JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 24, 107–114 (1995)

The Hough Transform on a Reconfigurable Multi-Ring Network

SUCHENDRA M. BHANDARKAR AND HAMID R. ARABNIA

Department of Computer Science, 415 Boyd Graduate Studies Research Center, University of Georgia, Athens, Georgia 30602-7404

A novel reconfigurable network referred to as the Reconfigurable Multi-Ring Network (RMRN) is described. The RMRN is shown to be a truly scalable network, in that each node in the network has a fixed degree of connectivity and the reconfiguration mechanism ensures a network diameter of $O(\log_2 N)$ for an N-processor network. Algorithms for the 2-D mesh and the SIMD n-cube are shown to map very elegantly onto the RMRN. Basic message passing and reconfiguration primitives for the SIMD RMRN are designed which could be used as building blocks for more complex parallel algorithms. The RMRN is shown to be a viable architecture for image processing and computer vision problems via the parallel computation of the Hough transform. The parallel implementation of the Y-angle Hough transform of an $N \times N$ image is shown to have a asymptotic complexity of $O(Y \log_2 Y + \log_2 N)$ on the SIMD RMRN with $O(N^2)$ processors. This compares favorably with the $O(Y + \log_2 N)$ optimal algorithm for the same Hough transform on the MIMD n-cube with $O(N^2)$ processors. © 1995 Academic Press, Inc.

1. INTRODUCTION

In distributed memory multiprocessor systems, the topology of the interconnection network is a crucial factor in determining overall system performance. It is desirable for the interconnection network to be symmetric and scalable and to have a low diameter and for each node in the network to have a low degree of connectivity. For most symmetric network topologies, the requirements of low degree of connectivity for each node and low network diameter are often conflicting. Low network diameter often entails that each node in the network has a high degree of connectivity, resulting in a drastic increase in the number of interprocessor connection links. Low degree of connectivity, on the other hand, results in a high network diameter, which in turn results in high interprocessor communication overhead and reduced efficiency of parallelism.

Reconfigurable networks attempt to address this tradeoff. In a reconfigurable network each node has a fixed degree of connectivity irrespective of the network size. The network diameter is restricted by allowing the network to reconfigure itself into different configurations. Broadly speaking, a reconfigurable system needs to satisfy the following criteria in order to be considered practically viable: (a) in each configuration the nodes in the network should have a fixed degree of connectivity irrespective of network size, (b) the network diameter should be kept low via the reconfiguration mechanism, and (c) the hardware for the reconfiguration mechanism (i.e. switch) should be of reasonable complexity.

The objective of this paper is two-fold:

(a) We describe a novel reconfigurable network which we refer to as the Reconfigurable Multi-Ring Network (RMRN) [1]. We show that the RMRN satisfies the aforementioned criteria for practical viability. We show that algorithms for the *2-D mesh* and the *SIMD n-cube* can be mapped very elegantly onto the RMRN. We design and analyze certain basic message passing and reconfiguration primitives on the SIMD RMRN which can be used as building blocks for more complex parallel algorithms.

(b) We show the RMRN to be a viable architecture for image processing and computer vision problems by demonstrating the parallel computation of the Hough transform on the SIMD RMRN. The Hough transform is known to be a very important though computationally intensive operation in computer vision and image processing applications and its parallelization has been attempted by several researchers on a variety of architectures such as the mesh, hypercube, pyramid, and systolic array. The parallel implementation of the Hough transform on the SIMD RMRN is shown to have an order of complexity comparable to that on the MIMD n-cube.

2. THE RMRN TOPOLOGY—BASIC DEFINITIONS

Let RMRN_n denote an RMRN with $N = 2^n$ processors. The processors are numbered 0, 1, 2, ..., $N - 1$. Each processor p in the RMRN is uniquely specified using an n-bit address $(p_0, p_1, \ldots, p_{n-1})$. The system RMRN_n has $n + 1$ different configurations, where each configuration is denoted by $\text{config}(\text{RMRN}_n, i)$, where $0 \le i \le n$. The RMRN_n in $\text{config}(\text{RMRN}_n, i)$ consists of $r = 2^i$ rings, R_0, R_1, ..., R_{r-1}, stacked in a pipelined manner so that each ring has $k = 2^{n-i}$ processors. Each processor in R_0 is connected to an input channel and each processor in R_{r-1} is connected to the output channel. For $0 < j < r - 1$ every processor in R_j is connected to its corresponding processor in R_{j+1} and R_{j-1}. For example, Figs. 1a and 1b show the RMRN_4 in configurations $\text{config}(\text{RMRN}_4, 1)$ (i.e., 2 rings of 8 processors each) and $\text{config}(\text{RMRN}_4, 2)$ (i.e., 4 rings of 4 processors each), respectively.

Given that the RMRN_n is in configuration $\text{config}(\text{RMRN}_n, i)$, a processor p is in ring R_j iff $p \bmod r = j$. Also, we say that p is in position q in the ring

0743-7315/95 $6.00

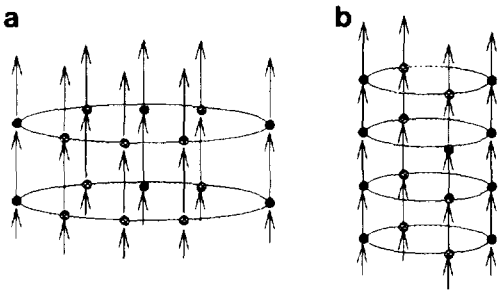

FIG. 1. $RMRN_4$ in (a) configuration 1 and (b) configuration 2.

R_j iff p div $r = q$. Furthermore, p is connected to $(p + r)$ mod N and $(p - r)$ mod N in ring R_j via bidirectional channels. If $j \neq 0$ and $j \neq r - 1$ then there are bidirectional channels between p and $p + 1$ and between $p - 1$ and p. If $j = r - 1$ then there is a channel between p and the output. Conversely, p is connected to the input if $j = 0$. Thus, in config($RMRN_n$, i), there is a link between the input and p iff p mod $r = p$ mod $2^i = 0$ (i.e., p is in R_0), else there is a link between $p - 1$ and p. Conversely, p is connected to the output iff p mod $r = p$ mod $2^i = r - 1$, else p is connected to the input of $p + 1$. The input and output control switches are thus designed to check whether the last i bits of p in config($RMRN_n$, i) are all 0 or all 1, respectively.

We now consider the switch that enables the system to reconfigure itself from config($RMRN_n$, i) to config($RMRN_n$, j), where $0 \leq i, j \leq n$ and $i \neq j$. In config($RMRN_n$, i), processor p must be connected to a switch which can connect it to $(p + 2^i)$ mod N and $(p - 2^i)$ mod N. Consider the circuit SW_1 shown in Fig. 2a with the truth table

S	Z_0Z_1
0	I_0I_1
1	I_1I_0

SW_1 is a reconfiguration switch for $RMRN_1$ if $Z_0 = I_0$ and $Z_1 = I_1$. Note that the switch has no effect when $S = 0$. Given a switch SW_{n-1} which works correctly for $N/2 = 2^{n-1}$ processors, Fig. 2b shows a switch SW_n which will work for N processors (for simplicity of the figure only half of the labels and connections are shown). This switch uses two copies of SW_{n-1} and N SW_1 switches. Note further that the switch has no effect when all the controls are 0. Thus, in general, we need $N \log_2 N$ switches to construct a switch for $RMRN_n$, where $N = 2^n$. Although the complexity of $O(N \log_2 N)$ is not very attractive for very large values of N, it is manageable for moderate values of N. Furthermore, the switch is composed of fairly simple components, is easily upgradable, and is amenable to VLSI implementation [2]. Most importantly, each processor in $RMRN_n$ needs four bidirectional channels irrespective of the network size $N = 2^n$. The fixed connectivity of each node in $RMRN_n$ ensures that the number of interprocessor communication links scales *linearly* with network size.

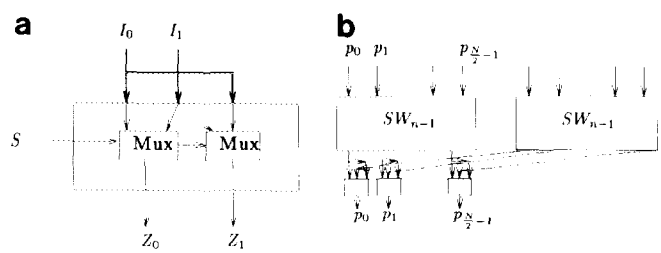

FIG. 2. The reconfiguration switches (a) SW_1 and (b) SW_n.

3. THE RMRN TOPOLOGY—SOME BASIC PROPERTIES

We state some important properties of the RMRN topology. For the sake of brevity, the proofs of these properties have been omitted here but can be found in [3].

Property 1. All 2-D mesh topologies of size $2^i \times 2^j$ are subsets of the $RMRN_n$, where $i + j = n$.

Property 2. The n-cube (i.e. a hypercube with $N = 2^n$ processors) can embed all possible configurations of the $RMRN_n$ (though not simultaneously).

Property 3. The configuration config($RMRN_m$, j) is a subconfiguration of config($RMRN_n$, i) whenever $m < n$, $j < i$ and $n - m = i - j$. In other words, config($RMRN_m$, j) is a subset of config($RMRN_n$, i) with the connectivity pattern preserved.

Property 4. Any algorithm which uses the communication links along a single dimension of the n-cube at any given point in time can be mapped to the $RMRN_n$ in $O(1)$ time.

Property 1 states that the $RMRN_n$ can be configured into a variety of mesh topologies. In fact, it can be proved that the $RMRN_n$ in config($RMRN_n$, i) contains as its subset a $(2^i \times 2^{n-i})$-processor 2-D wrap-around (i.e., toroidal) mesh. Property 2 shows how the n-cube can be used to simulate the behavior and function of the $RMRN_n$.

Property 3 shows that the $RMRN_n$ possesses an elegant recursive property with regard to its structure in a manner similar to the n-cube. In general, config($RMRN_n$, i) would contain 2^{n-m} subconfigurations of the type config($RMRN_m$, j) such that $m < n$, $j < i$, and $n - m = i - j$. The condition $n - m = i - j$ ensures that the number of processors in each ring of config($RMRN_n$, i) and config($RMRN_m$, j) is the same. The processor address p in config($RMRN_n$, i) can thus be decomposed as $p = p_m p_{n-m}$, where the lower-order $n - m$ bits specify a particular subconfiguration of the type config($RMRN_m$, j) and the higher-order m bits specify the address p' within the subconfiguration config($RMRN_m$, j).

Property 4 shows how an important and fairly wide class of algorithms for the n-cube can be mapped onto the

$RMRN_n$. Let $p[i]$ denote the ith bit in the n bit address of processor p. Let $p(i)$ represent the processor whose n-bit address differs from that of processor p only in the ith bit. Let F_i^n be the collection of edges in the n-cube in the ith dimension; i.e., $F_i^n = \{(p, p(i))\}$. $F_0^n, \ldots, F_{n-1}^n$ is a 1-*factor* of the n-cube. Without loss of generality, we may assume that $p[i] = 0$ and $p(i)[i] = 1$. Therefore $p(i) = p + 2^i$, which means that p and $p(i)$ are connected in config($RMRN_n$, i). Thus, $F_i^n \subset$ config($RMRN_n$, i). Let $A(p)$ denote the A register in processor p. Let the statement reconfig(n, i) reconfigure $RMRN_n$ in config($RMRN_n$, i). Also, let left(p), right(p), next(p), and previous(p) denote the processors $(p - 2^i) \bmod N$, $(p + 2^i) \bmod N$, $(p + 1) \bmod N$, and $(p - 1) \bmod N$, respectively, in config($RMRN_n$, i). The statements $A(p) \rightarrow A(p(i))$ and $A(p) \leftarrow A(p(i))$ in an algorithm for the n-cube which move the contents of the A register of p to the A register of $p(i)$ and vice versa along the ith dimension edge $(p, p(i))$ are respectively equivalent to the following statements on the $RMRN_n$:

```
reconfig(n, i);
IF (p[i] = 0) THEN A(p) --> A(right(p))
ELSE A(p) --> A(left(p));
```

and

```
reconfig(n, i);
IF (p[i] = 0) THEN A(p) <-- A(right(p))
ELSE A(p) <-- A(left(p));
```

This result is of special significance since it allows a wide class of algorithms designed for the n-cube to be mapped onto the $RMRN_n$ without loss of performance, assuming that the overhead in reconfiguration is not excessive. Any algorithm which uses the communication links along a single dimension of the n-cube at any given point in time can be mapped to the RMRN multiprocessor $RMRN_n$ in $O(1)$ time. Since a vast majority of algorithms that use the n-cube in the SIMD or SPMD (and sometimes even in the MIMD) modes of parallelism use the edges of the n-cube along one specific dimension at any given time, we can conclude that the RMRN topology provides the same generality *in practice* as the n-cube for a large class of problems with greater cost-effectiveness and without loss in performance.

4. SOME BASIC OPERATIONS ON THE RMRN

In this section we shall prove that a wide class of basic operations and algorithms can easily be implemented on the SIMD $RMRN_n$. Each processor has its own local memory for data storage. A single control unit broadcasts a common instruction stream to all the processors in the RMRN. Each processor can either execute the current instruction or ignore it altogether depending on the processor address and/or the state of the variables in its local memory. The control unit also issues the command(s) to the reconfiguration switch to reconfigure the RMRN in a given configuration thus ensuring that all processors operate in a single configuration at any given time. Intraprocessor assignments are denoted by :=, whereas $\rightarrow$ denotes interprocessor assignments. Interprocessor assignments utilize the interprocessor links in the RMRN. The term *unit hop* is used to denote communication between processors in the RMRN that are directly connected. The asymptotic complexity of any algorithm for the RMRN is decided by the number of unit hops in the algorithm.

4.1. General Message Passing Operations

In this section we show how general message passing strategies can be implemented on the $RMRN_n$.

4.1.1. Broadcast Operation

Let us assume that the data in register X of a processor p need to be broadcast to all the other processors. Reconfigurability permits the broadcast operation to be performed in $O(n)$, that is, $O(\log_2 N)$ unit hops. In fact, the diameter of the $RMRN_n$ network (allowing for reconfigurability) can be shown to be $\lceil 0.5 \times \log_2 N \rceil = \lceil n/2 \rceil$. The diameter of the RMRN, thus, scales logarithmically with network size. The broadcast algorithm is shown in Fig. 3. In this algorithm, the broadcast message is assumed to originate at processor 0. The function MSONE(p) denotes the position of the most significant 1 in the binary processor address p. For example, MSONE(5) = MSONE(0101) = 2 and MSONE(9) = MSONE(1001) = 3.

4.1.2. Combine Operation

Let $\oplus$ denote any associative binary operation, such as MAX, MIN, logical AND, logical OR, sum, or product.

```
Broadcast(X, n);
  BEGIN
    FOR i = 0 to n-1 DO
     BEGINFOR
       reconfig(n,i);
       IF (MSONE(p) < i) THEN
         IF (p[i] = 1) THEN
           X(p) --> X(left(p));
         ELSE
           X(p) --> X(right(p));
         ENDIF;
       ENDIF;
       IF (MSONE(p) = i) THEN
         IF (p[i] = 1) THEN
           X(p) <-- X(left(p));
         ELSE
           X(p) <-- X(right(p));
         ENDIF;
       ENDIF;
     ENDFOR;
  END;
```

FIG. 3. Broadcast operation.

```
Combine(X, n, F);
   BEGIN
      FOR i = 0 to n-1 DO
        BEGINFOR
          reconfig(n,i);
              IF (p[i] = 1) THEN
                  X(p) --> Y(left(p));
                  Y(p) <-- X(left(p));
              ELSE
                  X(p) --> Y(right(p));
                  Y(p) <-- X(right(p));
              ENDIF;
          X(p) := F(X(p),Y(p)); {F is an associative binary operation}
        ENDFOR;
   END;
```

FIG. 4. Combine operation.

Let each processor contain a data item in its X register. One is interested in the $\oplus$ (i.e., combine) of all these data items, that is, $\bigoplus_{p=0}^{N-1} X(p)$, where $N = 2^n$. In this case reconfigurability allows us to achieve the combine operation in $O(\log_2 N)$ unit hops. The algorithm for the combine operation is given in Fig. 4, in which $F(x, y)$ is assumed to be an associative operation. At the end of the combine operation each processor contains the final result of the combine operation.

The broadcast and combine operations can also be performed in a window (i.e., subconfiguration) within the RMRN. The broadcast and combine operations can then be carried out in parallel in each window (i.e., by the processors in each subconfiguration). If we are interested in config(w, k) in a window of size $W = 2^w$, then RMRN_n has to be placed in config(n, $n - w + k$). This ensures that config(w, k) is a proper subconfiguration of config(n, $n - w + k$) (Property 3). In RMRN_n, with $N = 2^n$ processors, a window of size $W = 2^w$ is defined by appropriately assigning values to the lower order $(n - w)$ bits in the address of the processors. The address of the processor within the window is given by appropriately masking the lower order $(n - w)$ bits in the address of the processor (Property 3). Let $p\{w\}$ denote the address of the processor after the masking operation for a window of size $W = 2^w$ has been carried out for RMRN_n. We can see that $p\{w\}$, right($p\{w\}$) and left($p\{w\}$) in config(w, k) refer to processors p, right(p) and left(p) respectively in config(n, $n - w + k$) since the connectivity pattern is preserved in the subconfigurations of the RMRN_n (Property 3). The broadcast and combine operations have an asymptotic complexity of $O(\log_2 W)$ when performed within a window of size $W = 2^w$.

4.2. The Rotate Operation on the RMRN

Assume that one is given a 2-D image $G = \{G(i, j); i, j \in [0, N - 1]\}$ and a SIMD RMRN with $N^2 = 2^{2n}$ processors where the value of the pixel (i, j) is stored in register $R(p)$ of processor $p = iN + j$ (i.e., row-major mapping). We will occasionally refer to the processor $p = iN + j$ by the ordered pair (i, j) keeping in mind that the row-major mapping is a bijection, since $i = p$ div N and $j = p$ mod N. Similarly, we will also occasionally refer to the register $R(p)$ as $R(i, j)$. One of the basic imaging operations that could be performed on the RMRN_{2n} with $N^2 = 2^{2n}$ processors is the *rotate* or the *cyclic shift* operation performed within a window (i.e., subimage) of size $W \times W = 2^{2w}$.

In RMRN_{2n} with $N^2 = 2^{2n}$ processors, a window of size $W \times W = 2^{2w}$ is defined by appropriately assigning values to the lower order $2(n - w)$ bits in the address of the processors. The address of the processor within the window is given by appropriately masking the lower order $2(n - w)$ bits in the address of the processor. The address of the processor after the masking operation for a window of size $W \times W = 2^{2w}$ has been carried out is denoted by $p\{2w\}$. The kth bit in the masked address is then denoted as $p\{2w\}[k]$. The rotate operation for the RMRN is described thus:

A 1-D rotate operation by 2^k-pixels in a window of size $W^2 = 2^{2w}$ can be described as (i) Rotate *left*: $R(p) \leftarrow R((p + 2^k) \bmod W^2)$ and (ii) Rotate *right*: $R(p) \leftarrow R((p - 2^k) \bmod W^2)$.

A 2-D rotate operation by 2^k-pixels can be described as (i) Rotate *right*: $R(i, j) \leftarrow R(i, (j - 2^k) \bmod W)$, (ii) Rotate *left*: $R(i, j) \leftarrow R(i, (j + 2^k) \bmod W)$, (iii) Rotate *up*: $R(i, j) \leftarrow R((i + 2^k) \bmod W, j)$, and (iv) Rotate *down*: $R(i, j) \leftarrow R((i - 2^k) \bmod W, j)$

A generic rotate operation within a window is described in Fig. 5. For a 1-D rotation in a window of size $W^2 = 2^{2w}$ the Rotate procedure would have to be invoked thus: (i) Rotate left by 2^k-pixels: `RotateWin(R, 2n, k, 2w-1, 2w, 0)` and (ii) Rotate right by 2^k pixels: `RotateWin(R, 2n, k, 2w-1, 2w, 1)`. For a 2-D rotation in a window of size $W^2 = 2^{2w}$ the Rotate procedure would have to be invoked thus: (i) Rotate down by 2^k pixels: `RotateWin(R, 2n, w+k, 2w-1, 2w, 1)`, (ii)

```
RotateWin(R, n, s, r, w, flag)
 BEGIN
  reconfig(n, n-w+s);
  IF (p{w}[s] = 0) THEN R(p) --> R(right(p))
  ELSE R(p) --> R(left(p));
  FOR b:= s+1 to r DO
   BEGINFOR
     reconfig(n, n-w+b);
     IF (flag = 0) THEN
       IF (p{w}[b-1] = 1) AND (p{w}[b-2] = 1) AND ... (p{w}[s] = 1) THEN
         IF (p{w}[b] = 0) THEN R(p) --> right(R(p))
         ELSE  R(p) --> left(R(p));
       ENDIF;
     ELSE
       IF (p{w}[b-1] = 0) AND (p{w][b-2] = 0) AND ... (p{w}[s] = 0) THEN
         IF (p{w}[b] = 0) THEN R(p) --> right(R(p))
         ELSE  R(p) --> left(R(p));
       ENDIF;
     ENDIF;
   ENDFOR;
 END;
```

FIG. 5. Rotate operation within a window.

Rotate up by 2^k pixels: `RotateWin(R, 2n, w+k, 2w-1, 2w, 0)`, (iii) Rotate left by 2^k pixels: `RotateWin(R, 2n, k, w-1, 2w, 0)`, and (iv) Rotate right by 2^k pixels: `RotateWin(R, 2n, k, w-1, 2w, 1)`.

The rotate operation within a $W \times W = 2^{2w}$ window contains $O(w) = O(\log_2 W)$ unit hops.

5. THE HOUGH TRANSFORM FOR LINE DETECTION

The Hough transform is known to be a very important though computationally intensive operation in computer vision and image processing. The conventional Hough transform is used to detect and extract features with well defined parametric descriptions in an image. Serial implementation of the Hough transform is computationally intensive and is not feasible for real-time applications. Parallelization of the Hough transform is imperative and has been attempted by several researchers on a variety of architectures such as the mesh [6], hypercube [7], pyramid [8], systolic array [9], and shared memory architecture [10]. In this paper, we consider the parallelization of the Hough Transform for line detection on the SIMD RMRN. By parallelizing the Hough transform on the RMRN, we wish to demonstrate its capability as a viable architecture for image processing and computer vision problems.

Let $E = E(i, j)\ 0 \le i, j < N$ be an edge image such that $E(i, j) = 1$ if the point (i, j) is an edge point and $E(i, j) = 0$ otherwise. The general equation of a straight line in E is given by the parametric equation,

$$r = i \cos \theta + j \sin \theta, \tag{1}$$

where r is the perpendicular distance of the line from the origin and θ is the angle made by the normal to the line with the i axis. The (r, θ) space is referred to as parameter space or *Hough space*. If the Hough space (r, θ) is discretized in the form of an array or accumulator $H(x, y)$ composed of a finite number of *cells* or *bins* then each point (i, j) in E can be considered to have cast a vote in each bin (x, y) that represents the (r, θ) parameters of the lines passing through (i, j). The maxima in the (r, θ) bins can be thus considered to correspond to the parameters of the desired lines.

Formally, the Hough transform $H(x, y)$ is given by

$$H(x, y) = |\{(i, j) \text{ such that } x = \lfloor i \cos \theta_y + j \sin \theta_y \rfloor\}|, \tag{2}$$

where $\theta_y = \pi(y + 1)/Y$, $0 \le y < Y$, $E(i, j) = 1$, and $|P|$ denotes the cardinality of set P.

The resulting transform is said to be the *Y-angle* Hough transform of the edge image $E(i, j)$. For an $N \times N$ image x lies in the range $[0, \sqrt{2}N)$ whenever y lies in the range $[0, Y)$. Let $N = 2^n$ and let $Y = 2^m$ such that $m < n$. Since x lies in the range $[0, \sqrt{2}N)$, we assume that the RMRN has $2N^2 = 2^{2n+1}$ processors, i.e., the smallest number of processors that is a power of 2 and greater than $N \times \sqrt{2}N = \sqrt{2}N^2$. We will view the RMRN as an $N \times 2N$ array. There are three phases in the computation of the Hough transform $H(x, y)$:

(I) Consider the subconfiguration RMRN_{n+m+1} consisting of $Y \times 2N = 2^{n+m+1}$ processors. Each such subconfiguration RMRN_{n+m+1} will be used to compute the Hough transform $h(x, y)$ in a window of size $Y \times 2N = 2^{n+m+1}$. That is,

$$h(x, y) = |\{(i, j) \text{ such that } x = \lfloor i \cos \theta_y + j \sin \theta_y \rfloor\}|, \tag{3}$$

where $\theta_y = \pi(y + 1)/Y$, $0 \le y < Y$, $E(i, j) = 1$ and (i, j) lies within the subconfiguration.

(II) Compute the window sum of each of the $h(x, y)$'s,

$$H(x, y) = \sum_{W} h(x, y), \tag{4}$$

where W is a window of size $N/Y = 2^{n-m}$. Since there are $N/Y = 2^{n-m}$ such windows, the accumulation can be implemented as a *combine* operation in a window of size $N/Y = 2^{n-m}$ using the *sum* function.

(III) Compute the maximum in $H(x, y)$. This too can be done using a combine operation in a window of size $Y \times 2N = 2^{n+m+1}$ using an associative function MAX. The function MAX would need to return not only the maximum value but also the address of the processor containing the maximum value. That is,

$$\text{MAX}((x, p_1), (y, p_2)) \begin{cases} (x, p_1) & \text{if } x > y \\ (y, p_2) & \text{if } y > x. \end{cases}$$

Phase I of the algorithm is elaborated upon in the following subsection. Phases II and III are straightforward and need no further elaboration.

5.1. Phase I: Computing the Hough Transform in Each Subconfiguration

Consider the subconfiguration RMRN_{n+m+1} consisting of $Y \times 2N = 2^{n+m+1}$ processors. There are $N/Y = 2^{n-m}$ such subconfigurations in RMRN_{n+m+1}. Each processor in the subconfiguration RMRN_{n+m+1} can be considered to be an element of an array with $Y = 2^m$ rows and $2N = 2^{m+1}$ columns. This array is mapped onto RMRN_{n+m+1} in a row-major fashion. That is, $p\{n + m + 1\} = y(2N) + x$, where $0 \le y \le Y - 1$, $0 \le x \le 2N - 1$, and $p\{n + m + 1\}$ denotes the address of the processor p within RMRN_{n+m+1} obtained by masking the lower order $n - m$ bits in the address of p. Alternatively, $y = p\{n + m + 1\}$ div $(2N)$ and $x = p\{N + m + 1\}$ mod $(2N)$. Let each processor $p(i, j)$ in RMRN_{2n+m+1} generate a triple (x, y, q), where x is the column number of $p(i, j)$ in RMRN_{n+m+1}, y is the row number of $p(i, j)$ in RMRN_{n+m+1}, and q is defined as

$$q = |\{(i, j) \text{ such that } x = \lfloor i \cos \theta_y + j \sin \theta_y \rfloor\}|, \tag{5}$$

where $\theta_y = \pi(y + 1)/Y$, $0 \le y < Y$, and $E(i, j) = 1$.

We assume that q has been initialized to zero. The triple (x, y, q) is stored in the VOTES register of each processor. Elements within the triple are referred to as VOTES.x, VOTES.y, and VOTES.q, respectively. Let W denote the window corresponding to the subconfiguration RMRN_{n+m+1}. The key factor in designing an efficient algorithm for Phase I is to realize that not all pixels $E(i, j)$ in a given row of the $Y \times 2N$ Hough accumulator array $h(x, y)$ contribute toward the vote count of a given value of (x, y) (i.e., a given bin in the Hough accumulator array). This fact has been exploited by Cypher *et al.* [6] and Ranka and Sahni [7] in their Hough Transform algorithms on the SIMD mesh and the SIMD/MIMD hypercube architectures, respectively. We subdivide Phase I in the Hough Transform algorithm in the following subphases:

Phase I(a). $\pi/4 \le \theta_y < \pi/2$.

```
procedure votecount(W)
 BEGIN
   VOTES.x := x;  {column address of p{n+m+1} in the window W}
   VOTES.y := y;  {row address of p{n+m+1} in the window W}
   VOTES.q := 0;  {initialize the number of accumulated votes to 0}
   FOR count1 := 1 to Y/4 DO
    BEGINFOR
      reconfig(2n+1, n);
      theta := ((VOTES.y + 1) * PI)/Y + (PI/4);
      x1 = trunc(i * COS(theta) + j * SIN(theta));
      IF (x1 = VOTES.x) THEN VOTES.q := VOTES.q + 1;
      VOTES(p) --> VOTES(right(p)); {Rotate one position to the
                                     right along the row}
      theta := ((VOTES.y + 1) * PI)/Y + (PI/4);
      x1 = trunc(i * COS(theta) + j * SIN(theta));
      IF (x1 = VOTES.x) THEN VOTES.q := VOTES.q + 1;
      VOTES(p) --> VOTES(left(p));
      VOTES(p) --> VOTES(left(p)); {Rotate two positions to
                                    the left along the row}
      RotateWin(VOTES, 2n+1, n+1, n+m,  n+m+1, 1); {Rotate one position down
                                                    along the column}
    ENDFOR
 END
```

FIG. 6. Algorithm for the Hough transform within a window using left and right operations.

Phase I(b). $\pi/2 \leq \theta_y < 3\pi/4$.

Phase I(c). $0 \leq \theta_y < \pi/4$.

Phase I(d). $3\pi/4 \leq \theta_y < \pi$.

For each of subphases I(a)–I(d), the RMRN_{n+m+1} is configured as composed of $4N/Y$ windows (i.e., subconfigurations) of size $Y/4 \times 2N$ each. In particular, we exploit the following two lemmas from Ranka and Sahni [7]:

LEMMA 5.1. *When $\pi/4 \leq \theta_y < \pi/2$, two pixels (i, j) and $(i, j + k)$, $k > 0$, can contribute to the count of the same $H(x, y)$ only if $k = 1$.*

LEMMA 5.2. *When $\pi/4 \leq \theta_y < \pi/2$, two pixels (i, j) and $(i + 1, k)$ can contribute to the count of the same $H(x, y)$ only if $k \in \{j, j - 1\}$.*

Using Lemmas 5.1 and 5.2 one can implement the vote accumulation in a single row by a sequence of two *rotate* operations; a rotate operation to the right by one position followed by another rotate operation to the left by two positions. Further improvement can be realized by replacing the rotate operation along the rows by the *left* operation. This can be done by visualizing the RMRN_{2n+1} as composed of $N = 2^n$ rings, each containing $2N = 2^{n+1}$ processors (i.e., config$(2n + 1, n)$). Each $Y/4 \times 2N$ window can then be looked upon as subconfiguration RMRN_{n+m-1} composed of $Y/4 = 2^{m-2}$ rings of $2N = 2^{n+1}$ processors each (i.e., config$(n + m - 1, m - 2)$). In order that each of the $N = 2^n$ rings in RMRN_{2n+1} contains a single row of the image $E(i, j)$, the mapping of the image $E(i, j)$ has to be modified from the row-major mapping described in earlier sections to a column-major mapping, where the value of the pixel (i, j) is stored in register $R(p)$ of processor $p = jN + i$ in RMRN_{2n+1}. The rotate (or left) operations along the rows are followed by a rotate operation by a single position along the columns. The pseudocode for subphase I(a) is shown in Fig. 6. Subphases I(b)–I(d) are tackled in an identical manner.

The asymptotic complexity of subphases I(a)–(d) is $O(Y \log_2 Y)$, since each left and right operation is $O(1)$. This is an interesting result, since it makes the complexity of Phase I of the Hough Transform algorithm independent of N.

5.2. Analysis of Complexity

The asymptotic complexity of the subphases I(a)–(d) is $O(Y \log_2 Y)$. The ensuing *combine* operations for the window sum and maxima detection are $O(\log_2 N)$. The overall complexity of the Hough Transform algorithm is thus $O(Y \log_2 Y + \log_2 N)$. The asymptotic complexity for a given value of Y is still $O(\log_2 N)$. It is to be noted that the $O(Y \log_2 Y + \log_2 N)$ complexity of the Hough Transform algorithm on the SIMD RMRN compares favorably with the optimal $O(Y + \log_2 N)$ Hough Transform algorithm on the MIMD n-cube reported by Ranka and Sahni [7].

6. CONCLUSIONS AND FUTURE RESEARCH

In this paper we have described a novel reconfigurable multi-ring network (RMRN) with the following salient features: (a) the degree of connectivity of each node in the RMRN is fixed irrespective of network size, thereby ensuring that the number of interprocessor communication links scales linearly with network size, and (b) the network diameter scales logarithmically with network size. We have stated some important properties of the RMRN structure, most importantly that (a) the various configurations of the RMRN_n with 2^n processors can be embedded in an n-cube, (b) the RMRN_n contains as its subset various 2-D mesh topologies, and (c) the set of edges of the n-cube in any given dimension are a proper subset of the RMRN_n. As a result, we have shown that a broad class of algorithms for the n-cube can be mapped to the RMRN_n in a simple and elegant manner. We have designed and analyzed a general class of procedural primitives for the SIMD RMRN and shown how these primitives can be used as building blocks for more complex parallel operations. We have demonstrated the usefulness of the RMRN by considering an important operation in computer vision and image processing, that is, the Hough transform for detection of linear features in an image. We have designed an $O(Y \log_2 Y + \log_2 N)$ parallel algorithm for the Y-angle Hough transform on the SIMD RMRN with $O(N^2)$ processors which is comparable to the optimal $O(Y + \log_2 N)$ Hough transform algorithm on the MIMD *hypercube* with $O(N^2)$ processors. Thus, we have shown that the RMRN is a viable architecture for problems in computer vision and image processing. Efforts are currently underway toward building the hardware for the RMRN using INMOS T800 transputers and custom VLSI. Also being considered are more complex problems in intermediate- and high-level computer vision such as surface reconstruction, image segmentation, and object recognition.

ACKNOWLEDGMENTS

The assistance of the University of Georgia Research Foundation, Office of Instructional Development, at the University of Georgia and the National Science Foundation (CCR-8717033 and CDA-8820544) is acknowledged.

REFERENCES

1. Arabnia, H. R. A transputer-based reconfigurable parallel system. *Transputer Research and Applications (NATUG)*. IOS Press, Vancouver, Canada, 1993, pp. 153–169.
2. Arabnia, H. R., and Smith, J. W. A reconfigurable interconnection network for imaging operations and its implementation using a multi-stage switching box. *Proc. 1993 Intl. Conf. High Performance Computing: New Horizons*. Alberta, 1993, pp. 349–357.
3. Bhandarkar, S. M., and Arabnia, H. R. The Hough transform on a reconfigurable multi-ring network. Technical Report, Dept. of Computer Science, University of Georgia, Athens, GA, 30602-7404, March 1993.

4. Dekel, E., Nassimi, D., and Sahni, S. Parallel matrix and graph algorithms. *SIAM J. Comput.* **10,** 4 (Nov. 1981), 657–675.
5. Fang, Z., Li, X., and Ni, L. M. Parallel algorithms for image template matching on hypercube SIMD computers. *IEEE Workshop on Comp. Arch. for Pattern Analysis and Image Database Mgmt.* 1985, pp. 33–40.
6. Cypher, R. E., Sanz, J. L. C., and Snyder, L. The Hough transform has $O(N)$ complexity on SIMD $N \times N$ architectures, *Proc. IEEE Workshop on Computer Architectures for Computer Vision and Pattern Recognition.* 1987, 115–121.
7. Ranka, S., and Sahni, S. Computing Hough transforms on Hypercube multicomputers. *J. Supercomputing*, **4** (1990), 169–190.
8. Jolion, J. M., and Rosenfeld, A. A $O(\log N)$ pyramid Hough transform. *Pattern Recognition Lett.* **9** (1989), 343–349.
9. Chaung, H. Y. H., and Li, C. C. A systolic array processor for straight line detection by modified Hough transform. *Proc. IEEE Workshop on Comp. Arch. for Pattern Analysis and Image Database Mgmt.* 1985, pp. 300–304.
10. Choudhary, A. N., and Ponnusamy, R. Implementation and evaluation of Hough transform algorithms on a shared-memory multiprocessor. *J. Parallel Distrib. Comput.* **12** (1991), 178–188.

SUCHENDRA M. BHANDARKAR received his B.Tech. in electrical engineering from the Indian Institute of Technology, Bombay, in 1983, and his M.S. and Ph.D. in computer engineering from Syracuse University, Syracuse, New York, in 1985 and 1989, respectively. He is currently an assistant professor in the Department of Computer Science at the University of Georgia, Athens, Georgia. He was a Syracuse University Fellow for the academic years 1986–1987 and 1987–1988. He is a member of the professional societies IEEE, AAAI, ACM, and SPIE and the honor societies Phi Kappa Phi and Phi Beta Delta. He is a coauthor of the book *Object Recognition from Range Images* (Springer-Verlag, 1992). His research interests include computer vision, pattern recognition, image processing, artificial intelligence, and parallel algorithms and architectures for computer vision and pattern recognition. He has published over 40 research articles in these areas.

HAMID R. ARABNIA received a B.Sc. honors degree in mathematics and computing in 1983 from the Polytechnic of Wales (Pontypridd, United Kingdom) and a Ph.D. degree in computer science from the University of Kent (Canterbury, England) in 1987. In 1987, he worked for nine months as a computer science consultant for Caplin Cybernetics Corporation (London, England), where he helped in the design and implementation of a number of image processing algorithms. These algorithms were targeted at an MIMD machine architecture made up of transputers. Dr. Arabnia is currently an assistant professor of computer science at University of Georgia, where he has been since 1987. His research interests include parallel algorithms, reconfigurable machines, interconnection networks, and application of parallel processing to computer graphics and image processing.

Received March 8, 1993; revised September 17, 1993; accepted January 1, 1994