Jing-Fu Jenq and *Sartaj Sahni*
University of Minnesota University of Florida

Abstract

We develop parallel algorithms to compute the Hough transform on a reconfigurable mesh with buses (RMESH) multiprocessor. The p angle Hough transform of an $N \times N$ image can be computed in $O(p\log(N/p))$ time by an $N \times N$ RMESH, in $O((p/N)\log N)$ time by an $N \times N^2$ RMESH with N copies of the image pretiled, in $O((p/\sqrt{N})\log N)$ time by an $N^{1.5} \times N^{1.5}$ RMESH, and in $O((p/N)\log N)$ time by an $N^2 \times N^2$ RMESH.

Keywords and Phrases
Hough transform, reconfigurable mesh with buses, parallel algorithms, complexity.

1. Introduction

Let (x_i, y_i) be any point on a straight line L in two dimensional space (Figure 1). The normal to L is a straight line that orignates at the origin (0,0), terminates at a point on L, and is perpendicular to L. Let θ be the angle between the normal and the x-axis and let r be the length of the normal. From Figure 1, we see that regardless of the position of (x_i, y_i) on L, the following equality holds:

$$x_i\cos\theta + y_i\sin\theta = r \qquad (1)$$

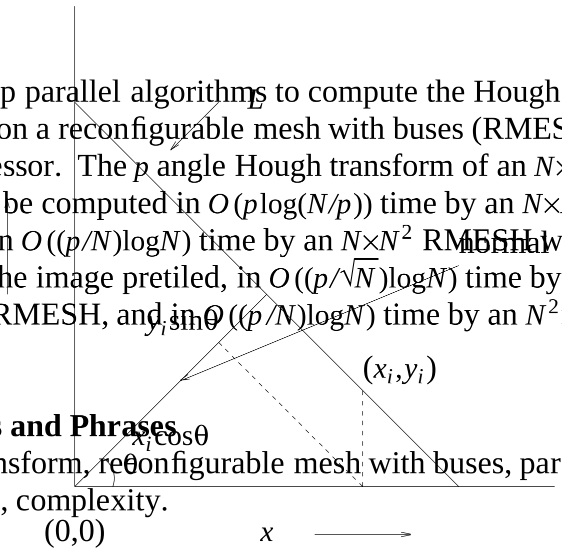


Figure 1 A line L and its normal

This equality may be used to detect straight lines or edges in images. In the Hough transform method this is done by trying out a set $\{\theta_j \mid 0 \le j < p\}$ of p possible angles for the normal. An angle θ_j and point (x_i, y_i) uniquely define the line L and its normal. For any angle θ_j all image points that have the same normal length r, as computed by Equation (1) lie on the same line L. If for a given θ_j we know how many image points have the same normal length r, we can determine the probability that these points actually define an edge of the image. Lines with many points on them have a higher likelihood of defining an edge in the image than those with fewer points. Let $I[0..N-1, 0..N-1]$ be an $N \times N$ image such that $I[i,j] = 1$ iff the point (i,j) is a candidate for an edge point of the image. $I[i,j] = 0$ otherwise. The p angle Hough transform of I is an array H such that:

$$H[r,j] = \left|\{(x,y) \mid r = \lfloor x\cos\theta_j + y\sin\theta_j \rfloor \text{ and } I[x,y] = 1\}\right|$$

$$\theta_j = \frac{\pi}{p}(j+1)$$

The second coordinate of H, j, corresponds to the p angles and is in the range 0 through $p-1$. Since $\theta_j = \frac{\pi}{p}(j+1)$, and $0 \le j < p$, $0 < \theta_j \le \pi$. Furthermore, since the image point coordinates x and y are in the range $0 \le x, y < N$, $r = \lfloor x\cos\theta_j + y\sin\theta_j \rfloor$ is in the range $-\sqrt{2}N$ through $\sqrt{2}N$. Hence H is at most a $2\sqrt{2}N \times p$ array. Pairs (r, j) for which $H[r,j]$ is greater than some threshold value define likely edges. The pair (r, j) defines the unique line L whose normal has length r and angle θ_j.

On a single processor computer, the Hough transform is easily computed in $O(N^2 p)$ time. Parallel algorithms for mesh connected computers have been proposed by Rosenfeld et al. [ROSE88], Cypher et al. [CYPH87], Guerra and Hambrush [GUER87], and Silberberg [SILB85]. The algorithm of Cypher et al. uses a pipelined technique and has complexity $O(N+p)$ on an $N \times N$ mesh. Fisher and Highnam [FISH87] consider a scan line array. Their algorithm has time complexity $O(N^2 p)$ and is suitable for VLSI implementation. The processor array size is $O(N)$ and each PE requires $O(p)$ space. Ranka and Sahni [RANK90] develop two $O(p + \log N)$ SIMD hypercube algorithms to compute H. Both of these use an N^2 processor SIMD hypercube. One uses $O(1)$ memory per processor while the other uses $O(\log N)$ memory per processor. They also develop algorithms for an MIMD hypercube and present experimental results on an NCUBE hypercube. The

* This research was supported in part by the National Science Fundation under grants DCR-84-20935 and MIP 86-17374

computation of the Hough transform on an SIMD tree machine is considered by Ibrahim et al. [IBRA86]. Rather than deal in (r,θ) space, their work uses the (m,c) space where m is the slope and c is the y-axis intercept of the line (i.e., the straight line equation $y = mx + c$ is used). A Hough transform algorithm for a polymorphic torus is developed in [LI89, MARE88, and MARE89] and a fast Hough transorm algorithm is given in [LI86].

In this paper we consider a variant of the mesh connected computer. This variant called "reconfigurable mesh with buses" (RMESH) was introduced by Miller, Prasanna Kumar, Resis, and Stout [MILL88abc]. We develop algorithms to compute the p angle Hough transform of an $N \times N$ image on different size RMESHs. Our algorithm for an $N \times N$ RMESH has complexity $O(p\log(N/p))$ which is a significant improvement over the $O(p + N)$ complexity for an $N \times N$ mesh when $p \ll N$. On an $N \times N^2$ RMESH we can compute the Hough transform in $O((p/N)\log N)$ time with N copies of the image pretiled over the RMESH, and in times $O(p/\sqrt{N}\log N)$ and $O((p/N)\log N)$ on $N^{1.5} \times N^{1.5}$ and $N^2 \times N^2$ RMESHs, respectively.

2 The RMESH Model

The important features of an RMESH are [MILL88abc]:

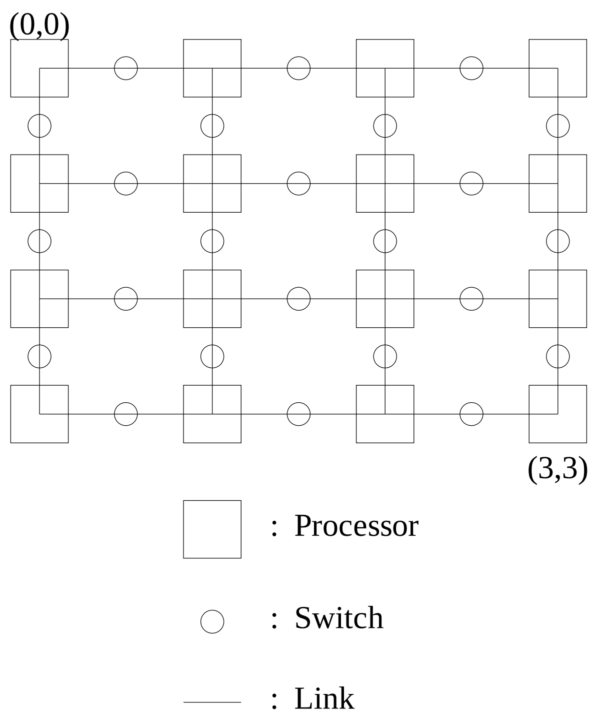


Figure 2 4×4 RMESH

1 An $N \times M$ RMESH is a 2-dimensional mesh connected array of processing elements (PEs). Each PE in the RMESH is connected to a broadcast bus which is itself constructed as an $N \times M$ grid. The PEs are connected to the bus at the intersections of the grid. A 4×4 RMESH is shown in Figure 2. Each processor has up to four bus switches that are software controlled and that can be used to reconfigure the bus into subbuses. The ID of each PE is a pair (i,j) where i is the row index and j is the column index. The ID of the upper left corner PE is (0,0) and that of the lower right one is $(N-1, M-1)$.

2 The up to four switches associated with a PE are labeled E (east), W (west), S (south) and N (north). Notice that the east (west, north, south) switch of a PE is also the west (east, south, north) switch of the PE (if any) on its right (left, top, bottom). Two PEs can simultaneously set (connect, close) or unset (disconnect, open) a particular switch as long as the settings do not conflict. The broadcast bus can be subdivided into subbuses by opening (disconnecting) some of the switches.

3 Only one processor can put data onto a given sub bus at any time

4 In unit time, data put on a subbus can be read by every PE connected to it. If a PE is to broadcast a value in register I to all of the PEs on its subbus, then it uses the command broadcast(I).

5 To read the content of the broadcast bus into a register R the statement R := content(bus) is used.

6 Row buses are formed if each processor disconnects (opens) its S switch and connects (closes) its E switch. Column buses are formed by disconnecting the E switches and connecting the S switches.

7 Diagonalize a row (column) of elements is a command to move the specific row (column) elements to the diagonal position of a specified window which contains that row (column). This is illustrated in Figure 3.

3 N^2 Processor RMESH

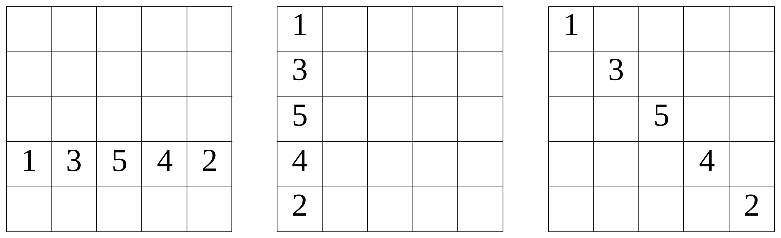
As is the case in the algorithms of [CYPH87], [GUER87], and [RANK90], our N^2 processor RMESH algorithm divides the p angles into four classes C1-C4 as below:

C1 = $\{\theta_j \mid 0 < \theta_j \le \pi/4\}$

C2 = $\{\theta_j \mid \pi/4 < \theta_j \le \pi/2\}$

C3 = $\{\theta_j \mid \pi/2 < \theta_j \le 3\pi/4\}$

C4 = $\{\theta_j \mid 3\pi/4 < \theta_j \le \pi\}$



(a) 4th row (b) 1st column (c) diagonalize

Figure 3 Diagonalize 4th row or 1st column elements of a 5×5 window

The algorithms for each of these classes are quite similar. So, we provide the details for just one of these, i.e., C3. The number of angles in each class is $q = p/4$ (for simplicity, we assume that 4 divides p). For any θ_j we may define a matrix V of normal vector lengths for lines that go through points (a,b), $0 \le a,\ b < N$ and whose normal angle is θ_j. This matrix is defined as below:

$$V[a,b] = \lfloor a\cos\theta_j + b\sin\theta_j \rfloor,\ 0 \le a,b < N$$

For any $N \times N$ image I, the j'th column, $H[*,j]$, of the Hough transform matrix can be computed using the equality

$$H[r,j] = |\{(a,b)|\ V[a,b] = r \text{ and } I[a,b] = 1\}|$$

We first consider some properties of V for the case $\pi/2 < \theta_j \le 3\pi/4$. Figure 4 shows a line L whose normal angle is in this range. The following properties are easily established [RANK90, CYPH87, and GUER87].
P1: If $V[a,b] = V[a,b+z]$ for any $z > 0$, then $z = 1$.
P2: If $V[a,b] = V[a+1,c]$, then $c = b$ or $c = b+1$.
P3: If $V[a,b] = V[a,b+1] = V[a+1,c]$, then $c = b+1$.
P4: If $V[a,b] \ne V[x,b+1]$ for $x > a$, then $V[a,b] \ne V[x,y]$ for $y > b$.Suppose we consider computing $H[r,j]$ for a fixed r and j such that $\theta_j \in$ C3 by sending a token through every point (a,b) such that $V[a,b] = r$. This token begins with the value 0 and is incremented by 1 each time it visits a point (a,b) with $I[a,b] = 1$. Since $I[a,b] \in \{0,1\}$ we may simply increment the token by $I[a,b]$ each time it reaches a point (a,b) with $V[a,b] = r$. Since for every r the corresponding line L must cross the left or bottom boundaries of the image and since $|\cos\theta|$ and $|\sin\theta|$ are in the range $[0,1]$, $V[0..N-1,0]$ and $V[0,0..N-1]$ cover the range of possible r values in

$V[0..N-1, 0..N-1]$.

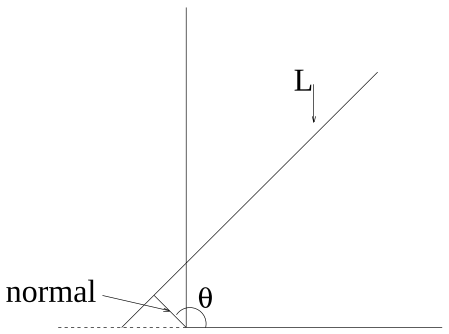


Figure 4 A line L with normal angle in the range $[\pi/2, 3\pi/4]$

Hence we may start our token for $H[r,j]$ at a left or bottom point (a,b) such that $V[a,b] = r$. From properties P1 - P4 it follows that the token needs to move according to the rule given in Figure 5.

```
Let $(a,b)$ be the current position of the token.
Let $r = \lfloor a\cos\theta_j + b\sin\theta_j \rfloor$.
**if** $r = \lfloor a\cos\theta_j + (b+1)\sin\theta_j \rfloor$
**then** move the token to (a,b+1)
**else if** $r = \lfloor (a+1)\cos\theta_j + b\sin\theta_j \rfloor$
    **then** move the token to $(a+1,b)$
    **else** move the token to $(a+1,b+1)$
```

Figure 5 Rule to move a token

The token is moved until it falls off the image. This will happen after the token reaches either the top or right boundary of the image. At this time the token's value is $H[r,j]$.

Rather than send a single token through the array, we can simultaneously send several. The discipline we adopt is that at any time all active tokens in a column correspond to the same angle θ_j. A token becomes inactive when a move according to the rule of Figure 5 would cause it to fall off the image. For convenience, we assume that the processors in the $N \times N$ RMESH are indexed such that PE (i,j) corresponds to the image point (i,j) (see Figure 6).

Let us consider the token movement strategy for the case of a single angle θ_j, $\pi/2 < \theta_j \leq 3\pi/4$. This is described in Figure 7. We assume one processor per pixel. The tokens for θ_j begin

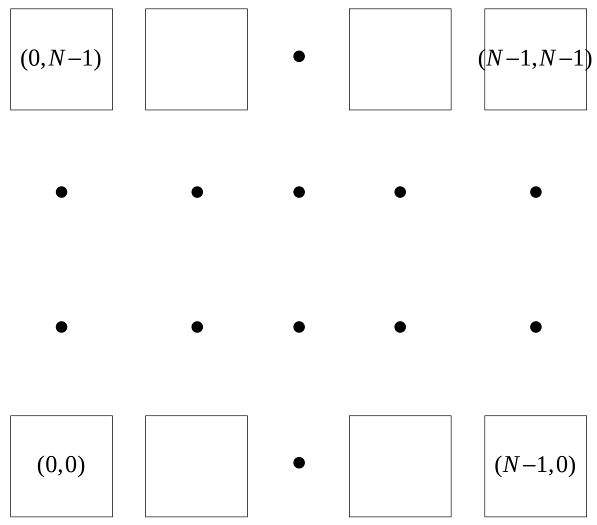


Figure 6 PE indexing scheme

in column 0. Each token corresponds to a distinct r value. From P1 we know that two image points in the same column can have the same r value only if they are adjacent. In step 1 of Figure 7 one token for each unique r in column 0 is created. These newly created tokens have the value 0. In step 2 the tokens are moved through all points (a,b) with the same $V[a,b]$ value. After incrementing the token values to account for the image values at their current locations in column k, the tokens that also correspond to the image point one up in the same column are moved one up and incremented. In case this requires the token in row $N-1$ to move, this token is deactivated as the move would cause it to fall off the image. Because of property P1, the tokens in column k are not to be moved to any other position in the column.

The tokens now need to be moved to column $k+1$ (unless $k = N-1$). A token in PE (k,a) moves to either $(k+1,a)$ or $(k+1,a+1)$. This is resolved by computing the r value for position $(k+1,a)$. At column $k+1$ it is possible that position $(k+1,0)$ corresponds to a new line. This is true if and only if PE $(k+1,0)$ does not receive a token from column k. In this case this PE initializes a new token with value 0.

Following step 2 we have at most one deactivated token in each of the PEs in row $N-1$ (i.e., top row of PEs) and at most one active token in each of the PEs in column $N-1$ (i.e., right most column of PEs). These tokens correspond to distinct r values and define the column of the Hough transform matrix that corresponds to θ_j.

The computations for all angles θ_j, $\pi/2 < \theta_j \le 3\pi/4$, can be done in a pipelined fashion. Following the movement of the

```
Step 1
[ Initialize column 0 tokens ]
PE $(0,i)$ creates a token with value 0 if
$\lfloor i\sin\theta_j \rfloor \neq \lfloor (i-1)\sin\theta_j \rfloor$, $1 \leq i < N$
PE $(0,0)$ creates a token with value 0

Step 2
[ Update and move tokens ]
for $k := 0$ to $N-1$ do
begin
  {tokens are in column $k$ }
  the PEs in column $k$ that have a token, add their $I$ value to it;
  { move some tokens up the column by 1 }
  PE $(k,i)$ determines if $\lfloor k\cos\theta_j + i\sin\theta_j \rfloor = \lfloor k\cos\theta_j + (i+1)\sin\theta_j \rfloor$.
  This is done by all PEs in column $k$ that have a token.
  If the equality holds, the PE sends its token to PE $(k,i+1)$
  unless $i+1 = N$.
  In this latter case PE $(k,i)$ saves the token as a deactivated
  token.
  { Increment moved tokens }
  Each PE $(k,i)$ in column $k$ that received a token adds its $I$ value
  to it;
  { token updating for column $k$ has been completed }
  { advance tokens to next column }
  if $k \neq N-1$ then
  begin
    every PE $(k,i)$ that has an active token determines if
    $\lfloor k\cos\theta_j + i\sin\theta_j \rfloor = \lfloor (k+1)\cos\theta_j + i\sin\theta_j \rfloor$.
    If so, it sends its token to PE $(k+1,i)$.
    Otherwise it sends it to PE $(k+1,i+1)$ except when $i+1 = N$.
    In this latter case the token is saved as a deactivated token by
    PE $(k,i)$.
    If PE $(k+1,0)$ does not receive a token, it creates one with
    value 0;
  end;
end;
```

Figure 7 Token movement and updating for angle θ_j

tokens for θ_j from column 0 to column 1, column 0 can initiate the tokens for the next angle θ_j. The scheme of Figure 7 is easily modified so that the PEs in a column know θ_j (or $\cos\theta_j$ and $\sin\theta_j$) for the tokens they currently hold. With this pipelining the Hough transform may be computed in $O(N+p)$ time. This is essentially the strategy of [CYPH87] and [GUER87].

The performance can be improved by employing the above strategy on $N\times(p/4)$ sub RMESH's only. Recall that we have assumed that the number of angles in each of C1, C2, C3, and C4 is $p/4$. We consider the $N\times N$ image as $4N/p$ independent $N\times(p/4)$ subimages and compute the Hough transform for each independently. Then the $4N/p$ Hough transforms are combined to get the Hough transform for the original $N\times N$ image (actually this will only get us the transform for angles in C3; similar algorithms need to be run to get the Hough transform for the remaining angles).

Assume that following the application of Figure 7, in a pipelined fashion, for all θ_j in C3, the Hough transform matrix is stored in columns 0 through $p/4-1$ of each $N\times(p/4)$ sub RMESH. For this, when the tokens for the j'th angle reach column $p/4-1$ of the sub RMESH, they are broadcast along row buses to the processors in column j of the sub RMESH. Also, when tokens get deactivated in row $N-1$ they are transmitted to processors in columns dedicated for their angles (column j processors of each sub RMESH are dedicated to the j'th angle in C3). Note that the tokens that get simultaneously deactivated in row $N-1$ correspond to different angles in C3 as at most one token deactivates in each row $N-1$ processor at any time and each column corresponds to a different angle. The deactivated token (if any) in processor $(k,N-1)$ of the sub RMESH is routed to processor (j,k) of the sub RMESH where θ_j is the angle corresponding to the token. This is accomplished in $O(1)$ time as in Figure 8.

Once the computation for all $p/4$ angles in C3 has been completed, each PE in column j of each $N\times(p/4)$ sub RMESH contains at most two token values. One received from the rightmost column in the sub RMESH and one from row $N-1$. The first is an active token and the latter a deactivated token. All tokens in column j of the sub RMESH correspond to the j'th angle in C3. The time needed to accomplish this is $O(p)$.

Now we need to combine together the partial Hough transform values computed in each $N\times(p/4)$ sub RMESH. Column j of each sub RMESH contains partial Hough transform values for the j'th angle in C3, $0\le j<p/4$. Across these columns, we need to add together values that correspond to the same r. Each processor has at most two tokens: active and deactivated. There are two quantities associated with each token. One is the r value and the other is a count of the number of pixels that have contributed to this token (this count has so far been referred to as the token value). Let us call these quantities *token.r* and *token.count*, respectively. The sum of the *token.count*'s for the same angle and *token.r* values can be obtained in $O(p\log(N/p))$ time by computing these sums for one angle at a time. This corresponds to considering all

Step 1
Set up column buses
Step 2
Each PE $(k,N-1)$ of the sub RMESH broadcasts its deactivated token together with the corresponding r and j values;
Step 3
PEs (i,i) of the sub RMESH, $0 \le i < p/4$ read their bus and are now the only PEs in the sub RMESH with deactivated tokens;
Step 4
Set up row buses local to each sub RMESH;
Step 5
The PEs with deactivated tokens broadcast tokens and the corresponding r and j values;
Step 6
All PEs read their bus. However, a PE stores the deactivated token value and r value read only if the PE is in column j of the sub RMESH (j is the third value read from the bus);

Figure 8 Redistributing deactivated tokens

columns j with $j \ mod \ (p/4) = k$ for a fixed k in $(0,p/4-1)$ at the same time and adding together the *token.count*s in these columns for tokens that have the same r values. This summation is done in $O(log(N/p))$ time by first summing up pairs of columns in adjacent blocks; then pairs of these results are summed, etc. Figure 9 shows the strategy for the case of 8 blocks each of size $N\times(p/4)$. The column j tokens of each block are to be summed. The leaves of the summation tree are labeled by the block number they represent. Blocks of the RMESH are numbered left to right 0 through $4N/p-1$. The input for each summation consists of two columns of tokens. The columns are initially $p/4$ processors apart; then, at the next level, they are $p/2$ processsors apart, then p; then $2p$; and so on. One of the two token columns is to the left of the other. This is called the L column and the other column is called the R column. On input, each processor contains at most two tokens; one active and one deactivated. The output of the summation operation is left in the input processor column corresponding to L. Again, each processor in this column will have at most two tokens; one active and the other deactivated. Furthermore the r values corresponding to the deactivated tokens (active tokens) decrease as we go down a column and the deactivated tokens have a larger r value than do the active tokens. When the columns being merged are $p/4$ apart (i.e., leaves of Figure 9), the sets of deactivated and active tokens

are as defined earlier. For a pair of columns L and R, these sets at the parent node (cf. Figure 9) are given by the equalities:
deactivated $(L \cup R)$ = deactivated $(L) \cup$ deactivated(R)
active $(L \cup R)$ = active (R)

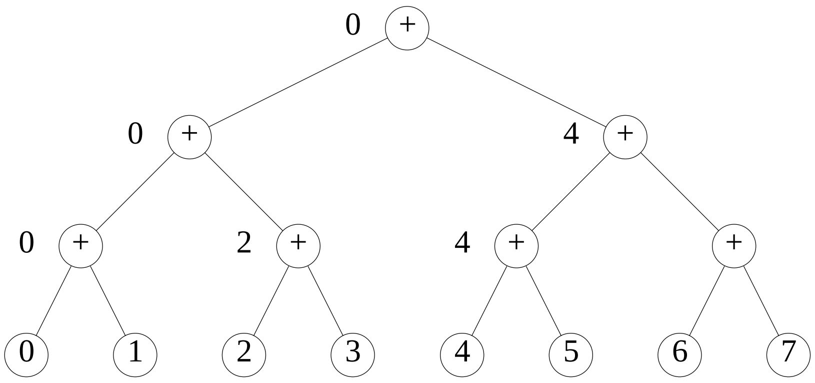


Figure 9 Summing the *j*'th column of 8 blocks

Let Z be a summation node (i.e., internal node) of Figure 9. Let the distance between the two columns L and R being summed at Z be s, $s \in \{p/4, p/2, \cdots, N/2\}$ It is easy to see that $|deactivated(Z)| \le 2s \le N$ and $|active(Z)| \le N$. We assume that each token has two values: *count* and r associated with it. *Count* is the number of pixels that has contributed to it and r is the length of the normal to the line represented by this token. We observe that the *count* value of the deactivated tokens of L does not change as a result of the summation. In fact the *count* values can change only for those tokens that are deactivated or active tokens of R. To get the new count values for the deactivated tokens of R, we use the processor columns L and R together with the $s-1$ columns between them. Thus an $N \times (s+1)$ sub RMESH is used. The code for such a sub RMESH is given in Figure 10. Its complexity is O(1). The deactivated tokens of R can next be compacted to lie in consecutive rows of L immediately following the last row of L that contains a deactivated token. This requires us to rank the deactivated tokens of R and then route these to row w of L where w is the token rank plus the number of deactivated tokens already in L. We assume that the deactivated tokens of L lie in consecutive rows of L beginning at row 0. This is not true for the leaf nodes of Figure 9. However, the deactivated tokens in these nodes may be compacted in O(1) time using the ideas used to route the deactivated tokens of R to L. The

```
Step 1
  Use column and row buses in the sub RMESH to obtain the
  data configuration:
  PE $(i,j)$ of the $N\times(s+1)$ sub RMESH contains the deactivated
  token (if any) from the row $i$ processor of column $R$ and the
  active token (if any) from the row $j$ processor of column $L$,
  $0 \le i < s+1$, $0 \le j < N$.
Step 2
  if  PE $(i,j)$ of the sub RMESH has two tokens and both have the
        same $r$ value
  then  update the count of the deactivated tokens to be the sum
        of the two token counts and destroy the active token of $L$
        by sending a signal down row bus $j$,
        $0 \le i < s+1$, $0 \le j < N$.
Step 3
  if  PE $(i,j)$ has a deactivated token with updated count
  then the updated count is reported back to the PE in row $i$
        of column $R$.
```

Figure 10 Updating the count for deactivated tokens of R

```
Step 1
  if PE $(R,k)$ has a token
  then it broadcasts it and the token's rank on its row bus,
        $0 \le k < s$ PE $(i,j)$ reads its bus, $0 \le i < (s+1)$, $0 \le j < s$.
Step 2
  PE $(i,j)$ retains the token (if any) read in Step 1 only if $i$
  equals the token rank, $0 \le i < s+1$, $0 \le j < s$.
Step 3
  The PEs that have tokens, broadcast these along column buses.
Step 4
  Let $t$ be the number of deactivated tokens in column $L$.
  PE $(j,t+j)$, $0 \le j < q$ where $q$ is the number of deactivated
  tokens in $R$, reads its bus.
Step 5
  PE $(j,t+j)$, $0 \le j < q$ broadcasts the token read on its row bus.
Step 6
   PE $(L,t+j)$, $0 \le j < q$ reads its bus.
```

Figure 11 Routing deactivated tokens of R to L

ranking and routing of the deactivated tokens of R can be done in O(1) time using the $N\times(s+1)$ processor sub RMESH comprised of columns L and R and the $s-1$ processor columns between them. The ranking takes $O(1)$ time (Appendix A) as we are ranking at most s tokens, one token to a row, in an $s\times(s+1)$ sub RMESH (the deactivated tokens lie in the first s rows of R). The routing scheme is described in Figure 11.

Step 1
PE (R,i) broadcasts its token's (if any) count and r values to all processors on row i using a row bus, $0 \le i < N$
Step 2
PE (L,i) broadcast its token's (if any) count and r values to PEs $(L+q,i+q)$, $0 \le q \le \min\{N-1-i,2s\}$ using the bus structures of Figure 13, $0 \le i < N$
Step 3
Set up row buses.
Step 4
Each PE (i,j) in the $N\times(2s)$ sub RMESH that has two tokens with the same r value adds their count and disconnects its W switch and then broadcasts the new count and r value to PE (R,j).
Step 5
PE (R,j) broadcast its r and *count* values, $0 \le j < N$.
Step 6
PE (L,j) reads r and *count* from the row bus, $0 \le j < N$

Figure 12 Updating counts for active tokens of R

To update the count of the active tokens in R a different strategy is used. First note that the set of active tokens in R includes the remaining active tokens of L (i.e., those not destroyed in step 2 of Figure 10). Also, note that as tokens progress through the image (see algorithm of Figure 5), they can move up by at most two rows for each column they move right. Hence, if a token of L has the same r value as a token in R, then the two tokens must reside in rows of L and R that are at most $2s$ apart (recall that L and R are s columns apart). To update the active tokens of R, we employ the PEs in the two blocks which L and R belong to by moving L to the left most column in its block and R to its right most column in its block. So, a total of $2s$ processor columns (i.e., an $N\times(2s)$ sub RMESH) are used. For the update, we need to bring active token pairs from L and R that have the same r value together. This is done using the strategy of Figure 12. We assume that L and R are the 2 extreme columns of the combined block. The broadcasting of step 2 is done in two stages. First, all PEs

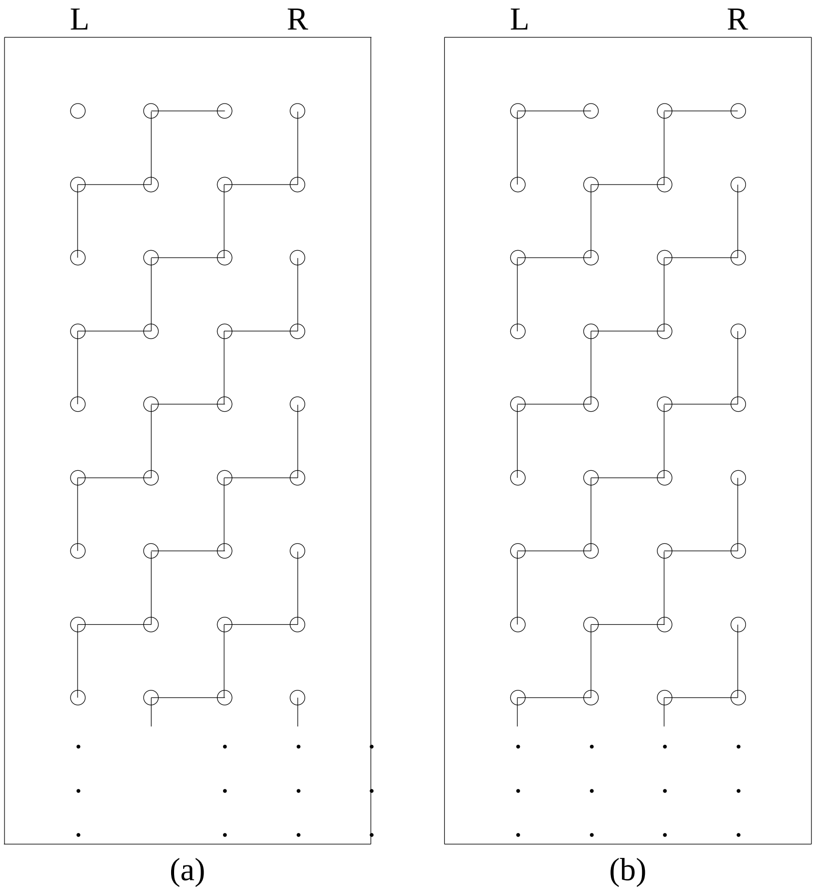


Figure 13 Bus structures used in step 2 of Figure 12

$[L,i]$ with i odd do this and then those with i even do it. Figure 13 (a) gives the bus structure used for the first stage and Figure 13 (b) gives the structure used in the second stage.

The complexity of the algorithm described to compute the Hough transform for the angles in C3 is $O(p\log(N/p))$. The time needed for the angles in each of the remaining sets C1, C2, and C4 is the same. So, the over all complexity of our N^2 processor RMESH Hough transform algorithm is $O(p\log(N/p))$.

4 N^3 Processor RMESH

If the RMESH is configured as an $N \times N^2$ array with one copy of the image in each $N \times N$ sub RMESH, then the Hough transform can be computed in $O((p/N)\log N)$ time by having each $N \times N$ sub MESH compute the transform for p/N angles using the algorithm of Section 2. In case the RMESH is configured as a $N^{1.5} \times N^{1.5}$ array with one pixel in each $\sqrt{N} \times \sqrt{N}$ sub MESH, the Hough transform can be computed in $O((p/\sqrt{N})\log N)$ time. For this, $p/\sqrt{N}$ passes are made. In each, the Hough transform for $\sqrt{N}$ angles is computed. The Hough transform for $\sqrt{N}$ angles can be computed in $O(logN)$ time by having PE (i,i) of each $\sqrt{N}\times\sqrt{N}$ sub RMESH initiate a token with value equal to that of the image value at the corresponding point. The r value for the token in PE (i,i) is obtained by using the i'th angle. Now we need to add up the token values of all tokens with the same r and θ values. This can be done by combining tokens in pairs of sub RMESHs at a time. The strategy employed is similar to that used in [JENQ90]. Two kinds of combinations are needed: horizontal and vertical. In horizontal combining two sub RMESHs each containing $k\times k$ $\sqrt{N}\times\sqrt{N}$ sub RMESHs are combined to get a single sub RMESH which is a $k\times(2k)$ configuration of $\sqrt{N}\times\sqrt{N}$ sub RMESHs. In a vertical combining, two $k\times(2k)$ arrays of $\sqrt{N}\times\sqrt{N}$ sub RMESHs are combined to get a single $2k\times2k$ sub RMESH. The initial configuration for a horizontal combine has the tokens in the PEs on the antidiagonals of the two $k\times k$ $\sqrt{N}\times\sqrt{N}$ sub RMESHs (Figure 14). As in the discussion of Section 2, assume that θ_j is in the range $\pi/2 < \theta_j \le 3\pi/4$. So, r is in the range $[-\sqrt{2}N/2, N]$. However for any fixed θ_j in the above range at most $\sqrt{2}N$ distinct integer r values are possible. The maximum number of tokens in each PE is therefore $\sqrt{2}k\sqrt{N}$ (at most $\sqrt{2}k$ per angle and $\sqrt{N}$ angles). The number of PEs on each antidiagonal is $k\sqrt{N}$. So, using k PEs per angle we need to store at most $\sqrt{2} < 2$ tokens in each PE. The final configuaration has at most $2\sqrt{2}k\sqrt{N}$ tokens stored at most 4 per PE in the antidiagonal PEs of the left sub RMESH A.

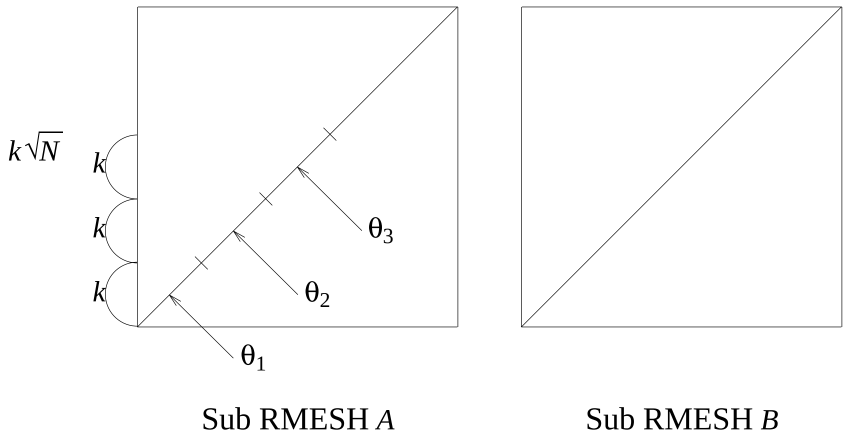


Figure 14 Initial configuration for horizontal combining

For a vertical combine, the initial configuration is the final configuation of a horizontal combine and the final configuration is the initial configuation of a horizontal combine. We shall describe how to go from the initial configuration of a horizontal combine to its final configuration in $O(1)$ time. The method for a vertical combine is similar. Since $O(logN)$ parallel combine steps suffice to combine all tokens, the complexity is $O(logN)$.

A horizontal combine for any angle θ_j, uses the PEs on the k rows dedicated to this angle. So, a $k \times 2k\sqrt{N}$ sub RMESH is available. Since $\pi/2 < \theta_j \le 3\pi/4$, the r values of the tokens in the k rows of the left sub RMESH A (call this sub RMESH KA) are $\ge$ the r values of the tokens in the same k rows of the right sub RMESH B (call this sub RMESH KB). Using column and row buses confined to the $k \times 2k\sqrt{N}$ sub RMESH formed by A and B we can obtain, in $O(1)$ time, a configuration in which PE (i,j) of a $k \times k$ sub RMESH of A contains the at most $2\sqrt{2}$ tokens in the i'th row of KA and j'th row of KB. In an additional $O(1)$ time, this PE can check if any of the tokens it has from KA have the same r value as any of the tokens it has from KB. If so, the KA token is deleted and the value of the KB token incremented by the value of the deleted KA token. The KA and KB tokens can be ranked in $O(1)$ time using the available processors and packed into the k antidiagonal processors of KA packing no more than four tokens per processor in an additional $O(1)$ time. Hence the horizontal combine can be completed in $O(1)$ time.

5 N^4 Processors

When N^4 processors are available, there is a rather straightforward $O((p/N)\log N)$ algorithm to compute the Hough transform. We assume that the RMESH is configured as an $N^2 \times N^2$ array with the N^2 pixel values initially in row 0. Since there are only $2\sqrt{2}N$ different r values possible, we compute the Hough transform for $N/(2\sqrt{2})$ angles simultaneously. Thus, $2\sqrt{2}p/N$ iterations are needed. Each row of the $N^2 \times N^2$ RMESH is assigned the task of obtaining the value of $H(r,j)$ for one pair (r,j). Since we are working with $N/(2\sqrt{2})$ angles simultaneously, the number of (r,j) pairs is N^2 which equals the numbers of rows. The image can be broadcast to each row using column buses. The processors in a row use the assigned angle to obtain the r for their pixel. If this equals the r value assigned to that row and if the pixel value is 1, the processor sets its *count* variable to 1; otherwise it sets it to 0. Adding the *count* variables in a row gives the Hough transform value for the (r,j) pair assigned to the row.

6 References

[CYPH87] R. E. Cypher, J. L. C. Sanz, and I. Snyder, "The Hough transform has $O(N)$ complexity on SIMD $N \times N$ mesh array architectures", Proceedings of

IEEE 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, pp 115-121, 1987.

[FISH87] A. Fisher and P. Highnam, "Computing the Hough transform on a scan line array processor", Proceedings of IEEE 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, pp 83-87, 1987.

[GUER87] C. Guerra and S. Hambrush, "Parallel algorithms for line detection on a mesh", Proceedings of IEEE 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, pp 99-106, 1987.

[IBRA86] H. A. Ibrahim, J. B. Kender, and D. E. Shaw, "On the application of massively parallel SIMD tree machine to certain intermediate-level vision tasks", Computer Vision, Graphics, and Image Processing, 36, 1986, pp 53-75.

[JENQ90] J. Jenq and S. Sahni, "Reconfigurable mesh algorithms for the area and perimeter of image components and histogramming", submitted.

[LI86] Li, Lavin, and LeMaster, "Fast Hough transform: A hierarchical approach", Computer Vision, Graphics, and Image Processing, 36, 3, Dec. 1986.

[LI89] Li, and Maresca, "Polymorphic-torus architecture for computer vision", IEEE Trans. on PAMI, 11, 3, March 1989.

[MARE88] Maresca, Lavin, and Li, "Parallel Hough transform algorithms on polymorphic torus", in High Level Vision in Multicomputers, Levialdi, (ed), Academic Press, 1988.

[MARE89] Maresca, Li, and Sheng, "Parallel computer vision on polymorphic torus architecture", Intl. Jr. on Computer Vision and Applications, 2, 4, Fall 1989.

[MILL88a] R. Miller, V. K. Prasanna Kumar, D. Resis and Q. Stout, "Data movement operations and applications on reconfigurable VLSI arrays", Proceedings of the 1988 International Conference on Parallel Processing, The Pennsylvania State University Press, 1988, pp 205-208.

[MILL88b] R. Miller, V. K. Prasanna Kumar, D. Resis and Q. Stout, "Meshes with reconfigurable buses", Proceedings 5th MIT Conference On Advanced Research IN VLSI, 1988, pp 163-178.

[MILL88c] R. Miller, V. K. Prasanna Kumar, D. Resis and Q. Stout, "Image computations on reconfigurable VLSI arrays", Proceedings IEEE Conference On Computer Vision And Pattern Recognition, 1988, pp 925-930.

[RANK90] S. Ranka and S. Sahni, *Hypercube algorithms with applications to image processing and pattern recognition*, Springer-Verlag, New York, 1990, pp 145-166.

[ROSE88] A. Rosenfeld, J. Ornelas, and Y. Hung, "Hough transform algorithms for mesh-connected SIMD parallel processors", Computer Vision, Graphics, and Image Processing, 41, 1988, pp 293-305.

[SILB85] T. M. Silberberg, "The Hough transform in the geometric arithmetic parallel processor", Proceedings IEEE Workshop on Computer Architecture and Image Database Management, 1985, pp 387-391.

Appendix A: O(1) Ranking

Consider an $N \times N$ RMESH in which each PE has a Boolean variable *selected*. If *selected* (i,j) is true then *rank* (i,j) is the number of PEs with *selected* (i,j) true that precede it in the defined linear ordering. If *selected* (i,j) is false, then *rank* (i,j) is undefined. Suppose that all the PEs with *selected* (i,j) = true are on row 0 (i.e. *selected* (i,j) = false, $i > 0$). Hence, at most, N elements are to be ranked. *rank* $(0,j)$, $0 \le j < N$, can be computed in $O(1)$ time using the steps of Figure A1.

Step 1 [rank even columns]
Compute $r(0,j)$ for j even where r is defined as
$r(0,j) = |\{q|\ q \text{ is even and } selected(0,q) \text{ and } q \le j\}|$

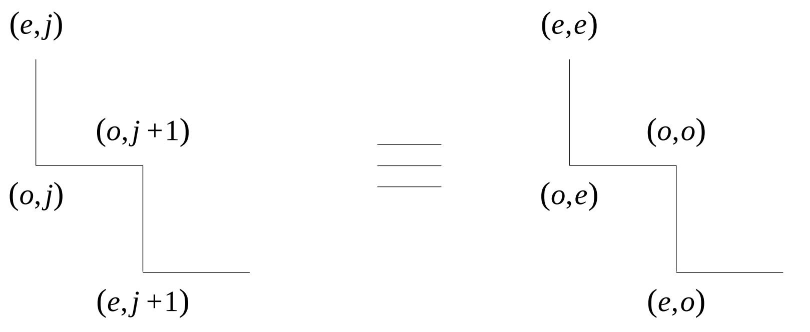
Step 2 [rank odd columns]
Compute $r(0,j)$ for j odd where r is defined as
$r(0,j) = |\{q|\ q \text{ is odd and } selected(0,q) \text{ and } q \le j\}|$

Step 3 [combine]

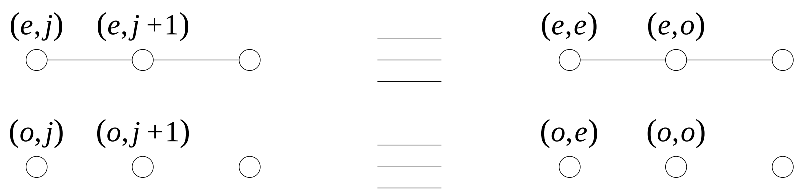
$$rank(0,j) = \begin{cases} r(0,0) - 1 & j = 0 \\ r(0,j) + r(0,j-1) - 1 & j > 0 \end{cases}$$

Figure A1 Steps in O(1) ranking

The algorithms for steps 1 and 2 are similar. So we describe the algorithm only for step 1. To compute $r(0,j)$, for even j, we set the bus switches as in Figure A2 (a) in case $selected(0,j)$ is true and as in Figure A2 (b) in case it is not. The switch settings are similar to those used to compute the exclusive or of 1's in [MILL88]. In this figure e denotes an even index and o an odd index. So, (e,j) denotes all PEs (i,j) with even i. Note that since j is even (e,j) is equivalent to (e,e) and $(e,j+1)$ to (e,o). Solid lines indicate connected (closed) switches and blanks indicate disconnected (open) switches.



(a) Settings for $selected(0,j)$ = true



(b) Setting for $selected(0,j)$ = false

Figure A2 Switch settings to compute $r(0,j)$ for j even

The algorithm to implement this strategy is given in Figure A3. Its complexity is readily seen to be $O(1)$. As mentioned earlier, the algorithm for step 2 is similar. Step 3 simply requires a rightward shift of 1 which can be easily done in $O(1)$ time. Hence the entire ranking can be done in $O(1)$ time.

```
   { Compute $r(0,j)$ for j even }
Step 1   $t(0,j) := selected(0,j),\ 0 \le j < N$;
Step 2   Set up column buses;
Step 3   Broadcast $t(0,j)$ on column bus $j$, $0 \le j < N$;
Step 4   $t(i,j)$ := content(bus); $0 \le i,j < N$;
Step 5   {send $t(i,j)$ for $j$ even to $t(i,j)$ for $j$ odd }
             All PEs $(i,j)$ with $j$ even disconnect their N, S,
             W switches and connect their E switch;
             All PEs $(i,j)$ with $j$ even broadcast $t(i,j)$;
             All PEs $(i,j)$ with $j$ odd set $t(i,j)$ to their bus
             content;
   Step 6   { set switches as in Figure A2 }
             if $t(i,j)$ then case $(i,j)$ of
                 (odd,odd),(even,even): PE $(i,j)$ disconnects
                 its E switch and connects its S switch;
                 else PE $(i,j)$ connects its E switch and
                 disconnects its S switch;
                 endcase
             else case $i$ of
                 odd : PE $(i,j)$ disconnects its E and S
                 switches;
                 else : PE $(i,j)$ connects its E switch and
                 disconnects its S switch;
             endcase;
   Step 7   PE $(0,0)$ broadcasts a special value on its bus;
   Step 8   All PEs $(i,j)$ with $i$ and $j$ even read their bus;
             If the special value is read, then they set their $S$
             value to true and $r$ value to $i/2 + 1$;
   Step 9   Set up column buses;
   Step 10  PE $(i,j)$ puts its $r$ value on its bus if $S(i,j)$ is true;
   Step 11  $r(0,j)$ = content(bus), $j$ even;
```

Figure A3 RMESH algorithm to compute $r(0,j)$ for j even