

# Pyramid Simulation of Image Processing Applications

*S. Shoari, A. Kavianpour, and N. Bagherzadeh*  
University of California  
Irvine, CA 92717

e-mail: simin@ece.uci.edu

## Abstract

Detecting lines in images using pyramid architecture is the main subject of this paper. The approach is based on the parallel calculation of Hough Transform. A pyramid architecture of size  $n$  is a fine-grain architecture with a mesh base of size  $\sqrt{n} * \sqrt{n}$  processors each holding a single pixel of the image. The pyramid operates in an SIMD mode. Two algorithms for computing the Hough Transform are explained. The first algorithm initially uses different angles,  $\theta_j$ 's, and its complexity is  $O(k + \log_2 n)$  with  $O(m)$  storage requirement. The second algorithm computes the Hough Transform in a pipeline fashion for each angle  $\theta_j$  at a time. This method produces results in  $O(k * \log_2 n)$  time with  $O(1)$  storage, where  $k$  is the number of  $\theta_j$  angles,  $m$  is the number of  $\rho_i$  normal distances from the origin, and  $n$  is the number of pixels. A simulation program is also described.

**Key Words:** Hough transform, image processing, parallel processing, pyramid architecture, simulation.

## 1 Introduction

Parallel computing for image processing has recently received considerable attentions [1, 2, 4, 12, 13, 14, 15, 16, 19]. Technological advances have made the design of fine-grain architectures possible and many practical algorithms have been implemented. The Hough Transform [8] is a powerful tool in shape

analysis. It extracts global features from images; however, because of its computational complexity, it is not easily implementable in real-time for some applications. One approach for achieving real-time implementation of the Hough Transform is by parallel processing. Various configurations have been suggested for parallel implementation of the Hough Transform. A Hough Transform algorithm on a fine-grain SIMD machine with broadcasting capability has been proposed by Ibrahim [7]. Recently several efforts have been made to speed up the computation of the Hough Transform by utilizing parallelism. Systolic arrays for regular and modified Hough Transform have been proposed by Chuang and Li [3]. An  $O(k\sqrt{n})$  time Hough Transform algorithm for  $n$  pixels is given by Silberberg [18] where  $k$  is the number of angles used for the calculation. A number of more efficient Hough Transform algorithms with  $O(k + \sqrt{n})$  time complexity have been suggested [4]. Algorithms on mesh connected arrays have been described by Kannan and Chuang [10]. An algorithm for the Hough Transform calculation on a linear array has been considered by Fisher and Highnam [6], but because of the communication latency between remote processors it takes as much as the serial version of the algorithm to complete. Blanford explains a generalized Hough Transform [1]; the method is similar to the standard Hough Transform except that the image is divided into circular, parabolic, or elliptical bands, as opposed to straight lines. Little et al. [11] described a possible implementation of the Hough Transform on the Connection Machine.

## 2 Pyramid Architecture

A pyramid computer with the base size of  $n$  is an SIMD machine that can be viewed as being constructed from  $(1/2)\log_2 n + 1$  levels of a two-dimensional mesh connected processor array, where the  $L$ th level,  $0 \leq L \leq (1/2)\log_2 n$ , is a two-dimensional mesh connected processor array of size  $n/(4^L)$ . A mesh connected computer of size  $n$  is a collection of  $n$  processing elements arranged in a  $\sqrt{n} * \sqrt{n}$  grid, where each processing element except for those along the border, is connected to its four neighbors. Each processing element at level  $L$  is connected to its neighbors at level  $L$  and four children at level  $L - 1$  ( $L > 0$ ) and a parent at level  $(L + 1)$ , ( $L < (1/2)\log_2 n$ ). Thus, each internal processing element has nine connections. Figure 1 illustrates a pyramid with a 4 by 4 base configuration. Circles represent processing elements and lines represent communication paths. All of the processing elements in the pyramid

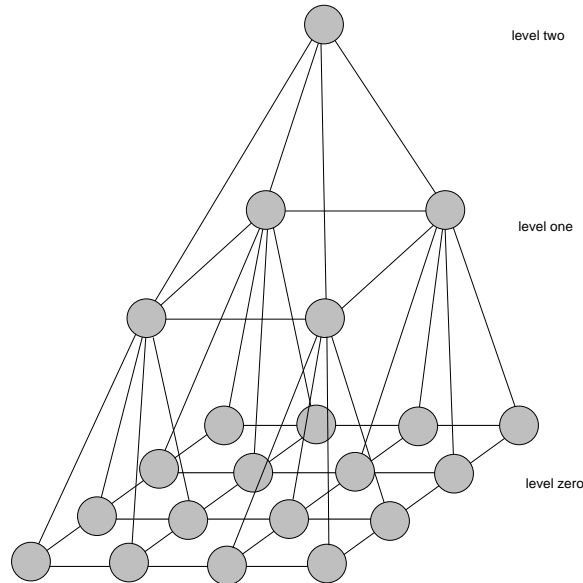


Figure 1: A 4 *by* 4 base pyramid

operate in a strict SIMD mode under the direct control of a single node. Each processing element has its own memory and all of the communication links are bidirectional. The total number of processing elements is given by:  $\sum_{L=0}^{(1/2)\log_2 n} n/(4^L) = (4n/3) - 1/3$ .

The pyramid topology has been proposed as an architecture for high-speed image processing where its simple geometry adapts naturally to many types of problems [4]. Pyramids are more attractive than meshes because they provide the potential for solving problems with logarithmic time complexity.

### 3 Hough Transform

The detection of lines and curves in an image is a fundamental problem in image processing. The problem is often solved by the Hough Transform. The Hough Transform and its generalization have frequently been used to detect object boundaries in image analysis. In the simplest case, the picture contains a number of discrete black figure points lying on a white background. The problem is to detect the presence of groups of collinear or almost collinear figure points. The Hough Transform was first introduced as a method for

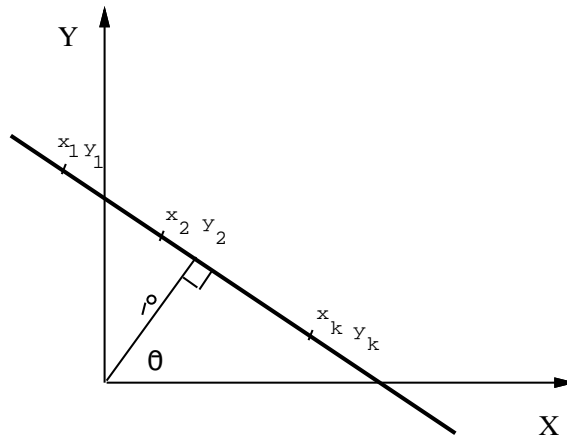


Figure 2: Representation of points in parametric coordinates

detecting complex patterns of points in binary image data. This method has many desirable features. Since, each image point can be treated independently, the method can be implemented using parallel processing.

The Hough Transform algorithm was introduced by Paul Hough in a patent filed in 1962 [8]. Rosenfeld et al. showed that it can be used to detect curves [17]. Later Duda and Hart [5] suggested that straight lines can be parameterized by the length  $\rho$  and orientation  $\theta$  of the normal vector to the line from the image origin. This relationship is given by  $\rho = x * \cos \theta + y * \sin \theta$ . Hough Transform implementation is  $(\rho, \theta)$ . The relationship between a point  $(x, y)$  in the Cartesian space and  $(\rho, \theta)$  is illustrated in Figure 2.

A straight line in the plane is uniquely specified by two parameters such as  $(a, b)$  in the Cartesian space ( $a$  and  $b$  represent slope and intercept respectively), or  $(\rho, \theta)$  in the polar(parameter) space where  $\rho$  is the length of the line segment from the origin perpendicular to the line, and  $\theta$  is the angle that the line makes with the positive  $x$  axis, measured clockwise. The most commonly used coordinate system in the Hough Transform implementation is  $(\rho, \theta)$ . With  $n$  image points the ranges of  $(\rho, \theta)$  can be taken to be  $\rho: [-M, M]$  and  $\theta: [-\pi, \pi]$  where  $M$  depends on the image size. If we restrict  $\theta$  to be in  $[0, \pi]$  range, then normal parameters for the line will be unique. With this restriction, every line in the  $x - y$  plane corresponds to a unique point in  $\rho - \theta$  plane. Given a set of  $n$  figure points,  $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})$ , it is possible to find a set of straight lines that connect some of these points together. For each point  $(x_i, y_i)$  the equation  $\rho = x_i * \cos \theta + y_i * \sin \theta$  creates a set of sinusoidal

curves on the  $(\rho, \theta)$  plane by varying  $\theta$  from 0 to  $\pi$ . Curves corresponding to the set of collinear figure points will have a common point of intersection which defines the line connecting them in the Cartesian coordinates. A point in the Cartesian plane corresponds to a sinusoidal curve in the parameter plane, and a point in the parameter plane corresponds to a straight line in the Cartesian plane.

Given  $n$  pixels  $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1}), (x_i, y_i)$  is transferred into a sinusoidal curve in  $\rho - \theta$  plane defined by  $\rho = x_i * \cos \theta + y_i * \sin \theta$ . The common point of intersection in  $\rho - \theta$  plane, represented as  $(\rho_i, \theta_i)$ , defines a line intersecting at the collinear points. When it is not necessary to determine the lines exactly, we can specify the acceptable error resolution  $\rho_{res}$  and  $\theta_{res}$ . The parameter space  $\rho$  and  $\theta$  can be quantized into  $m$   $\rho$  values  $\rho_1, \rho_2, \dots, \rho_m$  and  $k$   $\theta$  angles,  $\theta_1, \theta_2, \dots, \theta_k$  with an accumulator array of size  $k * m$ . Each entry of this accumulator array corresponds to a point defined by a pair of  $(\rho_i, \theta_i)$ . A region is treated as a two-dimensional array of accumulator cells. For each point  $(x_i, y_i)$  in the Cartesian plane the  $(\rho, \theta)$  of the cell selected by the curve with the relation  $\rho = x_i * \cos \theta + y_i * \sin \theta$  is incremented. A given cell in the two-dimensional accumulator array eventually records the total number of curves intersecting at the point represented by a given  $(\rho, \theta)$ . After all the pixels have been treated, the parameter array is inspected to find those cells with the largest counts. If the count value in a given cell  $(\rho_i, \theta_i)$  is  $z$ , then precisely  $z$  pixels lie (to within the quantization error) along the line whose parameters are  $(\rho_i, \theta_i)$ . Because of the importance of this problem, and its high complexity on a single processor, a lot of attention have been devoted to the development of efficient fine-grain SIMD parallel algorithm.

## 4 An Efficient Parallel Hough Transform

A processor in a  $\sqrt{n} - by - \sqrt{n}$  base pyramid architecture is addressed using a triple index scheme  $(L, j, i)$ . In this notation  $L$  is the level number ( $0 \leq L \leq (1/2) \log_2 n$ ) and  $j$  is the group number ( $1 \leq j \leq n/(4^{L+1})$  for all  $L$  such that  $0 \leq L < (1/2) \log_2 n$ ). The group number is designated clockwise using Hamiltonian cycle ( Hamiltonian cycle is a loop covering every node in the graph/system exactly once) starting from the upper left group, where a group is a 4-connected set of processors; for root processor, group number is zero). Finally,  $i$  is the index of each processor within a group numbered clockwise ( $0 \leq i \leq 3$ ). Group refers to those processors that have the same

parent. We assume that processors at level zero (base) store a whole image at once. Let's consider  $\rho_{res}$  as the resolution or maximum increment between two consecutive  $\rho$  values. This value is directly related to the size of the accumulator array,  $h(\theta_j, \rho_{jp})$ . If there are  $m$   $\rho$  values, then the size of the accumulator array  $h(\theta_j, \rho_{jp})$  for a particular angle  $\theta_j$  is  $m$ , i.e.,  $h(\theta_j, \rho_{jp}), 1 \leq p \leq m : array[1, 2, \dots, m]$  of integers.  $\theta_{res}$  is the resolution or maximum increment between two consecutive  $\theta$  values, i.e.,  $\theta_{res} = 180/k$ , where  $k$  is the number of angles considered for the Hough Transform calculation. Given a pyramid of a specified base size, it is always possible to choose  $\rho_{res}$  and  $\theta_{res}$  so that the accumulator array will meet the memory requirement. For example, if only 16  $\theta$  angles are available then  $\theta_{res} = 180/16 = 11.25$  degree. The procedure for calculating the Hough Transform has two steps. In step one, the accumulator array values are computed. In step two, maximum value in the array is determined. In the following algorithms, first we describe how accumulator array is formed, then we present a method for finding a maximum of the array.

## 4.1 Algorithm One

We assume the number of angles is equal to  $n/4$  ( $n$  is the number of base level processors). Also, each processor at level zero (base) stores the following information in its local memory:

$x_{ji}$  represents the x coordinate of the pixel processed by  $p(0, j, i)$ .  $y_{ji}$  represents the y coordinate of the pixel processed by  $p(0, j, i)$ .  $\rho_{res}$  represents the resolution or maximum increment between two consecutive  $\rho_{jp}$  values. Processor at level zero in group  $j$  stores  $\cos \theta_j$  and  $\sin \theta_j$  values. We assume each value of  $x_{ji}, y_{ji}, \rho_{res}, \cos \theta_j$ , and  $\sin \theta_j$  occupies  $b = \lceil \log_2 \sqrt{n} \rceil$  bits of local memory.

At level one parent processor of group  $j$  stores  $h(\theta_j, \rho_{jp})$ , for all values of  $\rho_{jp}$  such that  $1 \leq p \leq m$ . This is an  $array[1, 2, \dots, m]$  of integers, initially all entries of  $h(\theta_j, \rho_{jp}), 1 \leq p \leq m$ , are equal to zero. This array occupies  $m * b$  bits of local memory. Algorithm One consists of two parts: Sum and Maximum.

### 4.1.1 Algorithm Sum

There are three phases:

1) In this phase, each processor  $p(0, j, i)$  in the group  $j$  computes  $\rho_{jp} = \lfloor (x_{ji} * \cos \theta_j + y_{ji} * \sin \theta_j) / \rho_{res} \rfloor$ .

2) In phase two,  $\rho_{jp}$ 's from level zero are transferred to parent processors at level one, and the count corresponding to  $\rho_{jp}$ th position of  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$  for  $\theta_j$  is incremented, ( $h(\theta_j, \rho_{jp}) = h(\theta_j, \rho_{jp}) + 1$ ).

3) Coordinates of four pixels in group  $j$  located at the base level ( $x_{ji}$  and  $y_{ji}$ ,  $i = 0, 1, 2, 3$ ), are transferred to neighbor processors at the group  $j \bmod (n/4) + 1$  using clockwise Hamiltonian cycle over all  $n/4$  groups.

Phases one, two, and three are repeated  $n/4$  times. At the end of this algorithm, for each  $\theta_j$ , parent processor of group  $j$ ,  $p(1, j, i)$  has the accumulated sum of  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$ , for all of the pixels in the image.

#### 4.1.2 Algorithm Maximum

Each processor at level one will send the first count (corresponding to  $\rho_{j1}$ ) of  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$ , to its parent processor. Parent processors select maximum value of the first count of  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$ , received from their four children and send this value to parents at higher levels. Moreover, parent processors store the computed maximum value for the next iteration. This process is repeated for other entries of  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$ . At the end of this process, the apex processor (processor at level  $(1/2) \log_2 n$ ) has the global maximum count for all values of  $\rho_{jp}$  and  $\theta_j$  in the parameter space.

#### 4.1.3 Algorithm One complexity

In this section, we describe the number of operations needed in different phases of the Algorithm One. In Algorithm Sum at phase one, level zero processors perform two multiplications, one addition, one division, and one round off operation for a total of five operations per step. Assuming the transfer and arithmetic operation for  $b$  bits of data are equal, then in phase two, for each step four transfers and four additions are required. Since transfer and addition operations can be done in a pipeline fashion, the total number of operations in this phase is five. Phase three involves two shift operations. The total number of operations for all three phases is twelve. For  $n/4$  angles, the total number of operations is  $12 * n/4$ . In general for  $k$  angles the time complexity is  $O(k)$ . At the end of this algorithm processors  $p(1, j, i)$  will have the corresponding values of  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$ , for  $k$  angles given by  $\theta_1, \theta_2, \dots, \theta_j, \dots, \theta_k$ .

The time complexity for Algorithm Maximum is described next. For each count of  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$ , there is one transfer and three comparison operations. Since for other values of count in  $h(\theta_j, \rho_{jp})$ ,  $1 \leq p \leq m$  calculation can be completed in a binary tree fashion, then the maximum time is proportional to the number of levels in the pyramid. Thus, the time complexity for the Algorithm Maximum is  $O(\log_2 n)$ . The total time complexity of Algorithm One is given by  $O(k + \log_2 n)$ . Each processing element at the base needs  $6 * b$  bit storage for storing  $x_{ji}, y_{ji}, \rho_{res}, \rho_{jp}, \cos \theta_j$ , and  $\sin \theta_j$ . Each processing element at level one needs  $m * b$  bits for storing the  $h - array$ . The amount of memory needed is proportional to  $h - array$  with  $m \rho_{jp}$  entries resulting in an  $O(m)$  storage for each processing element.

## 4.2 Algorithm Two

In this algorithm  $\rho_{jp}$  is computed for one value of  $\theta_j$  at a time. The accumulated sum and the value of maximum are calculated in parallel. We assume:

$x_{ji}$  represents the x coordinate of the pixel processed by  $p(0, j, i)$ .  $y_{ji}$  represents the y coordinate of the pixel processed by  $p(0, j, i)$ .  $\rho_{res}$  represents the resolution or maximum increment between two consecutive  $\rho_{jp}$  values.

We assume each processor at level one stores the following information:

$\rho_{jp}$  = The computed value  $\rho$  for a given angle  $\theta_j$  and  $c_{jp}$  = The total count for  $\rho_{jp}$ . Initially  $\rho_{jp}$  and  $c_{jp}$ ,  $1 \leq p \leq m$ , are equal to zero.

The four phases of the algorithm are described below:

1) In this phase,  $\cos \theta_j$  and  $\sin \theta_j$  values are sent to each processor at the base level, then  $\rho_{jp} = \lfloor (x_{ji} * \cos \theta_j + y_{ji} * \sin \theta_j) / \rho_{res} \rfloor$  is calculated at each processor.

2) In this phase, each processor transfers the value of  $\rho_{jp}$  to its parent processor.

3) In this phase,  $m$  processors at level one collect  $\rho_{jp}$  values that are computed by the base processors and the corresponding counter  $c_{jp}$  will be incremented. The computed value of  $\rho_{jp}$ 's for a given angle  $\theta_j$  is transferred to the corresponding processor at level one using the shortest path. This implies that the tree connection in the pyramid or Hamiltonian cycle in the first level is used. The selection of the path depends on the position of the  $\rho_{jp}$  with respect to the corresponding level one processor. At the end of this phase  $m$  processors at level one will have the accumulated sum  $\rho_{jp}$  for a particular angle  $\theta_j$  ( $m$

entries of  $h$  - array are stored in  $m$  different processors).

4) Maximum of  $c_{jp}$ ,  $1 \leq p \leq m$ , will be calculated using an algorithm similar to Algorithm Maximum as described before.

Phases one through four are repeated for all values of  $\theta_j, j = 1, \dots, k$ .

#### 4.2.1 Algorithm Two complexity

In phase one, two loads (for  $\cos \theta_j$  and  $\sin \theta_j$ ), two multiplications, one addition, one division, and one round off operations are needed. Phase two needs four transfer operations. Phase three has a time complexity of  $O(\log_2 n)$  using the tree connection. Phase four has the complexity of  $O(\log_2 n)$ . For other values of  $\theta_j$  the calculation can be done in a binary tree fashion. Thus, the total time complexity for  $k$  angles is  $O(k + k * \log_2 n + \log_2 n) = O(k * \log_2 n)$ . Each processor requires a constant amount of storage, and the amount of memory needed is  $O(1)$ .

Table 1 compares different algorithms for calculating the Hough Transform on different architectures. As shown in Table 1, the algorithms defined in this paper are superior to mesh algorithms because of the combined feature of tree and the mesh base of the pyramid architecture. Even with the tree algorithms we can observe that our algorithm has better performance by utilizing the mesh base. As for the previous pyramid algorithms, our work provides a better combined storage and time complexity. This is due to the smaller complexity variables when compared with the previous algorithms in this area. Another work in this area was proposed by Jolin and rosenfeld [9]. The main difference between their scheme and ours is the notion of clustering. Also they considered gray levels and introduced a method for pixel selection which is beyond the scope of our paper. They showed that by using the clustering method you can reduce the amount of data that has to be stored at each level by a factor of four per step. However the technique requires a set of local operations for finding the closest pair of segments and reducing the number of segments until they are only  $k$  left. Since in our work we considered the entire Hough space, we require more storage with fewer number of local operations, rendering a faster convergence by a constant factor.

<i>Algorithm</i>	<i>no. of pixels</i>	<i>Memory</i>	<i>Time Complexity</i>	<i>Network</i>
[Chu85]	$k^2$	$O(1)$	$O(n + k)$	Systolic
[Cyp87]	$n$	$O(1)$	$O(\sqrt{n} + k)$	Mesh $\sqrt{n} * \sqrt{n}$
[Kan90]	$k^2$	$O(k)$	$O(\sqrt{n} + k)$	Mesh $\sqrt{n} * \sqrt{n}$
[Sil85]	$n$	$O(1)$	$O(k * \sqrt{n})$	Mesh $\sqrt{n} * \sqrt{n}$
[Ibr85]	$k^2$	$O(1)$	$O(n)$	Tree
[Pan90]	$n$	$O(k)$	$O(\sqrt{n} + k)$	Cube size $n$
[Pan90]	$n$	$O(k)$	$O(k * \log_2 n / \log_2(k * n))$	Cube size $n$
[Pan90]	$nk^2$	$O(1)$	$O(\log_2 n + \log_2 k)$	Cube size $n * k^2$
[Pan90]	$n * k^2 / \log_2 n$	$O(\log_2 n)$	$O(\log_2 n + \log_2 k)$	Cube size $n * k^2$
[Jol89]	$n$	$O(1)$	$O(\log_2 n)$	Pyramid $\sqrt{n} * \sqrt{n}$
[Bon90]	$n$	$O(1)$	$O(k * \sqrt[4]{n})$	Pyramid $\sqrt{n} * \sqrt{n}$
Alg. one	$n$	$O(m)$	$O(k + \log_2 n)$	Pyramid $\sqrt{n} * \sqrt{n}$
Alg. two	$n$	$O(1)$	$O(k * \log_2 n)$	Pyramid $\sqrt{n} * \sqrt{n}$

Table 1: Comparison of time and memory complexities for different algorithms.

## 5 Simulation Program

The simulation program for detecting lines was written in C language. Figure 3 illustates a 16 by 16 image created by *bitmap editor* under X windows [20]. The bitmap file of the image in Figure 3 is illustrated in the Figure 4 and is called portable bitmap(pbm).

In the pbm file 1 means *black* and 0 means *white*. The simulation program consists of two major parts: *Simulation Controller* and *Algorithm Controller*. The Simulation Controller is used for scheduling. It offers a menu to the user to select the size of a pyramid.

The simulation program has a global clock called *Simulation Clock* and incremented whenever the processors at the same level terminate execution. Therefore, at the end of the program, Simulation Clock denotes the number of times that processors activated during execution of the program. Figures 5, 6, 7, and 8 illustrate the flow diagrams and the pseudo-codes for line detection algorithms.

Table 2 illustrates the computer simulation results for different size binary images using Algorithm One and Algorithm Two.

Figure 9 represents the relationship between image size and simulation clock for line detection algorithms.

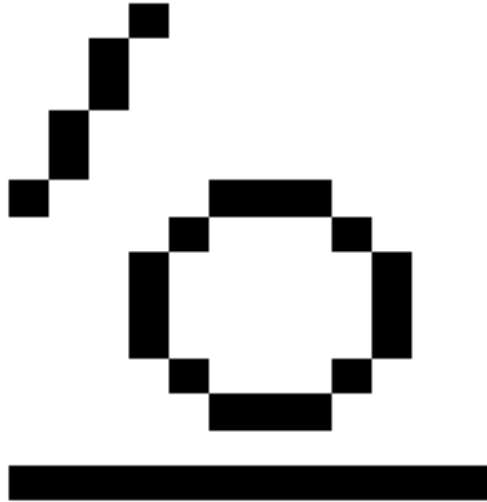


Figure 3: The 16x16 image created by bitmap editor.

```

0000000000000000
0000100000000000
0000100000000000
0000100000000000
0001000000000000
0001000000000000
0001000000000000
0010000111000000
0000001000100000
0000010000010000
0000010000010000
0000010000010000
0000001000100000
0000000111000000
0000000000000000
0011111111111100
0000000000000000

```

Figure 4: The pbm file of a 16x16 image.

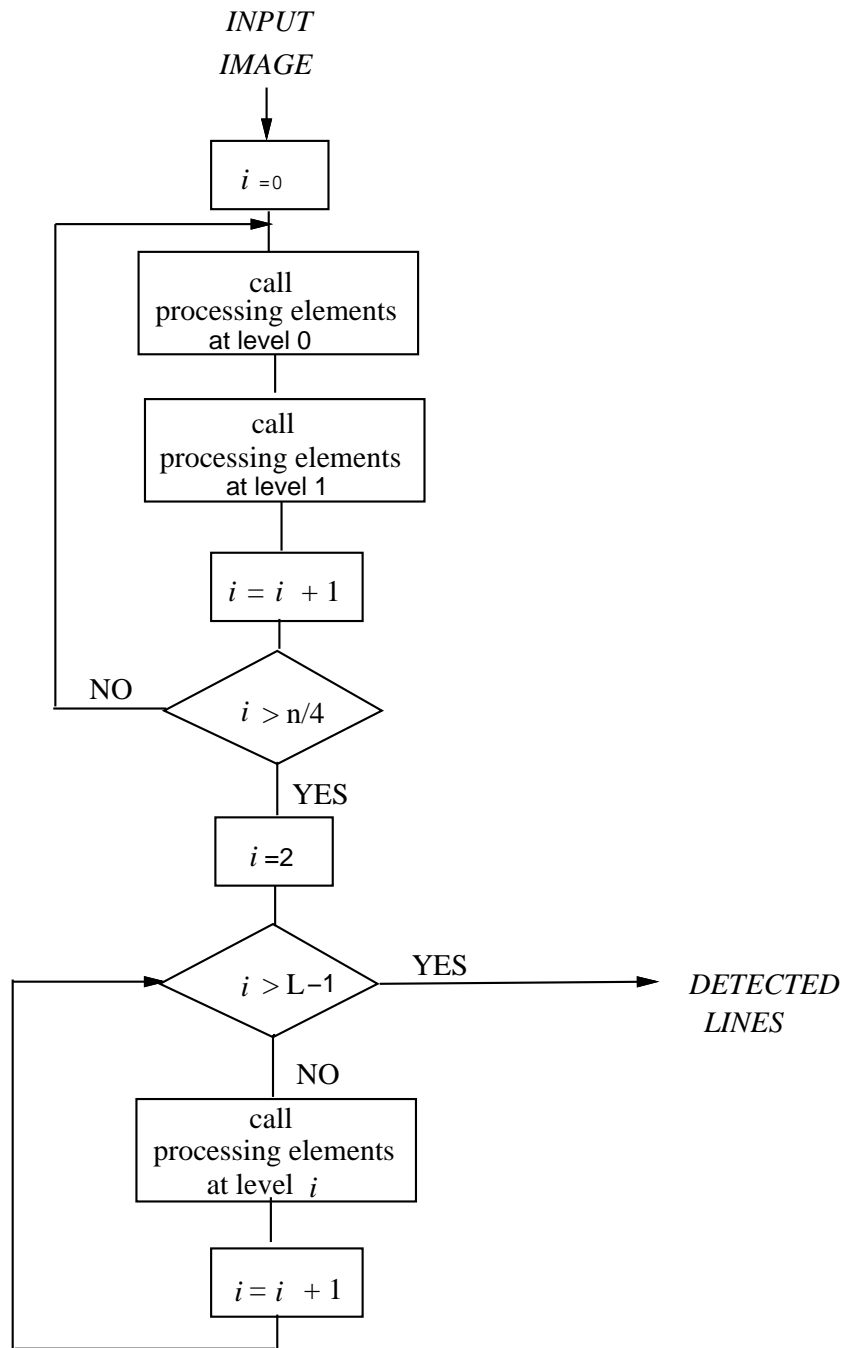


Figure 5: The flow diagram of Algorithm One.

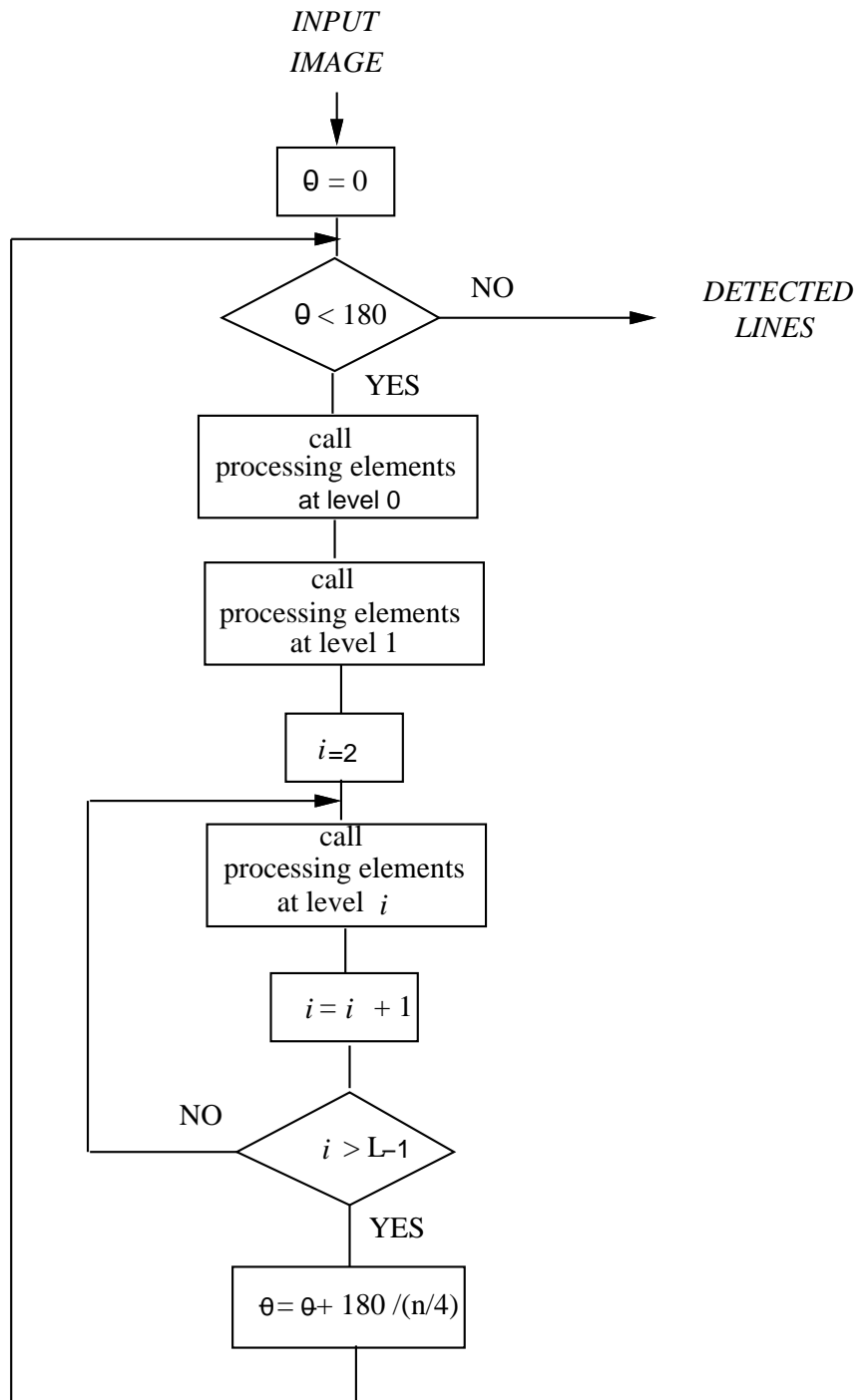


Figure 6: The flow diagram of Algorithm Two.

```

for i = 1 to n/4
    for all processing elements at level zero do
        receive pixel's coordinates
        compute  $\rho$ 
        send  $\rho$  and coordinates to parent's mailbox
    end /* for all */
processor1 ()
    for all processing elements at level one do
        receive  $\rho$ 's and four coordinates from four children
        increment the corresponding counter /*  $\rho$  */
        send four coordinates to neighbor processing element
        receive four new coordinates from neighbor processing element
    end /* for all */
processor2 ()
end /* for i */
for k = 1 to m /* different iterations of  $h$  */
    for all processing elements at level one do
        send  $h$  [t, m] to parent processing element
    end /* for all */
    for j=2 to L-1 /* for all processing elements from level two
to L-1 */
        for all processing elements at level j do
            receive four  $h$  values
            select  $max = \text{maximum}(h)$ 
            if ( $j < L-1$ )
                send  $max$  to parent processing element at level  $j + 1$ 
            else /* apex processor */
                store  $max$ 
            end /* for all */
processor3 ()
        end /* for j */
    end /* for k */
end /* for k */
sort stored  $max$  values
select values(  $max > \text{threshold}$  ) /* they represent straight lines */

```

Figure 7: The pseudo-code of Algorithm One.

```

for r = 1 to k
  calculate  $\sigma$ 
  send  $\sigma$  to all processing elements at level zero
  for all processing elements at level zero do
    receive  $\sigma$ 
    calculate  $\rho$ 
    send to parent processing element
  end /* for all */
processor1()
  for all processing elements at level one do
    receive four  $\rho$ 's from four children
    send  $\rho$ 's to corresponding processing element at level one
  end /* for all */
processor2()
  for all processing elements at level one do
    receive  $\rho$ 's from processing elements at level zero
    increment the corresponding counter (  $c$  )
    send counter  $c$  to parent processing element at upper level
  end /* for all */
processor2()
  for z=2 to L-1 /*for all processing elements at level two to L-1*/
    for all processing elements at level z
      receive four  $c$  values from four children
      select  $max = \text{maximum} ( c )$ 
      if (z <> L-1)
        send  $max$  to parent processing element at upper level
      else /* apex processing element */
        store  $max$ 
      end /* for all */
    end /* for z */
  end /* for k */
  sort stored  $max$  values 15
  select values (  $max > \text{threshold}$  ) /* they represent straight lines */

```

Figure 8: The pseudo-code of Algorithm Two.

### Algorithm One and Algorithm Two

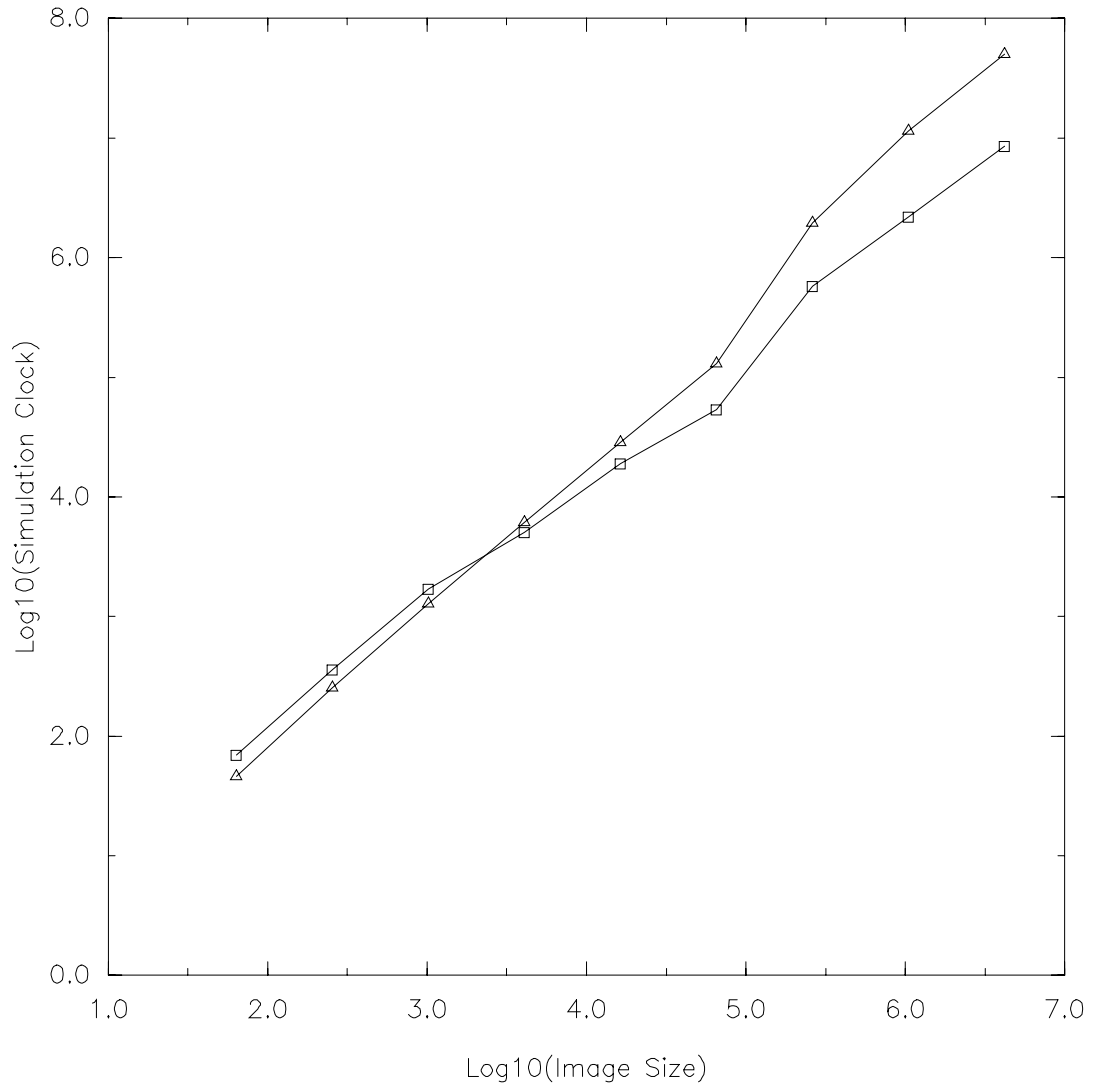


Figure 9: Simulation clock for line detection algorithms

<i>Image size</i>	<i>Algorithm One</i>	<i>Algorithm Two</i>
8 by 8	70	46
16 by 16	362	256
32 by 32	1,700	1,280
64 by 64	5,033	6,144
128 by 128	18,974	28,672
256 by 256	53,747	131,072
512 by 512	164,648	589,824
1024 by 1024	578,261	1,966,080
2048 by 2048	2,205,122	11,534,336
4096 by 4096	8,553,575	50,331,648

Table 2: Simulation clock for Algorithm One and Algorithm Two

## 6 Summary

In this paper two new and efficient parallel algorithms for calculating the Hough Transform on a pyramid architecture were described. In these algorithms both mesh and tree connections of a pyramid were exploited. The result of simulation proves the usefulness of pyramid architecture in image processing and pattern recognition.

## References

- [1] R. P. Blanford, "Dynamically Quantized Pyramids for Hough Vote Collections," *Proc. IEEE Workshop Computer Architecture Pattern Anal. Machine Intell.*, pp. 145-152, Oct. 1987
- [2] G. Bongiovanni, C. Guerra, and S. Levialdi, "Computing the Hough Transform on a Pyramid Architecture," *Machine Vision and Applications*, vol. 3(2), pp. 117-123, 1990
- [3] H. Y. H. Chuang and C. C. Li, "A systolic array processor for straight line detection by modified Hough transform," *Proc. IEEE Workshop Computer Architecture Pattern Anal. Image Database Manag.*, pp. 300-304, Nov. 1985

- [4] R. E. Cypher, J. L. C. Sanz, and L. Snyder, "The Hough Transform has  $O(N)$  complexity on SIMD  $N*N$  mesh array architecture," *Proc. IEEE Workshop Computer Architecture Pattern Anal. Machine Intel.*, pp. 115-121, Oct. 1987
- [5] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *CACM*, 15, (1), pp. 11-15, 1972
- [6] A. L. Fisher and P. T. Highnam, "Computing the Hough Transforms on a Scan Line Array Processor," *IEEE Trans. on PAMI*, vol. 11, no. 3, pp.262-265, March 1989
- [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," *Proceedings IEEE Conf. Comput. Vis. Pattern Recog.*, pp. 248-256, June 1985
- [8] P. V. C. Hough, "A method and means for recognizing complex patterns," U.S. Patent 3,069,654, 1962
- [9] J. M. Jolion and A. Rosenfeld, "An  $O(\log(n))$  Pyramid Hough Transform," *Pattern Recognition Letters*, vol. 9, pp. 343-349, 1989
- [10] C. S. Kannan and H. Y. H. Chuang, "Fast Hough Transform on a Mesh Connected Processor Array," *Information Processing Letters*, vol. 33, pp. 243-248, 1990
- [11] J. J. Little, G. Blecco, and T. Cass, "Parallel Algorithms for Computer Vision on the connection Machine," *The First IEEE Inter. Conf. on Computer Vision*, pp. 587-591, 1987
- [12] A. Kavianpour, and N. Bagherzadeh, "Circle Detection in Black and White Images," *Patent pending*, UC Case no. 91-195-1, 1991
- [13] A. Kavianpour, and N. Bagherzadeh, "Finding Circular Shapes in an Image on a Pyramid Architecture," *Pattern Recognition Letters*, vol. 13, no. 12, pp. 843-848, December 1992
- [14] M. Maresca, H. Li, and M. Sheng, "Parallel Computation on a Polymorphic Torus Architecture," *Machine Vision and Applications*, vol. 2(4), pp. 215-230, 1989

- [15] Z. N. Li, D. Zhang, "Fast Line Detection in a Hybrid Pyramid," *Pattern Recognition Letters*, vol. 14(1), pp. 53-63, 1993
- [16] Yi Pan and Henry Y.H. Chuang, "Parallel Hough Transform Algorithms on SIMD Hypercube Array," 1990 *International Conference on Parallel processing*, pp. 83-86, Aug. 1990
- [17] A. Rosenfeld, *Picture processing by computer*, Academic Press, 1969
- [18] T. M. Silberberg, "The Hough Transform on the geometric arithmetic parallel processor," *Proc. IEEE Workshop Computer Architecture Pattern Anal. Machine Intel.*, pp. 387-393, Oct. 1985
- [19] S. L. Tanimoto, T. J. Ligoeki, and R. Ling, "A Prototype of pyramid machine for hierarchical cellular logic," *Parallel Hierarchical Computer Vision*, L. Uhr, Ed. London, 1987
- [20] Unix on-line computer manual