1987

Parallel Algorithms for Line Detection on a Mesh

Concettina Guerra

Susanne E. Hambrusch
Purdue University, seh@cs.purdue.edu

Report Number:
87-663

PARALLEL ALGORITHMS FOR LINE DETECTION ON A MESH

Concettina Guerra
Susanne E. Hambrusch

CSD-TR-663
February 1987

Parallel Algorithms for Line Detection on a Mesh

*Concettina Guerra and Susanne Hambrusch**

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907, USA.

February 1987

CC-87-13

Abstract. We consider the problem of detecting lines in an $n \times n$ image on an $n \times n$ mesh of processors. We present two new and efficient algorithms which perform a Hough transform by projection. Our first algorithm runs in time $O(k\sqrt{m}+n)$, where k and m are the number of θ- and ρ-values in the parametric representation of the lines in the Hough transform, respectively. The second algorithm runs in $O(n+k)$ time. Both algorithms perform only simple data movement operations over relatively short distances.

Keywords. Hough transform, image processing, line detection, meshes, parallel processing.

---

* This work was supported by the Office of Naval Research under Contracts N00014-84-K-0502 and N00014-86-K-0689, and by the National Science Foundation under Grant DMC-84-13496.

1. Introduction

The detection of lines and curves in an image is a fundamental problem in image processing. The problem is often solved by a Hough transform [BA, DH, HH], a method based on a relation between points lying on a line or curve in the image space and the parameters of that curve. A line is parameterized by two values θ and ρ according to the following expression:

$$\rho = x\cos\theta + y\sin\theta \qquad (1)$$

where θ is the angle of the normal line and ρ is the distance of the line from the origin. The transformation associates a point (x,y) of the image space with a line of the parameter space and has the following property: Points lying on a line are transformed into lines intersecting at the same point in the parameter space. This property allows us to convert the line detection problem into an intersection problem in the parameter space. This latter problem is solved as follows [HH, DH]. The parameter space is quantized into k θ-values, $\theta_0, \cdots, \theta_{k-1}$, and m ρ-values, $\rho_0, \cdots, \rho_{m-1}$, and an accumulator array of size $k \times m$ with each entry corresponding to a pair (θ_i, ρ_j).

We assume that all the edge points of the $n \times n$ image have been determined. In order to solve the line detection problem each edge point (x,y) in the image space "votes" for the parameter values of possible lines passing trough it; i.e., for every θ_i the ρ-values of the lines passing through (x,y) are derived from (1) and the corresponding entries in the accumulator array are incremented. After all edge points have been treated, the entry with the maximum number of votes is found. The sequential implementation of this procedure uses $O(km+b\min\{m,k\})$ time, where b is the number of edge points in the image, $b \leq n^2$.

In this paper we present two new and efficient parallel algorithms for performing the Hough transform on an $n \times n$ mesh of processors. Our first algorithm, which we call the block algorithm, runs in $O(k\sqrt{m}+n)$ time, and our second algorithm, called the tracing algorithm, runs in $O(n+k)$ time. The algorithms assume that the mesh contains an $n \times n$ binary image with one image point per processor. Both algorithms use projections of the image along different directions to determine the number of edge points lying on every line (θ_i, ρ_j), $0 \leq i \leq k-1$, $0 \leq j \leq m-1$. It is well known that the Hough transform can be viewed in terms of projections [SD]. The projection of a picture in a direction θ is obtained by adding up the edge points of the image along the family of

lines perpendicular to θ.

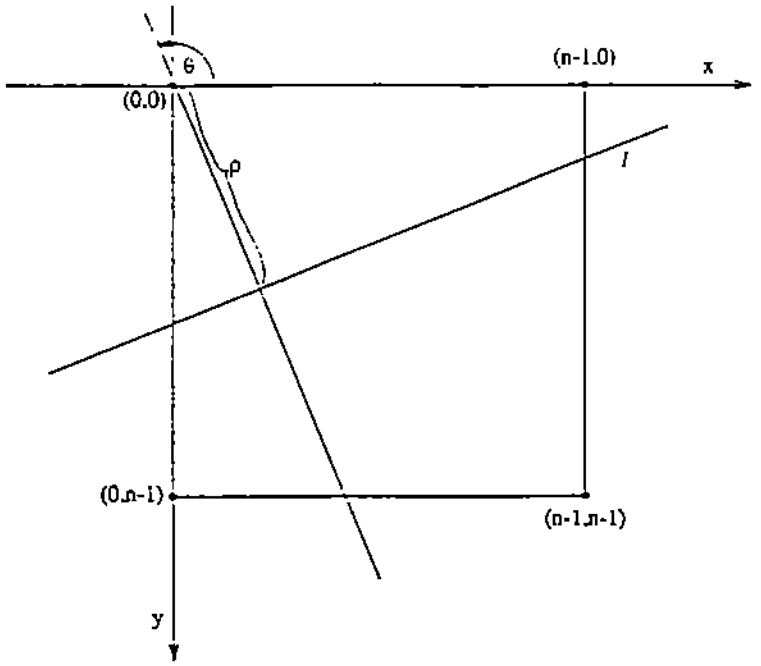
The block algorithm partitions the mesh in submeshes of appropriate sizes and combines the partial results accumulated in the submeshes in k stages. The time bound of $O(k\sqrt{m}+n)$ can also be achieved by taking votes, but the accompanying constant is larger. The second algorithm performs projections by tracing lines through the image in a pipelined fashion. While the tracing algorithm is asymptotically optimal and superior to the block algorithm for $k\sqrt{m} = \Omega(n)$, we expect the block algorithm to outperform the tracing algorithm in an actual implementation (e.g., on the MPP [BT]). While many mesh algorithms [AH, MS, NS] rely heavily on sorting as a subroutine for performing arbitrary data movement, both of our algorithms do not use sorting. Both algorithms perform only simple data movement operations over relatively short distances in the mesh.

We briefly review some of the previously reported parallel implementations for the Hough transform. An implementation for an $n \times n$ mesh running in time $O(nk)$ is described in [SI]. In this algorithm the accumulation process for every θ-value is carried out by first accumulating along the rows of the mesh and then along the columns in a manner similar to histogramming [KR]. Algorithms for linear systolic arrays are described in [CL, KW]. An implementation for a tree machine consisting of bk processors is proposed in [IK]. An $O(n^2k)$ time implementation of the Hough transform, based on projections of lines along different directions, is presented in [SD] for a linear pipeline. A similar idea is used in [FH] on a scanline array processor.

2. Preliminaries

Throughout the paper we assume that processor (0,0) is the top-left processor of an $n \times n$ mesh, and processor (cx,ry) is the processor at column cx and row ry. This unusual indexing is done to have the indexing in the mesh agree with the indexing of the image points in the x–y-plane. We thus assume that the image is in the lower-right quadrant of the x–y plane with the positive y-axis as shown in Figure 2.1. Every image point has integer coordinates and the image point at (cx,ry) is stored at processor (cx,ry) in the mesh. From now on we refer to the image points as pixels and to an edge point as a 1-pixel.

The lines to be considered by our algorithms are given by k θ-values,



The image in the $x-y$ plane; line l has a positive ρ value
Figure 2.1

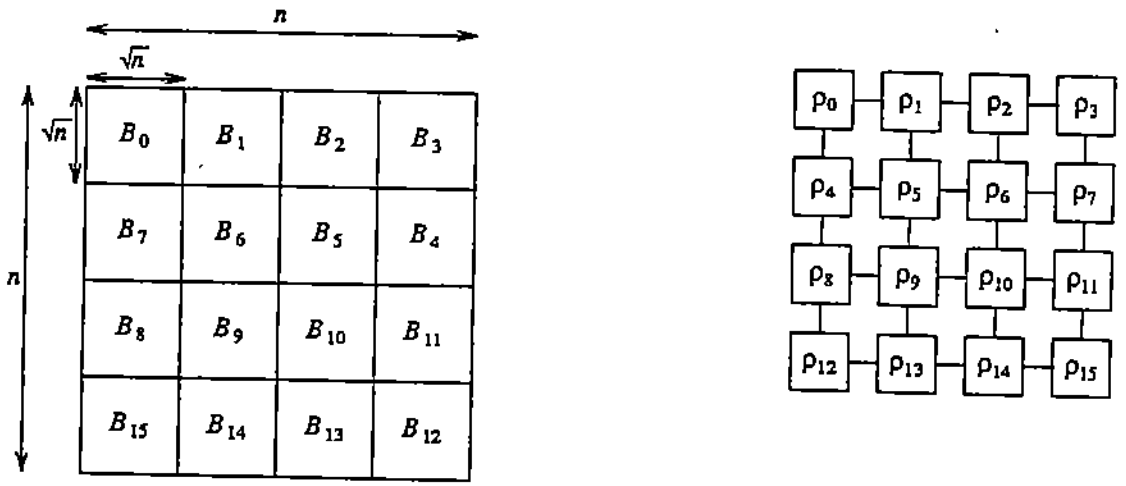
$0 \le \theta_0 < \theta_1 < \cdots < \theta_{k-1} \le 180$, $k \le n$, and n ρ-values, $-n\sqrt{2} \le \rho_0 < \cdots < \rho_{n-1} \le n\sqrt{2}$. The modifications to be done for $m \neq n$ ρ-values are either straightforward or are described when necessary. Our algorithms make no assumptions about the distribution of the θ-values and in many practical applications the k θ-values will be spaced equally. We do, however, make the following reasonable assumption about the ρ-values. Let *cell* (c,r) be the square of unit size associated to pixel (c,r); i.e., the square that has point (c,r) in the $x-y$ plane as the bottom-left corner. We say a line l *crosses* cell (c,r) if l and cell (c,r) have a non-empty intersection. The assumption we make with respect to the ρ-values is that $|\rho_i - \rho_{i+1}| \ge \sqrt{2}$ for any i. This guarantees that no two parallel lines cross a common cell.

Our algorithms can be modified to handle the case when a larger number of θ- and ρ-values are to be considered. As long as $k = O(n)$, the claimed time bounds of $O(n\sqrt{m} + n)$ and $O(n)$ hold, respectively. With respect to the number of ρ-values, our algorithms can easily handle up to $2n$ ρ-values as long as no two parallel lines cross the same cell. If more than $2n$ ρ-values are to be considered, the routing routines of the algorithms need to be adjusted to the handle the larger number of parallel lines crossing a cell.

We represent a line (θ,ρ) in the image plane as the sequence of cells (c,r) crossed by the line and we refer to it as a digital line. This is a standard representation and its properties have been widely studied in image processing [FR, RW]. Thus, when counting the number of 1-pixels on a line our algorithms count the number of cells (c,r) associated with a 1-pixel and crossed by the line.

3. The Block-Algorithm.

We first describe the block-algorithm for the case when $k=m=n$. In this case we partition the $n\times n$ mesh into n blocks, $B_0, B_1, \cdots, B_{n-1}$ with each block being a submesh of size $\sqrt{n}\times\sqrt{n}$. Let the blocks be indexed in snake-like row-major order as shown in Figure 3.1(a). The processors in block B_i are numbered from 0 to $n-1$ in row-major order as shown in Figure 3.1(b).



(a) partitioning into blocks (b) distribution of ρ-values in block B_i

Figure 3.1

The algorithm starts by providing the j-th processor of every block with a copy of ρ_j, $0\le j\le n-1$, as shown in Figure 3.1(b). It then executes n iteration steps. Every *iteration step* consists of two phases, the line computation and the shift phase. The line computation proceeds on each block independently and simultaneously. During the line computation every block B_i has a unique value θ associated with it. For every line represented by (θ,ρ_j), $0\le j\le n-1$, the line computation determines the number of 1-pixels in block B_i that lie on the line. This value is added to a variable *SUM* in processor j of B_i (which is initially set to 0). In the shift phase,

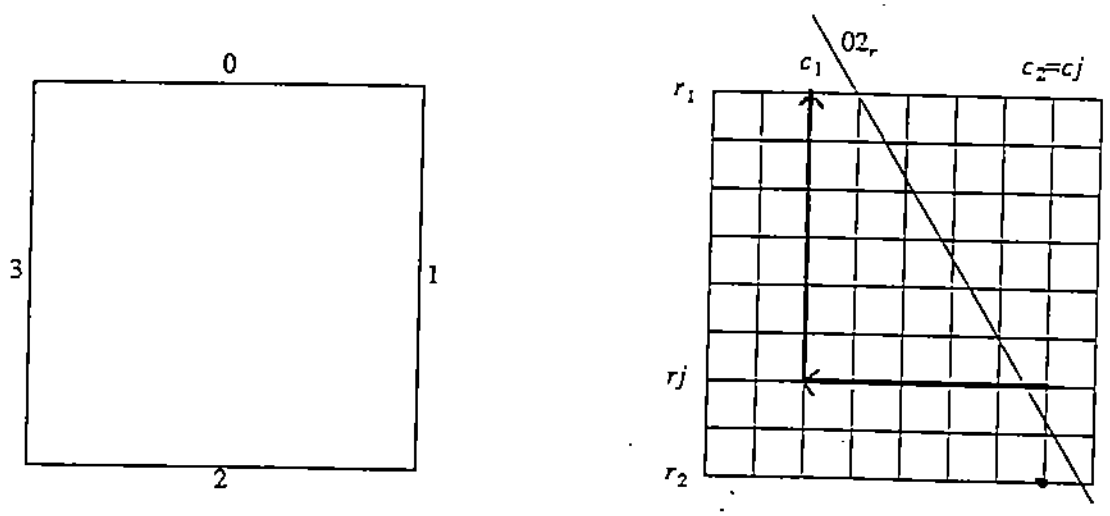
which follows the line computation, block B_i receives the SUM entries and the associated θ-value from block $B_{(i+1) \bmod n}$. After n iteration steps, every one of the n θ-values has been considered in every block and the SUM entries represent the solution. We describe solutions to the line computation and shift phase that run in $O(\sqrt{n})$ time, respectively, and thus the $O(n\sqrt{n})$ overall time of the block-algorithm for $k=m=n$ follows.

3.1 The Line Computation in a Block B.

We now turn to the implementation of the line computation done in every block B. We first describe an $O(\sqrt{n})$ time solution for the line computation that is conceptually simple, but is expensive from a practical point of view since it uses sorting as a subroutine [RE, RM]. Let processor j of block B contain a 1-pixel. Since processor j knows θ, it can compute $\hat{\rho}_j$ such that the line represented by $(\theta,\hat{\rho}_j)$ contains the 1-pixel. After every processor of B has computed a possible $\hat{\rho}$-value, we determine how many processors have the same $\hat{\rho}$-value and add the resulting values to the appropriate SUM entries in the block. This can be done in $O(\sqrt{n})$ time by sorting the $\hat{\rho}_j$'s, performing the summation, followed by a simple routing step and the final addition. We next describe an implementation of the line computation that does not use sorting, only simple routing steps.

In this implementation we first mark processor j if the line represented by (θ,ρ_j) crosses at least one cell of block B. There can be at most $2\sqrt{n}-1$ marked processors and they occupy consecutive processor locations (i.e., if s processors have been marked and processor i is the one with smallest index, processors $i+1,\cdots,i+s-1$ are also marked). Every marked processor j determines the two cells on the border of B crossed by the line (θ,ρ_j). This is done by computing the intersection of the line $\rho_j = x\cos\theta + y\sin\theta$ with the two horizontal and vertical border lines of B, respectively. Let (c_1,r_1) and (c_2,r_2) be the two cells on the border, $r_1 \le r_2$. Note that a line may cross more than two cells on the border of B. How to handle such a situation and which two cells to select will become apparent from the algorithm.

We denote the four sides of B as side 0,1,2, and 3 (as shown in Figure 3.2(a)), and classify the lines into the following eight types. A line represented by (θ,ρ_j) is a line of type $i(i+1)$ if one border cell lies on side i and the other one on side $(i+1)$. (The additions are all done modulo 4.)



(a) the sides of B

(b) interior-to-border routing path

Figure 3.2

A line is of type $i(i+2)_r$ (resp. $i(i+2)_l$) if one border cell lies on side i, the other one on side $(i+2)$, and $c_1 \leq c_2$ (resp. $c_1 > c_2$). Note that the type of lines that can occur in block B can be described easily. For example, if the block contains a line of type 01, it can possibly contain lines of type 31_r or 02_r (but not both), and possibly lines of type 32.

After the border squares have been determined, every marked processor creates a packet (j,ρ_j,s) with $s=0$. This packet is sent to the border processor (c_1,r_1). Once all the packets have reached the border, processors (c_1,r_1) initiate the "trace". In the trace the packets visit all the processors associated with cells crossed by the line (θ,ρ_j). The number of 1-pixels encounters are accumulated in the entry s of each packet. The final step is to send the packet from processor (r_2,c_2) back to processor j. We give below a more detailed description of the line computation and then show that all the routing steps are congestion-free.

The marking step executes only a constant number of internal computations. The interior-to-border routing consists of two parts, each one requiring no more than $\sqrt{n}-1$ data movements. See Figure 3.2(b) for an example. In the implementation all packets complete the first part of the routing before starting the second part. A packet that has completed the first part of the routing, is stored at a processor and other packets may still 'travel' through this processor in order to complete their first part of the routing. Once all packets have completed the first part, a processor

Procedure LINE_COMP;

(1) *Marking Step*
(* every processor contains an entry (θ,ρ_j) *)
if the line (θ,ρ_j) crosses at least one cell in block B
then $MARK = 1$; compute (c_1,r_1) and (c_2,r_2); create packet $(j,\rho_j,0)$
else $MARK = 0$

(2) *Route Interior-to-border*
(* send packet (j,ρ_j,s) to processor (c_1,r_1) *)
Let j be a marked processor in row rj and column cj.
case of
$c_1 \le c_2$ and (c_1,r_1) is on side 0:
send packet (j,ρ_j,s) to column c_1 by performing $|c_1-cj|$ horizontal movements;
send packet (j,ρ_j,s) to (c_1,r_1) by performing $|r_1-rj|$ vertical movements;

$c_1 \le c_2$ and (c_1,r_1) is on side 3:
send packet (j,ρ_j,s) to row r_1 by performing $|r_1-rj|$ vertical movements;
send packet (j,ρ_j,s) to (c_1,r_1) by performing $|c_1-cj|$ horizontal movements;

$c_1 > c_2$ and (c_1,r_1) is on side 0:
send packet (j,ρ_j,s) to column c_1 by performing $|c_1-cj|$ horizontal movements;
send packet (j,ρ_j,s) to (c_1,r_1) by performing $|r_1-rj|$ vertical movements;

$c_1 > c_2$ and (c_1,r_1) is on side 1:
send packet (j,ρ_j,s) to row r_1 by performing $|r_1-rj|$ vertical movements;
send packet (j,ρ_j,s) to (c_1,r_1) by performing $|c_1-cj|$ horizontal movements;
endcase

(3) *Tracing Step*
set $(c,r) = (c_1,r_1)$
while processor (c_2,r_2) has not been reached **do**
Packet (j,ρ_j,s) is at processor (c,r) and the line (θ,ρ_j) crosses cell (c,r); assume the line is of type 01, 23, 02_r, or 13_r (the situation for the other four types is symmetrically).

if processor (c,r) contains a 1-pixel **then** $s=s+1$
$\hat{c}$ = position where the line (θ,ρ_j) crosses row $r+1$
if $\hat{c} \le c$ **then** send packet to processor $(c,r+1)$
else send packet to processor $(c+1,r)$
endwhile

(4) *Route Border-to-interior*
Similar to step (2);
SUM = SUM + s
end.

may contain 0, 1 or 2 packets. If it contains two packets, these packets will leave this processor in 'different directions'. Hence, the interior-to-border routing is congestion-free and completed after $2\sqrt{n}-2$ data movement steps. Note that it is crucial whether the horizontal or the vertical movement is done first. The tracing step visits, for every line (θ,ρ_j), the cells crossed by this

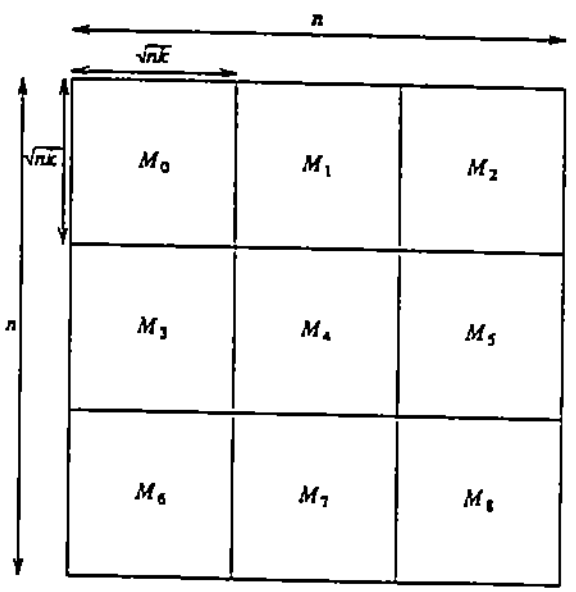
line. Since no two parallel lines cross the same cell, every processor is visited by only one packet. After at most $2\sqrt{n}-1$ data movements, all the packets have reached processor (c_2,r_2). Step (4) is similar to step (2), and thus LINE_COMP runs in $O(\sqrt{n})$ time.

3.2. The Shift-Phase.

When snake-like row-major indexing is used for the blocks, block B_i receives new SUM values and a new θ value from block $B_{(i+1) \bmod n}$. Hence, $n(n-1)$ blocks receive the new entries from blocks within the same row (using horizontal connections), $n-1$ blocks receive them from blocks within the same column (using the vertical connections), and block B_{n-1} receives its new entries from B_0 (using the wrap-around connections). When performing the shifts in this order, block B_i considers $\theta_{f(l,i)}$, where $f(l,i) = (i+l) \bmod n$, at the l-th iteration, $0 \le l \le n-1$. While the running time of a single shift phase in obviously $O(\sqrt{n})$, our planned MPP implementation will consider the indexing resulting in the running time with the smallest constant. Note that the use of the wrap-around connections is not necessary for achieving $O(\sqrt{n})$ time. Using the shuffled row-major indexing (shown in Figure 3.3), $O(\sqrt{n})$ time can be achieved using horizontal and vertical connections only.

B_0	B_1	B_4	B_5
B_2	B_3	B_6	B_7
B_8	B_9	B_{12}	B_{13}
B_{10}	B_{11}	B_{14}	B_{15}

Shufffled row-major indexing
Figure 3.3



Partitioning into submeshes when $k < n$
Figure 3.4

3.3. Generalizing the Block-Algorithm.

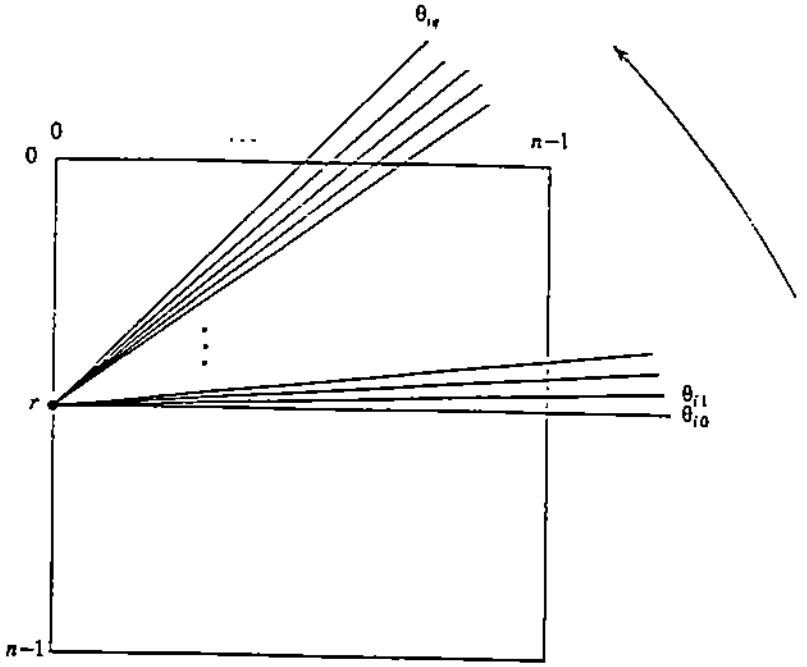
When $k<n$ and $m=n$ we modify the algorithm in the following way. Partition the $n\times n$ mesh into n/k submeshes of size $\sqrt{nk}\times\sqrt{nk}$ each. Let $M_0,\cdots,M_{n/k-1}$ be the resulting submeshes (as shown in Figure 3.4). Within every submesh run the algorithm as described for $k=n$ by creating k blocks of size $\sqrt{n}\times\sqrt{n}$ each. The final step computes the sum of the corresponding SUM entries in the submeshes. This step is done in $O(n)$ time and thus the total time is $O(k\sqrt{n}+n)$.

When we are given k θ-values and m ρ-values, $k\le n$, $m\le n$, the number of 1-pixels on each of the km lines can be determined in $O(k\sqrt{m}+n)$ time. This algorithm works simply with blocks of size $\sqrt{m}\times\sqrt{m}$ instead $\sqrt{n}\times\sqrt{n}$.

4. A Linear Time Tracing Algorithm

This section presents the tracing algorithm which performs line detection on k θ-values and n ρ-values in $O(n+k)$ time. In the block-algorithm presented in the previous section the accumulation of 1-pixels lying on a line (θ,ρ) is obtained by tracing part of the line in a unique submesh B_i at every iteration step. Recall that in every block B_i only parallel line segments are traced simultaneously and thus no collisions can occur. In the tracing algorithm described in this section we start tracing a line (θ,ρ) at one border processor of the mesh and move towards the second border processor (which is either on the opposite or an adjacent side of the mesh).

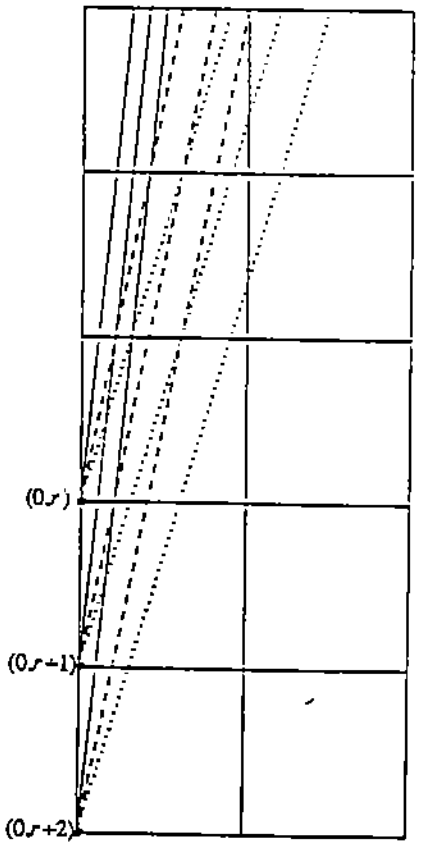
The algorithm initiates a total of eight *trace sequences*. In the $2i$-th and $2i$+1-st trace sequence a subset of the lines crossing side i of the mesh is traced, $0\le i\le 3$. The numbering of the sides is as shown in Figure 3.2(a). During a trace sequence lines with the same θ-value (i.e., parallel lines) start their trace from a border processor at the same time. When a trace is started at a border processor, a packet (θ,ρ,s) with $s=0$ is created. Recall that the entry s counts the number of 1-pixels on the line (θ,ρ). Traces move through the mesh in a pipelined fashion and at some point during a trace sequence $O(kn)$ lines are traced simultaneously. Parallel lines can again cause no congestion. But non-parallel lines originating at the same border from different processors at different times could cause congestion. Their packets (θ,ρ,s) could reach the same processor at the same time and may want to leave on the same link. We next describe how to organize the eight trace sequences such that the overall $O(n+k)$ time bound is obtained.



The lines in the tracing sequence starting from side 3 with $90 \le \theta \le 135$; arrow indicates order in which trace of lines is started

Figure 4.1

discussions will apply, with minor changes, to the other three sides. Let $(0,r)$ be any processor on side 3. The first trace sequence starting from side 3 considers all the lines (θ,ρ) crossing cells $(0,r)$, $0 \le r \le n-1$, with $90 \le \theta \le 135$. These lines are shown in Figure 4.1. The second trace sequence starting from side 3 considers the lines crossing cells $(0,r)$ with $45 \le \theta \le 90$. Intuitively, the traces corresponding to lines (θ,ρ) with $45 \le \theta \le 135$ move away from side 3 "fast enough" and do thus not collide with the traces of other lines. Traces corresponding to lines (θ,ρ) with $135 < \theta < 180$ or $0 < \theta < 45$, "stay closer" to side 3 and this can cause collisions. Consider, for example, Figure 4.2 which shows the nine lines formed by θ_i, θ_{i+1}, θ_{i+2}, and ρ, ρ' and ρ'', $135 < \theta_i < \theta_{i+1} < \theta_{i+2} < 180$. Assuming that parallel lines start their trace at the same time and the trace of the lines corresponding to θ_j is initiated either right before or right after the trace for θ_{j+1}, collisions cannot be avoided.



Dotted lines have direction θ_i
Dashed lines have direction θ_{i+1}
All other lines have direction θ_{i+2}

Figure 4.2

The lines crossing side 3 which are not traced in one of the two trace sequences starting from side 3 correspond thus to lines with a θ-value in the range (135–180) or (0–45). These lines will be traced in trace sequences originating from other sides. For θ-values in the range (135–180) it is side 0, and for θ-values in the range (0–45) it is side 2. The θ-values considered in the trace sequences for each side are as follows.

side	ranges of θ considered
0	[0,45] and [135,180]
1	[45,90] and [90,135]
2	[0,45] and [135,180]
3	[45,90] and [90,135]

It is easy to see that every line (θ_i,ρ_j), $0 \le i \le k-1$, $0 \le j \le n-1$, is traced in at least one of the trace sequences. (Some lines are actually traced by two different trace sequences.) In the next section we show that the lines originating from side 3 can be traced in $O(n+k)$ time without congestion. Section 4.2 describes the overall algorithm and the accumulation of the partial results generated by each trace sequence.

4.1. Tracing Lines from Side 3

Let $\theta_{i0}, \theta_{i1}, \cdots, \theta_{iq}$ be the θ-values in the range [90,135] ordered such that $90 \le \theta_{i0} < \theta_{i1} < \cdots < \theta_{iq} \le 135$. Assume the traces of the parallel lines with θ-value θ_{i0} start from side 3 at time t_i. Then the traces of the parallel lines with θ-value θ_{ij} start at time t_i+j, where one time step is sufficiently large to perform the necessary computations outlined below.

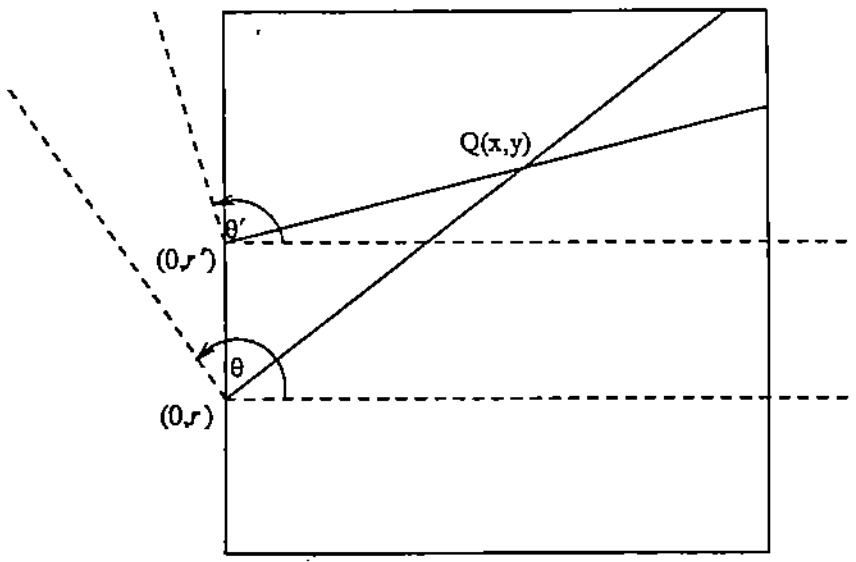
Assume first that the θ-values and the ρ-values are equally spaced and that s_θ and s_ρ is their step-size, respectively. Let t be an arbitrary time during the trace sequence and let θ be the θ-value considered for initiating a trace at time t. The value θ is known to all processors $(0,r)$. At the beginning of time step t every processor $(0,r)$ determines whether it needs to set up a packet for a trace. This is done by every processor computing the value $\rho = \lceil \frac{r \sin\theta}{s_\rho} \rceil$. If the line (θ,ρ) crosses cell $(0,r)$, processor $(0,r)$ creates a packet (θ,ρ,s) with $s=0$. Line (θ,ρ) crosses cell $(0,r)$ if $r-1 < y \le r$, where $y = \rho/\sin\theta$.

When the θ- and ρ-values are not spaced out equally, the values need to be "fed" to the processors in column 0. Supplying every processor $(0,r)$ at time step t with θ_{it} is straightforward. The organization of the ρ-values is more involved. Note that processor $(0,r)$ needs the ρ-values in the order $\rho_{n-1}, \cdots, \rho_0$ and that some ρ-values might be skipped over. The packets can still be set up within the same time bound, but with some overhead. Since this situation is not likely to arise in practise, we will not discuss it any further.

After the processors in column 0 have generated new packets, if necessary, every processor currently containing a packet determines where to send the packet next. Assume processor (c,r) contains a packet (θ,ρ,s). This packet may have been generated at time step t (in which case $c=0$) or at some earlier time. If processor (c,r) contains a 1-pixel, set s to $s+1$. The packet is then sent to a neighboring processor. If line (θ,ρ) crosses cell $(c+1,r)$, then the packet is sent to

processor $(c+1,r)$. Otherwise the packet is sent to processor $(c,r-1)$.

We need to show that routing the packets in this fashion is free of collisions. We do so by showing that, if two packets arrive at the same processor at the same time, they leave the processor using different links. Suppose packets (θ,ρ,s) and (θ',ρ',s') reach processor (i,j) at the same time and that, if they did meet at a processor at an earlier time, they did leave this processor on different links. Let $(0,r)$ and $(0,r')$ be the two processors that originated the two packets. We assume, w.l.o.g., that $r>r'$. We first state two simple properties about two lines. Recall that $90\leq\theta,\theta'\leq 135$.



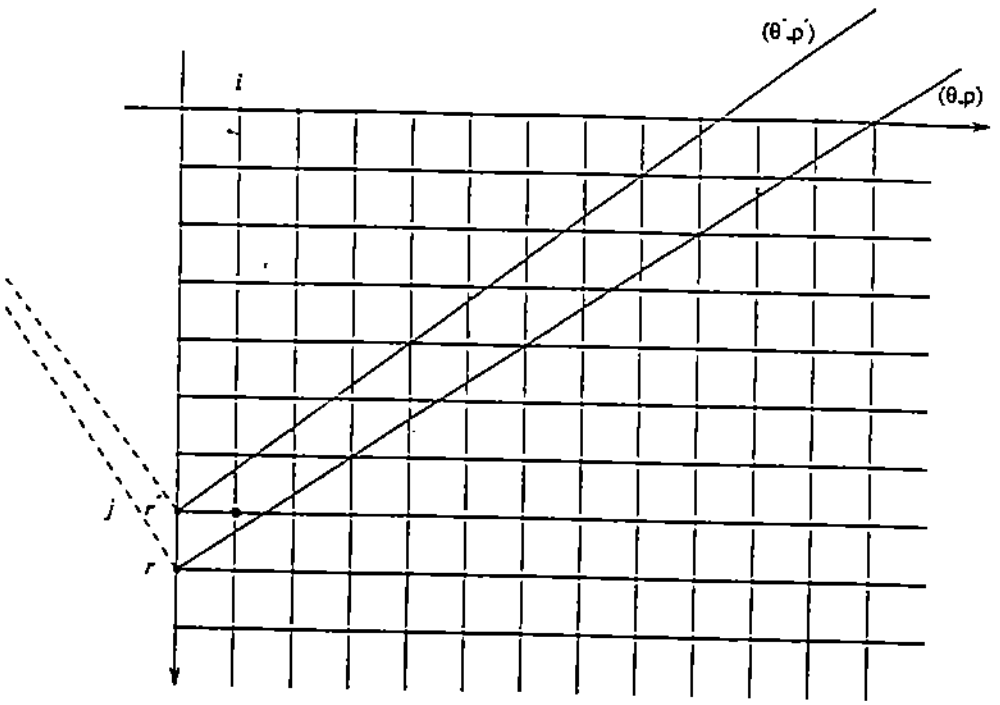
Lines (θ,ρ) and (θ',ρ') crossing in the image
Figure 4.3

Property 1. Let (θ',ρ') and (θ,ρ) be two lines that cross cell $(0,r')$ and $(0,r)$, respectively, with $r>r'$. The two lines intersect at a point $Q(x,y)$, $x,y>0$, in the plane if and only if $\theta'<\theta$. (See Figure 4.3.)

Property 2. A line (θ,ρ) cannot cross three consecutive vertically adjacent cells.

Consider again the two packets (θ,ρ,s) and (θ',ρ',s') that arrive at processor (i,j) at the same time. Since the length of the path (in Manhattan distance) from $(0,r')$ to (i,j) is smaller than the length of the path from $(0,r)$ to (i,j), packet (θ',ρ',s') was generated before packet (θ,ρ,s). Since our algorithm generates packets on column 0 with increasing θ-values, we have $\theta'>\theta$. Then, from Property 1, we can conclude that the two lines (θ,ρ) and (θ',ρ') do not cross in

the image. The situation looks similar to the one shown in Figure 4.4. By assumption the two packets reach processor (i,j) coming from different processors. Since we know that the two lines do not cross, packet (θ',ρ',s') must come from the horizontally adjacent processor $(i-1,j)$ and packet (θ,ρ,s) must come from the vertically adjacent processor $(i,j+1)$.



The trace of line (θ,ρ) is initiated at time t, the trace of (θ',ρ') at time $t'=t+1$, and the two lines meet at processor (i,j) at time $t+2$

Figure 4.4

We show next that the two packets leave processor (i,j) using different links. Because of Property 2, packet (θ,ρ,s) has to move to cell $(i+1,j)$. Suppose packet (θ',ρ',s') also wants to move to processor $(i+1,j)$. In this case line (θ',ρ') crosses the three cells $(i-1,j)$, (i,j), and $(i+1,j)$. The angle $\alpha' = \theta'-90$ must then satisfy each of the three following inequalities:

$$cotg^{-1}\frac{i}{r'-j} \le \alpha' \le cotg^{-1}\frac{i-1}{r'-j+1} \qquad (1)$$

$$cotg^{-1}\frac{i+1}{r'-j} \le \alpha' \le cotg^{-1}\frac{i}{r'-j+1} \qquad (2)$$

$$cotg^{-1}\frac{i+2}{r'-j} \le \alpha' \le cotg^{-1}\frac{i+1}{r'-j+1} \qquad (3)$$

By combining the above three relations, we obtain:

$$cotg^{-1}\frac{i}{r'-j} \le \alpha' \le cotg^{-1}\frac{i+1}{r'-j+1}.$$

Let $\alpha = \theta - 90$. Since line (θ,ρ) crosses cell (i,j) we also have:

$$cotg^{-1}\frac{i+1}{r-j} \leq \alpha$$

Since $r \geq r'+1$, it follows that:

$$\alpha' \leq cotg^{-1}\frac{i+1}{r'-j+1} \leq cotg^{-1}\frac{i+1}{r-j} \leq \alpha$$

Thus $\alpha' \leq \alpha$, a contradiction, and packet (θ',ρ',s') leaves processor (i,j) using the vertical link to processor $(i,j-1)$.

We observe that the collision of two packets at the same processor occurs rather infrequently. In fact, for two packets initiated from processors $(0,r)$ and $(0,r')$, $r>r'$, and reaching processor (i,j) at the same time, the following relation holds:

$$r \leq r'+2 \tag{4}$$

The inequality (4) is derived as follows. From (1) and (2) we have:

$$cotg^{-1}\frac{i}{r'-j} \leq \alpha' \leq cotg^{-1}\frac{i}{r'-j+1}$$

Similarly, for line (θ,ρ) we have:

$$cotg^{-1}\frac{i+1}{r-j} \leq \alpha \leq cotg^{-1}\frac{i}{r-j}$$

Since $\alpha<\alpha'$ then:

$$cotg^{-1}\frac{i+1}{r-j} < cotg^{-1}\frac{i}{r'-j+1}.$$

Let $r = r'+k$. Then,

$$cotg^{-1}\frac{i}{r'-j} > cotg^{-1}\frac{i}{i(k-1)-1}.$$

Thus for $k \geq 3$ we have $\frac{i}{i(k-1)-1} < 1$ and

$$cotg^{-1}\frac{i}{r'-j} > cotg^{-1}1 = 45$$

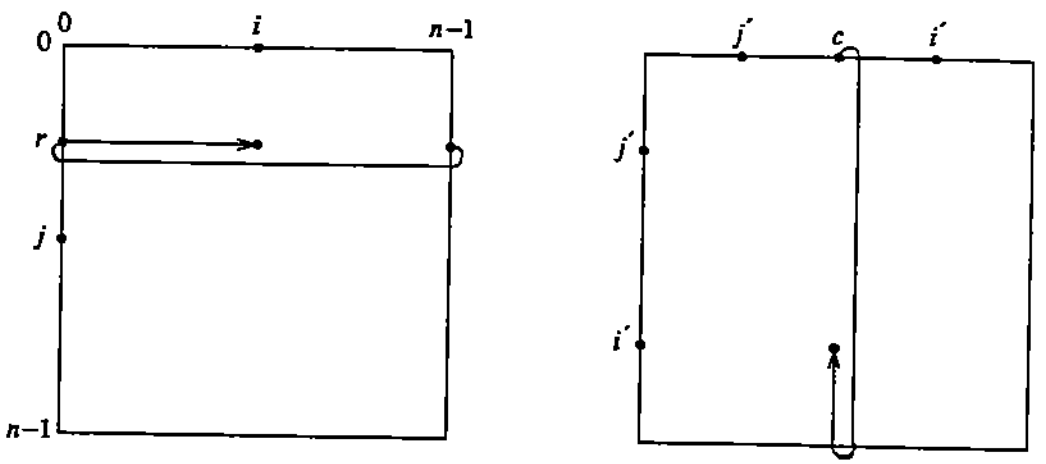
and $\alpha'>45$. This implies $\theta'>135$, a contradiction, and inequality (5) follows.

The second trace sequence starting from side 3 considers lines with θ-values in the range [45–90]. The trace is similar to the previous one, except that packets are generated from side 3 with decreasing θ-values. Since all the lines intersecting the remaining three sides can be thought of as obtained by rotations of the lines intersecting side 3, similar trace algorithms exist for the other three sides.

4.2. Accumulation of Partial Results

In this section we describe how the tracing algorithm places the packets (θ_i,ρ_j,s), where s is the number of 1-pixels on line (θ_i,ρ_j), $0\le i\le k-1$, $0\le j\le n-1$, at processor (i,j). In a tracing sequence starting from side u, $0\le u\le 3$, a packet (θ_i,ρ_j,s) reaches the opposite or an adjacent side of side u with the final value of s and needs to be routed to its processor (i,j). We assume that the mesh has vertical and horizontal wrap-around connections as available on the MPP. This assumption holds without loss of generality, since a mesh with wrap-around is, in the asymptotic sense, no more powerful that one without wrap-around connections.

Consider the trace sequence which starts at side 3 and traces the lines with $90\le\theta\le135$. Let $(n-1,r)$ be any processor on side 1. Since no two packets reaching processor $(n-1,r)$ have the same θ-value, at most n packets can reach processor $(n-1,r)$ during the trace sequence. Assume packet (θ_i,ρ_j,s) reaches processor $(n-1,r)$. The algorithm first routes the packet to processor (i,r) which is called its *intermediate processor*. When routing to the intermediate processor the packet stays on row r. If $i\neq n-1$, the packet uses the horizontal wrap-around connection to processor $(0,r)$ and moves right until it reaches column i. This movement is shown in Figure 4.5(a). We call a packet that has reached its intermediate processor in this fashion an *i-packet*.



(a) (i,r) is the intermediate processor (b) (c,i') is the intermediate processor

Figure 4.5

Consider next a processor $(c,0)$ on side 0. In the trace, the packets reaching processor $(c,0)$ have again distinct θ-values (and hence there can be at most n of them). It is easy to see that the

ρ-values of the packets reaching $(c,0)$ need not be distinct. Hence, routing a packet $(\theta_i,\rho_{j'},s')$ reaching processor $(c,0)$ to processor (c,j') can lead to congestions. We solve this problem by making processor (c,i') the intermediate processor of the packet. It is reached by vertical movements after using the vertical wrap-around of column c. See Figure 4.5(b). We call a packet that has reached its intermediate processor (c,i') a *t–i–packet*. The 't' stands for 'transpose'. The t-i-packet will later be routed to processor (j',i') and a transpose operation will send the packet to its final processor.

The routing of packets to their intermediate processor occurs simultaneously with the tracing of the lines. After all lines have been traced, the algorithm completes the routing to intermediate processors which takes at most n additional time steps. At this point a processor in the mesh may contain up to two packets, one i-packet received from side 1 and one t-i-packet received from side 0. These packets remain in this processor until all eight traces have been completed. It is easy to see that when performing the all eight trace sequences a processor can receive at most one i-packet and one t-i-packet. Consider, for example, the trace sequence from starting from side 3 with $90\leq\theta\leq135$. An i-packet (θ_i,ρ_j,s) finished its trace at some processor $(n-1,r)$ on side 1. The lines parallel to line (θ_i,ρ_j) that do not cross side 3 are traced in a trace sequence starting from side 1. After the packets of these lines reach side 2, they are routed to intermediate processors and stored as t-i-packets.

After all eight traces have been completed, the algorithm performs the final routing steps. Consider first the i-packets. Column i contains i-packets with a θ-value of θ_i, but not ordered according to ρ-values. Ordering the ρ-values can easily be done in n time steps. Similarly, the t-i-packets are arranged so that row i' contains all the packets with a θ-value of $\theta_{i'}$ ordered by ρ-values. The final operation is to perform a transpose on the t-i-packets which can be done in $2n$ time steps [U]. This completes the description of our linear time tracing algorithm.

5. Conclusions

We presented two algorithms, the block and the tracing algorithm, which perform the Hough transform for an $n\times n$ image on an $n\times n$ mesh of processors. While the asymptotic running time of the tracing algorithm is better than the one of the block algorithm, the constant asso-

ciated with the block algorithm appears to be smaller. We plan to implement and compare the performance of both algorithms on the MPP, a mesh of size 128×128.

The algorithms can easily be generalized to handle numbers of θ- and ρ-values different from the ones used in our description. The algorithms can also be adapted to count the number of 1-pixels on a line (θ,ρ) of width w, where a line (θ,ρ) of width w consists of all the cells at most distance $w/2$ away from the cells crossed by the line (θ,ρ).

References

[AH] Atallah, M. J., Hambrusch S. E., "Solving tree problems on a mesh-connected processor array", *Information and Control*, Vol. 69, pp. 168-187, 1986.

[BA] Ballard, D. H., "Generalizing the Hough transform to detect arbitrary shapes", *Pattern Recognition*, Vol. 3, Nr. 2, pp.11-122, 1981.

[BT] Batcher, K. E., "Design of a massively parallel processor", *IEEE Trans. on Comp.*, C-29, Nr. 9, pp. 836-840, 1980.

[CL] Chuang, H. Y., Li, C.C., "A systolic array for straight line detection by modified Hough transform", *IEEE Workshop on Comp. Archit. for Pattern Analysis and Image Data Base Management*, pp. 300-304, 1985.

[DH] Duda, R. O., Hart, P.E., "Use of the Hough transformation to detect lines and curves in pictures", *CACM*, Vol. 15, Nr. 1, 1972.

[FH] Fisher, A. L., Highnam, P. T., "Real-time image processing on scanline array processors", *IEEE Workshop on Comp. Archit. for Pattern Analysis and Image Data Base Management*, pp. 484-489, 1985.

[FR] Freeman, H., "Computer processing of line drawing images", *Computing Surveys*, 6, 1, pp. 57-98, 1974.

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

[IK] Ibrahim, H. A. H., Kender, J. R., and Shaw, D. E., "The analysis and performance of two middle-level vision tasks on a fine-grained SIMD tree machine", *Proc. of IEEE Comp. Soc. Conf. on Computer Vision and Pattern Recognition*, pp. 248-256, 1985.

[KW] Kung, H. T., Webb, J., "Mapping image processing operations onto a linear systolic machine", *Distributed Computing*, pp. 246-257, 1986.

[KR] Kushner, T., Wu, A. Y., and Rosenfeld, A. , "Image processing on MPP", *Pattern Recognition*, Vol. 15, Nr. 3, pp. 121-130, 1982.

[MS] Miller R., Stout Q., "Geometric Algorithms for Digitized Pictures on a Mesh-Connected Computer", *IEEE Trans. on PAMI*, pp. 216-228, 1985.

[NS] Nassimi D., Sahni S., "Finding connected components and connected ones on a mesh-connected parallel computer", *SIAM J. on Computing*, Vol. 9, pp. 744-757, 1980.

[RE] Reeves, A. P., "The Massively Parallel Processor: a highly parallel scientific computer", manuscript, 1986.

[RM] Reeves, A. P., Moura C. H., "Data manipulations on the Massively Parallel Processor", *Proc. of 19-th Hawaii Internat. Conf. on System Sciences*, pp. 222-229, 1986.

[RW] Rothstein, J., Weiman, C., "Parallel and sequential specification of a context-sensitive language for straight lines on grids", *Computer Graphics and Image Processing*, 5, pp. 106-124, 1976.

[SD] Sanz, J. L. C., Dinstein, I., " Projection-based geometrical feature extraction for computer vision: algorithms in pipeline architectures", *IEEE Trans. on PAMI*, Vol. PAMI-9, Nr. 1, pp. 160-168, 1987.

[SI] Silderberg, T. M., "The Hough transform on the Geometric Arithmetic Parallel Processor", *IEEE Workshop on Comp. Archit. for Pattern Analysis and Image Data Base Management*, pp. 387-391, 1985.

[UL] Ullman, J. D., *Computational Aspects of VLSI*, Computer Science Press, 1984.