeable by adjusting the local connection among ports within each processor. Only one processor is allowed to broadcast a value within the same sub-bus. Each processor can communicate with other processors by broadcasting values on the bus system. We assume that the bus width is $O(\log N)$ and each broadcast takes $O(1)$ time. The arithmetic operations in the processors are performed on $O(\log N)$ bit words. Hence, each processor can perform one logical and arithmetic operation on $O(1)$ words in unit time. In Fig. 3, we depict a 4 × 4 reconfigurable mesh with a global bus configuration.

A high-dimensional reconfigurable mesh can be defined similarly. For example, a processor in a 3-D $N_1 \times N_2 \times N_3$ reconfigurable mesh is identified by a unique index (i, j, k), $0 \leq i < N_1$, $0 \leq j < N_2$, $0 \leq k < N_3$. The processor with index (i, j, k) is denoted by *PE*(i, j, k). Within each processor, six ports are built with every two ports for each of the three directions: i-direction, j-direction, and k-direction. In each direction, a single bus or several sub-buses can be established.

A subarray is denoted by replacing certain indices by $*$'s. For example, the ith row of processors in a 2-D reconfigurable mesh is represented by *ARR*$(i, *)$. Similarly, *ARR*$(*, j, k)$, $0 \leq j < N_2$, $0 \leq k < N_3$, is a one-dimensional (1-D) subarray in a 3-D reconfigurable mesh, and these $j \times k$ subarrays can execute algorithms independently and concurrently. Finally, a memory location L in *PE*(i, j, k) is denoted as $L(i, j, k)$.

III. The Constant-Time Algorithm

In this section, we propose a constant time algorithm for computing the Radon and Hough transforms of an $\sqrt{N} \times \sqrt{N}$ image. In the following discussion, we partition the image into parallel bands and these bands run at an angle of θ with respect to the horizontal axis, and then sum the pixel values contained in each band. If a pixel

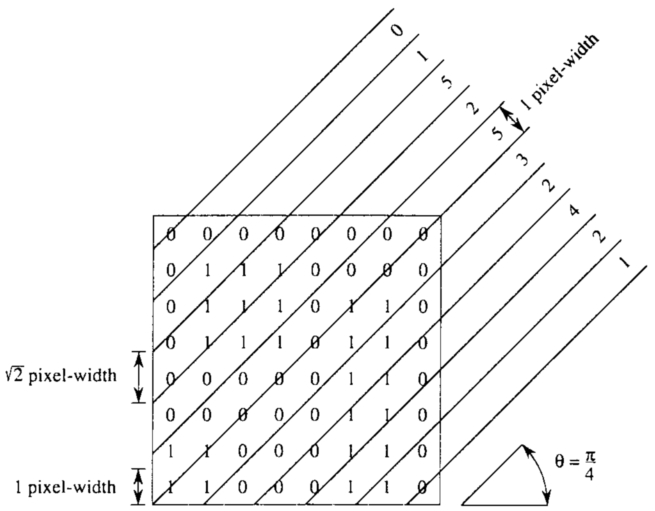

Fig. 4. Parallel bands for $\theta = \pi/4$ in an 8×8 image.

is contained in two or more bands, then it will be counted in only the band that contains its center. If the center of a pixel lies on the boundary between two bands, then it is counted only in the uppermost of the two bands. For example, we have computed a $\pi/4$ angle Hough transform for an 8×8 pixel array in Fig. 4, where the bands are one pixel-width wide. Clearly, there are ten different ρ's in the Fig. 4. In Fig. 4, the number of one-pixels contained in each band is displayed at the upper right end of the band. For the Radon transform, the only difference is that pixels are no longer black and white pixels and the sum is to accumulate the grey values of the pixels contained in each band. For a particular angle θ and a particular distance ρ, only the values of the pixels lying in the band specified by θ and ρ need to be added together.

Many algorithms previously proposed [12], [22], [24] use edge pixels as the inputs. There are three problems with that assumption.

1) Generating edge pixels needs preprocessing and normally takes more than $O(1)$ time.
2) It is impossible to exploit the geometric features and relations of pixels in an image.
3) The number of edge pixels differs for different images.

Hence, we need arrays of different sizes for different images. This makes their algorithm unscalable and is certainly not very practical for image processing. In our algorithm, all pixels in an image are used as the input. Hence, we can exploit easily the geometric features and relations of pixels in an image. Clearly, for a given pair of ρ and θ, we do not need to consider all the pixels in an image. Instead, only those pixels that are centered in the band will contribute to the value of that band. In this way, we can improve the efficiency of the algorithm during computation.

Let $\rho_{x,y,k}$ denote the value computed by the pixel at location (x, y) with angle θ_k in the (ρ, θ) plane

$$\rho_{x,y,k} = (x \cos\theta_k + y \sin\theta_k)/\rho_{\text{res}} \tag{2}$$

where R_{res} is the resolution along the ρ direction. Assume that the reconfigurable mesh used here is configured as a 3-D $n * n \times \sqrt{N} \times \sqrt{N}$ array.

By a suitable transformation, we can map $\rho_{x,y,k}$ and θ_k into the range $[0, n-1]$. Thus, we have the following relation for the n quantized θ values, i.e., $\theta_i = i$ for all $0 \le i < n$. Similarly, the following relation holds for the n quantized ρ values, i.e., $\rho_i = i$ for all $0 \le i < n$. Let $C(n * s + t, 0, 0)$ denote the weighted sum of all pixels of the band specified by the angle θ_s and the distance ρ_t. In the case of 0-1 pixel values, $C(n * s + t, 0, 0)$ $(0 \le s, t < n)$ is the Hough transform and its value represents the number of pixels on the line defined by the (θ_s, ρ_t) pair. The main idea of our algorithm is to compute the Radon and Hough transforms of a single image for different angles and different distances on different subarrays. By mapping the parameter space into different subarrays, we can calculate the sum of pixel values in each band independently. Since a summation operation can be carried out efficiently on a subarray, we can calculate the Radon transform quickly. In our algorithm, we also assume that the range of gray values for the Radon transform can be represented in $\log n$ bits. Also we assume that $0 \le \theta \le \pi/4$ in the following discussion. For other θ values, the projections are calculated the same way. In the following, we formally describe the algorithm.

Algorithm:

Input: An $\sqrt{N} \times \sqrt{N}$ image and an $n \times n$ parameter space, where ρ_{res} is the resolution along the ρ direction. Assume that each pixel value $a(x, y)$ is stored in processors $PE(0, x, y)$, for $0 \le x, y < \sqrt{N}$, and each θ_k and ρ_{res} are stored in processors $PE(k, 0, 0)$, for $0 \le k < n$, respectively. The algorithm consists of eight steps.

Step 1: All processors $PE(0, j, k)$, $0 \le j < \sqrt{N}$, $0 \le k < \sqrt{N}$, broadcast the image pixels $a(j, k)$ concurrently through its sub-buses in direction i such that processors $PE(i, j, k)$, $0 \le i < n * n$, each receives a pixel from $PE(0, j, k)$. At the end of step 1, all processors in subarray $ARR(*, j, k)$ where $0 \le j < \sqrt{N}$, $0 \le k < \sqrt{N}$, contain the pixel value $a(j, k)$ at location (j, k) in the original image.

Step 2: Every processor calculates its θ value and ρ value. In our algorithm, $ARR(s * n + t, *, *)$, $0 \le s, t < n$, is responsible for accumulating the pixel values in the band specified by θ_s, ρ_t. First, $PE(i, j, k)$ calculates the s and t values based on its own processor index i in the array. Then, we can easily calculate the values of θ_s and ρ_t since we already know the resolutions of θ and ρ.

Step 3: $PE(i, j, k)$ checks if its local pixel (the pixel at location (j, k)) belongs to the band owned by the parameter pair θ_s and ρ_t using (1). If yes, mark itself in a variable, i.e., $BELONG(i, j, k) = 1$. Otherwise, $BELONG(i, j, k) = 0$.

Step 4: In this step, we select the processor (i, j, k), whose j index in the column k is the largest and whose local pixel [the pixel at location (j, k)] belongs to the band owned by the parameter pair θ_s and ρ_t. This is done by setting $BELONG(i, j, k) = 2$ if $BELONG(i, j+1, k) = 0$ and $BELONG(i, j-1, k) = 1$.

Step 5: Sum the pixel values, which belong to a band owned by the parameter pair θ_s and ρ_t, in the same column of the image. Depending on the resolution ρ_{res}, several pixel values in a column of the image may belong to a band owned by a parameter pair. Hence, to add these pixel values, we need several iterations. This step is done as follows. In each iteration, there are three cases:

1) $PE(i, j, k)$, whose $BELONG(i, j, k) = 1$, checks its neighboring processor along dimension j to see if $BELONG(i, j + 1, k) = 2$. If yes, add *gray-value*$(i, j+1, k)$ to *gray-value*(i, j, k). Then, $PE(i, j, k)$ sets $BELONG(i, j, k) = 2$. Otherwise, do nothing;
2) $PE(i, j, k)$, whose $BELONG(i, j, k) = 2$, checks its neighboring processor along dimension j to see if $BELONG(i, j - 1, k) = 1$. If so, sets its $BELONG(i, j, k) = 0$ at the end of an iteration. Otherwise, do nothing and go to step 6;
3) $PE(i, j, k)$, whose $BELONG(i, j, k) = 0$, is idle. The above iteration continues until only one processor contains the sum of the pixel values in the same column. The final sum is in a processor whose $BELONG(i, j, k) = 2$.

Step 6: *PE*(i, j, k), whose *BELONG*$(i, j, k) = 2$, sends *gray-value*(i, j, k) to *gray-value*$(i, 0, k)$. In other words, the sum of pixel values in a column moves to the first processor in the same column.

Step 7: Processors except the first processor in a column, i.e., processors (i, j, k), $0 \le i < n*n$, $1 \le j < \sqrt{N}$, and $0 \le k < \sqrt{N}$, set their local variable *gray-value*$(i, j, k) = 0$. At this stage, processor $(i, 0, k)$ contains the sum of the pixel values in column k which belong to the band owned by θ_s and ρ_t, where $i = s*n + t$, $0 \le i < n*n$, and $0 \le s, t < n$.

Step 8: Perform a summation of C across each 2-D subarray *ARR*$(i, *, *)$, $0 \le i < n*n$, concurrently and store the results in $C(i, 0, 0)$. That is

$$C(i,0,0) = \sum_{j=0}^{\sqrt{N}-1} \sum_{k=0}^{\sqrt{N}-1} \textit{gray-value}(i,j,k), \qquad \text{for all } 0 \le j, \quad k < \sqrt{N}.$$

Notice that each subarray *ARR*$(i, *, *)$ is a 2-D $\sqrt{N} \times \sqrt{N}$ reconfigurable mesh, and initially only the first rows *ARR*$(i, 0, *)$, $0 \le i < n*n$, of these subarray contains a sequence of data values of length $\sqrt{N}$. Notice also that each data value is a binary number with at most $O(\log N)$ bits.

The summation can be performed as follows. Broadcast $\sqrt{N}$ data using column processors, such that processor $(i, l \times \log^2 N, k)$ has the lth bit of the data in *PE*$(i, 0, k)$, where $0 \le l \le O(\log N)$. We then divide the subarray *ARR*$(i, *, *)$ into row band subarray with each having $\log^2 N$ rows. Every row band subarray counts the number of 1's in constant time using the addition algorithm for binary numbers described in [21]. Then, we add $\log(\sqrt{N})$ $(\log N)$-bit numbers together using the algorithm in [25].

The result is the Radon transform, i.e., $C(i, 0, 0)$, where $i = n*s + t$, contains the sum of pixel values contained in the band defined by the pair (θ_s, ρ_t). When the image contains only black and white pixels, it is a Hough transform, and $C(i, 0, 0)$, where $i = n*s + t$, contains the number of pixels lying on a line in the image space specified by the pair (θ_s, ρ_t). Hence, line detection is an easy job now since we can select those $C(i, 0, 0)$, $0 \le i < n*n$, which have a large number.

In the following, we calculate the time complexity of the algorithm. Clearly, steps 1 and 6 use broadcast operation only and thus each takes a constant time based on the assumption in [20]. Steps 2–4 and 7 perform some local operations and require a constant time. Each iteration in step 5 needs data communication only between neighboring processors, hence requires $O(1)$ time. Since $n = O(\sqrt{N})$, each column contains a constant number of pixels which belong to the same band. Hence, only a constant number of iterations are needed. In other words, step 5 takes $O(1)$ time.

Now, let us figure out the time used in step 8. The idea is the same as that used in [25]. The broadcast operation takes a constant time. The binary number addition in each row band subarray requires a constant time. It has been shown that the addition of N $O(\log N)$-bit numbers can be performed in $O(1)$ time on a $\log^3 N \times N$ reconfigurable mesh. Hence, adding $\log(\sqrt{N})$ $(\log N)$-bit numbers together on a subarray *ARR*$(i, *, *)$ of size $\sqrt{N} \times \sqrt{N}$ requires a constant time. All subarrays *ARR*$(i, *, *)$, $0 \le i < n*n$, perform the summation concurrently. Thus, step 8 uses $O(1)$ time.

Summarize the above analysis, we can show the following result.

Theorem 1: For a problem with an $\sqrt{N} \times \sqrt{N}$ image and an $n \times n$ parameter space, the algorithm described above correctly computes the Radon and Hough transforms, and runs in constant time on a 3-D $n*n \times \sqrt{N} \times \sqrt{N}$ reconfigurable mesh, where the data bus is only $\log N$-bit wide.

IV. Conclusion

In this paper, a constant time parallel algorithm for computing the Radon and Hough transforms has been proposed on a 3-D reconfigurable mesh with $n*n \times \sqrt{N} \times \sqrt{N}$ processors, where the bus width is only $O(\log N)$ bits. Compared to the algorithm of Kao *et al*. [12], the two algorithms use the same number of processors and require the same time. The only difference is the bus width used in the two algorithms. In other words, both algorithms could be implemented in VLSI using the same amount of chip area if bus width is not considered. However, when bus width is considered, our algorithm is better. In our model, buses of $O(\log N)$ bits are enough to achieve $O(1)$ time algorithm, while the algorithm of Kao *et al*. requires the use of buses of $O(N^{1/c})$ bits, where c is a constant and $c \ge 1$. Clearly, the bus width used in their model is wider than that of ours. Since most reconfigurable meshes are implemented in VLSI, wider bus width implies more VLSI area used. In fact, the problem of layout for wide buses in a higher dimensional reconfigurable mesh is more severe than in a 2-D mesh since we have to map them onto a 2-D VLSI area [31]. Thus, reducing the bus width is very important when implementing a reconfigurable mesh using VLSI technology.

Since the sequential algorithm requires $O(nN)$ time, our algorithm presented in this paper is still not optimal based on the measure of processor-time product. It would be interesting to see if we could reduce the number of processors to nN and still have a constant time complexity. Both the algorithm of Kao *et al*. [12] and the algorithm proposed in this paper use a 3-D reconfigurable mesh. Since higher dimensional arrays require more VLSI layout areas than a lower dimensional arrays even though they use the same number of processors, it would also be interesting to see if a 2-D reconfigurable mesh with the same number of processors could be used to achieve the same result.

Acknowledgment

The authors would like to thank K. R. Pattipati for his professional and timely handling of our submission. Thanks also go to the three anonymous referees for their valuable comments and suggestions.

References

[1] H. Y. H. Chuang and C. C. Li, "A systolic array processor for straight line detection by modified Hough transform," in *Proc. IEEE Computer Society Workshop Computer Architecture Pattern Analysis Image Database Management*, Nov. 1985, pp. 300–304.

[2] R. E. Cypher, J. L. C. Sanz, and L. Snyder, "The Hough transform has $O(N)$ complexity on SIMD N by N mesh array architectures," in *Proc. IEEE Computer Society Workshop Computer Architecture Pattern Analysis Machine Intelligence*, Oct. 1987, pp. 115–121.

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

[4] A. L. Fisher and P. T. Highnam, "Computing the Hough transform on a scan line array processor," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, pp. 262–265, Mar. 1989.

[5] C. Guerra and S. Hambrusch, "Parallel algorithms for line detection on a mesh," *J. Parallel Distrib. Comput.*, vol. 6, pp. 1–19, Feb. 1989.

[6] P. V. C. Hough, "Methods and means to recognize complex patterns," U.S. Patent 3 069 654, 1962.

[7] H. A. H. Ibrahim, J. R. Kender, and D. E. Shaw, "The analysis and performance of two middle-level vision tasks on a fine-grained SIMD tree machine," in *Proc. IEEE Computer Society Conf. Computer Vision Pattern Recognition*, June 1985, pp. 387–393.

[8] J. F. Jeng and S. Sahni, "Reconfigurable mesh algorithms for the Hough transform," in *Int. Conf. Parallel Processing*, Aug. 12–16, 1991, vol. III, pp. 34–41.

[9] J. Jolion and A. Rosenfeld, "An $O(\log n)$ pyramid Hough transform," *Pattern Recognit. Lett.*, vol. 9, pp. 343–349, 1989.

[10] C. S. Kannan and H. Y. H. Chuang, "Fast Hough transform on a mesh connected processor array," *Inf. Process. Lett.*, vol. 33, pp. 243–248, 1990.

[11] T. W. Kao, S. J. Horng, Y. L. Wang, and K. L. Chung, "A constant time algorithm for computing Hough transform," *Pattern Recognit.*, vol. 26, no. 2, pp. 277–286, 1993.
[12] T.-W. Kao, S.-J. Horng, and Y.-L. Wang, "An $O(1)$ time algorithm for computing histogram and Hough transform on a cross-bridge reconfigurable array of processors," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 681–687, Apr. 1995.
[13] H. Li and M. Maresca, "Polymorphic-torus network," *IEEE Trans. Comput.*, vol. 38, pp. 1345–1351, Sept. 1989.
[14] H. Li and Q. F. Stout, *Reconfigurable Massively Parallel Computers*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[15] ——, "Reconfigurable SIMD massively parallel computers," *Proc. IEEE*, vol. 79, pp. 429–433, 1991.
[16] H. F. Li, D. Pao, and R. Jayakumar, "Improvements and systolic implementation of the Hough transformation for straight line detection," *Pattern Recognit.*, vol. 22, no. 6, pp. 697–706, 1989.
[17] J. El Mesbahi, "$O(1)$ algorithm for image component labeling in a mesh connected computer," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, pp. 427–433, Apr. 1991.
[18] ——, "Regional projection contour transform algorithm and its VLSI implementation," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 87–99, Jan. 1994.
[19] ——, "$O(1)$ time quadtree algorithm and its application for image geometric properties on a mesh connected computer," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 1640–1648, Dec. 1995.
[20] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, "Meshes with reconfigurable buses," in *MIT Conf. Advanced Research VLSI*, 1988, pp. 163–178.
[21] K. Nakano, T. Masuzawa, and N. Tokura, "A sub-logarithmic time sorting algorithm on a reconfigurable array," *IEICE Trans.*, vol. E-74, pp. 3894–3901, Nov. 1991.
[22] Y. Pan and H. Y. H. Chuang, "Faster line detection algorithms on enhanced mesh connected arrays," *Proc. Inst. Elect. Eng.*, vol. 140, pp. 95–100, Mar. 1993.
[23] ——, "Parallel Hough transform algorithms on SIMD hypercube arrays," in *1990 Int. Conf. Parallel Processing*, St. Charles, IL, Aug. 13–17, pp. 83–86.
[24] Y. Pan, "A more efficient constant time algorithm for computing the Hough transform," *Parallel Process. Lett.*, vol. 4, no. 1–2, pp. 45–51, 1994.
[25] H. Park, H. J. Kim, and V. K. Prasanna, "An $O(1)$ time optimal algorithm for multiplying matrices on reconfigurable mesh," *Inf. Process. Lett.*, vol. 47, pp. 109–113, Aug. 1993.
[26] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*. New York: Academic, 1982.
[27] A. Rosenfeld, J. Ornelas, and Y. Hung, "Hough transform algorithms for mesh-connected SIMD parallel processors," *Comput. Vis., Graph. Image Process.*, vol. 41, pp. 293–305, 1988.
[28] J. Rothstein, "On the ultimate limitations of parallel processing," in *Proc. Int. Conf. Parallel Processing*, 1976, pp. 206–212.
[29] ——, "Bus automata, brains, and mental models," *IEEE Trans. Syst., Man, Cybern.*, vol. 18, no. 4, pp. 522–531, 1988.
[30] T. M. Silberberg, "The Hough transform on geometric arithmetic parallel processor," in *Proc. IEEE Computer Society Workshop Computer Architecture Pattern Analysis Image Database Management*, Nov. 1985, pp. 387–393.
[31] A. Vaidyanathan and J. L. Trahan, "Optimal simulation of multidimensional reconfigurable meshes by two-dimensional reconfigurable meshes," *Inf. Process. Lett.*, pp. 267–272, Oct. 1993.
[32] B. F. Wang, G. H. Chen, and F. C. Lin, "Constant time sorting on a processor array with a reconfigurable bus system," *Inf. Process. Lett.*, vol. 34, pp. 187–192, Apr. 1990.

Distributed Logic Processors Trained Under Constraints Using Stochastic Approximation Techniques

Kaddour Najim and Enso Ikonen

***Abstract*— This paper concerns the estimation under constraints of the parameters of distributed logic processors (DLP). This optimization problem under constraints is solved using stochastic approximation techniques. DLP's are fuzzy neural networks capable of representing nonlinear functions. They consist of several logic processors, each of which performs a logical fuzzy mapping. A simulation example, using data collected from an industrial fluidized bed combustor, illustrates the feasibility and the performance of this training algorithm.**

***Index Terms*—Fuzzy models, parameter estimation, power plants.**

I. Introduction

The needs for process models arise from various requirements. Different properties are required from the model depending on its application: process design, optimization, control, monitoring, fault detection, operator training, etc. When humans are expected to interact with the model, transparency of a model becomes an important issue. Compared to other basis function approaches, fuzzy models provide transparency to experimental nonlinear modeling, due to their rule-based format. In Sugeno type of fuzzy models [1], the consequent part of a rule is a function of the model inputs; in a 0-order Sugeno model the consequent is simply a constant. A distributed logic processor (DLP) model [2] is built using several logic processors (LP), resulting to a kind of 0-order Sugeno model. However, the rule bases in a DLP model, modeled by separate LP's, are parameterized. This is a major difference compared to many of the other neuro-fuzzy modeling approaches, where the parameters control the shape and location of the fuzzy partitioning (data base). By concentrating to the rule base, the concepts modeled via fuzzy sets are not changed in the adaptation. Hence the familiarity of the concepts is not degenerated, but emphasis is in finding the logical relationships between the concepts.

A DLP model is trained by estimating the parameters of each LP separately. The training task of an LP is formulated as the minimization of some quadratic cost function. Gradient-based "neural" training methods can be used, since the chain-rule can be applied for computing the required gradients. Several training methods have been investigated:

1) simple gradient descent with a constant or adaptive step size [2]–[4];
2) recursive prediction error method and its modifications [5];
3) random search based on games of stochastic learning automata [6].

All these studies have been concerned with the estimation of the model parameters for unconstrained systems. However, the behavior

Manuscript received July 7, 1998; revised February 5, 1999. This work was supported by the Human Capital and Mobility Programme of the Commission of the European Communities, the Academy of Finland under Project 40481, and the COSY program of the European Science Foundation.

K. Najim is with the Process Control Laboratory, Ecole Nationale Supérieure d'Ingénieurs de Génie Chimique. F-31078, Toulouse Cedex, France.

E. Ikonen is with the Systems Engineering Laboratory, Infotech Oulu and Department of Process Engineering, University of Oulu, FIN-90401 Oulu, Finland.

Publisher Item Identifier S 1083-4427(99)05339-4.