Executing Tree Routing Algorithms on a High-Performance Pattern Associative Router

Douglas H. Summerville[†], José G. Delgado-Frias[‡] and Stamatis Vassiliadis[*]

[†]Electrical Engineering Dept.
University of Hawaii at Manoa
Honolulu, HI 96822

[‡]Electrical Engineering Dept.
State University of New York
Binghamton, NY 13902-6000

[*]Electrical Engineering Dept.
Delft University of Technology
The Netherlands

Abstract

In this paper a novel programmable approach to execute implicit routing algorithms is presented. The proposed router is based on an associative scheme that uses the attributes of the routing algorithm and the interconnection network topology. In this approach routing algorithms are mapped (or programmed) onto a set of bit-patterns that are matched in parallel. To show the applicability of this router, we have selected oblivious and fault-tolerant routing algorithms for 10 different tree interconnection network topologies; however, the proposed scheme is flexible enough to accommodate other network topologies and routing algorithms. For the studied topologies, the number of required bit-patterns is of the same order as the topology degree. The proposed organization requires only one comparison and one read delays. This in turn yields a high-speed port assignment that is comparable to single topology routers (non-flexible routers). In the context of flexible router schemes, the proposed approach not only is one of the fastest but also requires a very small amount of hardware for its implementation.

Keywords

Routing Algorithm Execution, Interconnection Networks, Pattern Associative Memories, Oblivious Routing, Fault-Tolerant Routing, and Flexible Routers.

1 Introduction

The interconnection network is often considered to be the critical element in a multiprocessor computer due to the machine's performance sensitivity to network latency and throughput [2]. A large number of interconnection network topologies have been proposed and used

[5][16]; each network has features that make it suitable for a set of applications and algorithms. No single interconnection network, however, has been widely accepted as suitable for a large number of parallel computers and applications [16].

An interconnection network node has a router that provides a means of handling messages on the network. The router receives, forwards, and delivers messages as well as controls message flow through the network. Based on a routing algorithm, the router system transfers messages from its input ports to the proper output ports. Messages have two major components: the header (H), which contains the destination node address, and the body (B), which carries the message. Figure 1 shows a typical router system whose components include: an input port, a routing algorithm decoder (often called router), a flow controller, a switching network, and a set of output ports. The input port receives messages and handles the communication protocol with the sender. The message header is sent to the routing algorithm decoder which determines the output port the message should be sent to. The switching network, which is set by the routing algorithm decoder, provides a path to all the output ports. Messages are transferred in a manner dictated by the flow controller.

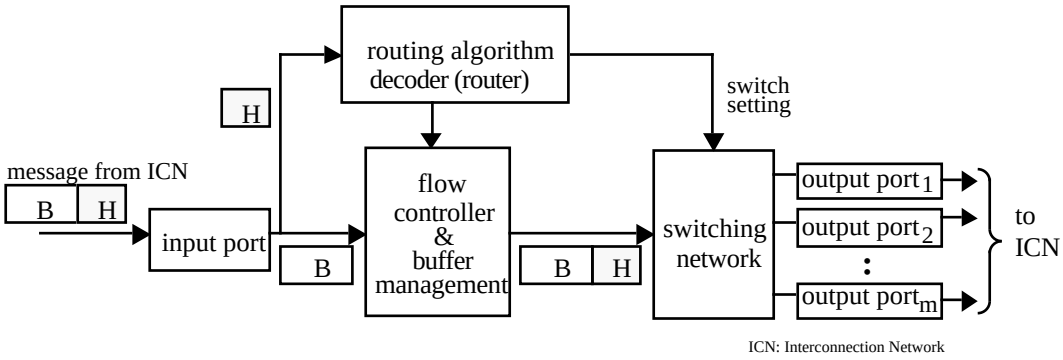


Figure 1: Router system.

A crucial component of any large scale parallel machine is the routing algorithm [12]. Given the large number of interconnection network topologies, it is highly desirable to have a reconfigurable router to provide flexibility to accommodate various network topologies and support to carry out a number of different routing algorithms. This would not only allow a single hardware design to be shared for a number of interconnection network topologies but also allow the routing algorithm to be customized based on application requirements. As routing delays tend to dominate transmission time [2], the performance of a reconfigurable router should be comparable to a topology-specific router; flexibility should not come at the expense of network performance. Thus, a reconfigurable router should provide expeditious determination of the destination port to be comparable to specialized routers while providing flexibility to accommodate modifications to a network and programmability to support a number of routing algorithms.

There have been two major approaches to the realization of flexible routers. These approaches are: dedicated processors [15][17] and look-up tables [18]. A dedicated processor allows the bit manipulation that most routing algorithms require to be easily executed. This type of router is able to accommodate a number of interconnection networks. Due to the sequential execution of the routing algorithms, however, this approach results in slow

routers. Furthermore, the resulting router can be expensive in terms of hardware. On the other hand, the lookup table approach requires the storage in memory of a predetermined routing path for all possible destination nodes. Using lookup tables reduces the time of determining the output port to a memory access delay. However, large tables are required to store the routing information. In order to decrease the number of entries in each table the interval routing approach has been proposed [21]. In this approach, each output port is assigned with intervals of destination node addresses; thus, the message is routed to the proper port according to the interval it belongs to. Drawbacks of this approach include that a short path is not always provided, cyclic interconnections are difficult to map, and adaptive routing may not be possible [4].

In this paper we present a novel router approach that provides flexibility to execute implicit routing algorithms as well as fast execution that is comparable to the look-up table systems. This is done by incorporating a pattern matching array that stores few bit-patterns. These bit-patterns are used to compare in parallel a number of potential routing alternatives; this implements the bit manipulation that is required to execute a routing algorithm. The number of entries in the pattern matching array is of the same order as the number of output ports. The proposed approach tends to map implicit routing algorithms onto an extremely small array. The chosen path is determined by a selection function.

A study of this approach as it applies to a large number of tree interconnection networks using oblivious and fault-tolerant routing algorithms is reported in this paper. Tree interconnection networks have been chosen to show the capabilities of the proposed router because they tend to have more lengthy and complex routing algorithms than other interconnection network topologies [17]. The approach, however, is not limited to tree interconnection networks. In [20] the mapping of routing algorithms for 37 interconnection network topologies is shown. In this paper as well as in [20] it is shown that the same design is used for all the interconnection networks; this is turn leads to a single hardware design for a large number of interconnection networks with an arbitrary number of nodes.

This paper has been organized as follows. The proposed router organization along with its features are described in Section 2. In Section 3 we show the capabilities of the proposed approach with a detailed description of the mapping of three tree interconnection network routing algorithms: the binary tree algorithm, and oblivious hypertree algorithm and a fault-tolerant hypertree algorithm. A summary of the results for all the studied tree interconnection network topologies is included. A comparison with other flexible routing approaches is provided in Section 4. Some concluding remarks are included in Section 5.

2 Bit-Pattern Associative Router

The proposed bit-pattern associative router scheme has been designed to support intrinsic routing algorithms which are used in most router schemes. In intrinsic routing the topological characteristics of the interconnection network are inherent in the network addressing scheme. Sets of bits in a node's address contain information about the node's position in the network and the topology. To make a routing decision, a routing algorithm examines the status of these bit fields in the destination node address. An output port is chosen based on a comparison with the corresponding fields in the current node address. The current and

destination node addresses contain sufficient information to make a routing decision.

2.1 Bit-Pattern Associative Router Approach

In this section we present the basic requirements for executing intrinsic routing algorithms and how these requirements are fulfilled by the bit-pattern associative router. Using intrinsic routing algorithms to determine the output port requires consideration of the following two issues:

- *Some addressing bits need be ignored depending on the current and the destination nodes.* A routing algorithm examines the status of addressing bits that are of importance to determine the output port. This is accomplished by discarding the bits that provide no useful information because of the current node's position in the network.

- *An output port should be selected.* A number of routing alternatives may be available between source and destination nodes. Choosing an output port may be determined by the need to provide a short path as well as deterministic algorithm execution. This in turn requires a selection approach to favor a port that would yield a short path; this feature should be included in the scheme. In the case of the processor this priority is determined by the program sequence.

The proposed bit-pattern associative router addresses these issues by providing a bit-pattern matching mechanism that allows all the alternatives to be considered in parallel. Figure 2 shows the basic scheme of bit-pattern associative router. The modules along with the mode of operation for this router are described below.

1. *Destination address used as input.* The destination address is passed to the router by an input port (this destination address is part of the header of a message). This address becomes the search argument for the pattern associative mechanism.

2. *Routing Function.* The destination address is compared to the current address by means of a bit-pattern matching mechanism (explained in detail later). An associative computation with all the bit-patterns is performed to determine potential routing paths.

3. *Selection Function.* If more than one condition is satisfied only one assignment should be processed based on a priority scheme. For an oblivious router, the other matches are ignored. For an adaptive router, the other matches may be selected as alternatives to the port assignment.

4. *Output port assignment.* The output port assignment is made by selecting the output port associated with the selection condition.

The patterns stored in the bit-pattern match unit allow the router to make a decision about the destination port based on the routing algorithm. The address of a destination node D $(d_{n-1}, \ldots d_0)$ needs to be compared to the current node's address C $(c_{n-1}, \ldots c_0)$. A routing algorithm compares the bits of the two addresses; some bits are ignored since they

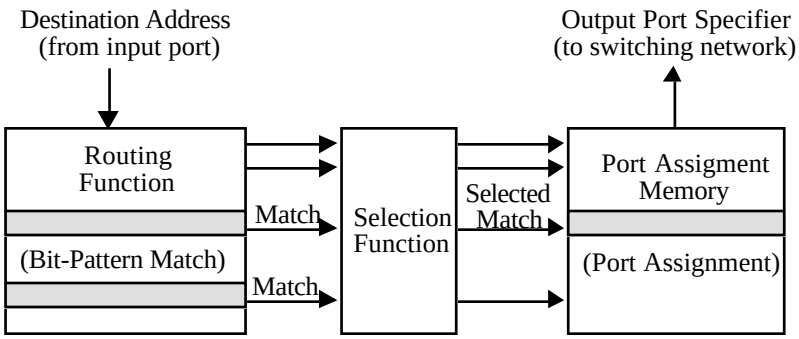


Figure 2: Bit-pattern associative router scheme.

do not affect the current routing decision. These "don't care" bits usually occur at different positions for each potential path being considered and must be customized according to the routing algorithm requirements. This in turn imposes a requirement for the bit-pattern matching unit to include "don't care" bits at different positions in each entry. To provide the flexibility required to support multiple interconnection networks and routing algorithms the routing function must be programmable. In the following sub-section we show how some basic programming statements can be implemented using the bit-pattern associative approach.

2.2 Routing Function Programming Using a Bit-Pattern Associative Approach

To implement or execute a routing algorithm, it is necessary to map such algorithm onto the proposed bit-pattern associative approach. Mapping could be considered as coding the algorithm onto a program. In this section, we introduce some of the basic programming statements that are present in most routing algorithms [17]. In Section 3 we provide specific examples of how these statements can be used and translated onto a set of bit-patterns.

IF_THEN Statement Each entry in the bit-pattern matching unit could be considered as an IF_THEN statement. This statement has the form

IF *condition* THEN *assign*(*port*).

The condition is true when $S(D) = S(C)$, where $S(D)$ specifies the selected bits from the input to the matching unit (which is usually the destination address) and $S(C)$ specifies the corresponding bits from the current bit-pattern (which is usually based on the source address). This condition requires that all the corresponding selected bits of D and C be equal. By allowing $\prod$ to express a logical AND operation of all the terms we have that

$$condition = \prod_{i=0}^{n-1} \left(s_i \cdot \left(d_i \cdot c_i + \overline{d}_i \cdot \overline{c}_i \right) + \overline{s}_i \right), \tag{1}$$

where s_i represents the selection condition for bit position i. The ability to select each bit position allows the condition to take the form of a single field comparison or a

compound expression composed of several fields logically ANDed together. In the bit-pattern associative router, the statement to be conditionally executed takes the form of an output port assignment in the port assignment memory.

A composed condition that requires an OR combination of two conditions has the following structure:

IF ($condition_1$ OR $condition_2$) THEN $assign(port_A)$.

This composed condition requires two IF_THEN statements to execute as follows:

IF $condition_1$ THEN $assign(port_A)$
IF $condition_2$ THEN $assign(port_A)$

If either condition evaluates true then the port assignment is made.

IF_THEN_ELSE and CASE Statements The IF_THEN statement can be extended to form more complex control structures that the bit-pattern associative router can execute. An IF_THEN_ELSE structure of the form

IF $condition_j$ THEN $assign(port_A)$ ELSE $assign(port_B)$

requires two bit-patterns to execute. The evaluation of the condition is the same as that in the IF_THEN statement; using a similar notation we have that

$$condition_j = \prod_{i=0}^{n-1} \left(s_{j,i} \cdot \left(d_i \cdot c_{j,i} + \overline{d}_i \cdot \overline{c}_{j,i} \right) + \overline{s}_{j,i} \right), \tag{2}$$

where the subscript notation (j, i) indicates the bit i of bit-pattern j. The ELSE part of the structure is implemented by setting $s_{j+1,k} = 0$ for $0 \leq k \leq n-1$ such that

$$\prod \left(\overline{s}_{j+1,k} \right) = 1 \tag{3}$$

Although the IF and ELSE conditions are evaluated in parallel, the selection function ensures deterministic execution by imposing a priority on the bit-patterns; thus, the ELSE statement will only be executed if the condition evaluates to false. An extension of the IF_THEN_ELSE control structure may contain several nested IF_THEN_ELSE statements. This structure has the form:

IF $condition_1$ THEN $assign(port_1)$
ELSE IF $condition_2$ THEN $assign(port_2)$
$\vdots$
ELSE IF $condition_N$ THEN $assign(port_N)$
ELSE $assign(port_{N+1})$

Each of the ELSE IF statements can be implemented with one bit-pattern. The conditions are set as follows:

$$\begin{array}{l} condition_1 = \prod_{i=0}^{n-1} \left(s_{1,i} \cdot \left(d_i \cdot c_{1,i} + \overline{d}_i \cdot \overline{c}_{1,i}\right) + \overline{s}_{1,i}\right) \\ condition_2 = \prod_{i=0}^{n-1} \left(s_{2,i} \cdot \left(d_i \cdot c_{2,i} + \overline{d}_i \cdot \overline{c}_{2,i}\right) + \overline{s}_{2,i}\right) \\ \vdots \\ condition_N = \prod_{i=0}^{n-1} \left(s_{N,i} \cdot \left(d_i \cdot c_{N,i} + \overline{d}_i \cdot \overline{c}_{N,i}\right) + \overline{s}_{N,i}\right) \\ condition_{N+1} = \prod_{k=0}^{n-1} \left(\overline{s}_{N+1,k}\right) = 1 \end{array} \tag{4}$$

Although all the conditions are evaluated in parallel, the selection function enforces the sequence of the structure. A CASE statement can be coded onto the bit-pattern associative router using this IF_THEN_ELSE structure coding.

Iteration Structures A number of routing algorithms may be expressed in a loop form to avoid repetitious code sequences. Iteration structures can be coded onto the bit-pattern associative router by unrolling the loops. Each iteration of the loop is executed in parallel and determinism is maintained by the selection function. The format of the iteration structures is similar to Eq. 4.

3 Mapping of Implicit Routing Algorithms For Tree Networks

In this section we present the mapping of implicit routing algorithms onto the proposed bit-pattern associative router. Mapping an algorithm onto the proposed router results in a set of bit-patterns that implements the algorithm. The mapping of an algorithm is obtained through a study of the interconnection network features, node labeling scheme, and routing algorithm requirements.

Without losing the generality of the proposed approach, we present a set of routing applications of the proposed bit-pattern associative router scheme: tree interconnection network routing algorithms. Tree interconnection networks have been chosen to show the features of the proposed router scheme because they tend to have more lengthy and complex routing algorithms [17]. A tree interconnection network is a static topology network, which is characterized by a one-to-many relationship among its component nodes. The structure consists of a unique first node, called the root, to which may be connected several successor nodes. Each of these nodes, in turn, may have several successors, and so on.

The logical organization of tree networks is hierarchical in nature. Thus, each node may be viewed as residing on a particular level of the tree. The root is at the top level; the root's successors are located on the second level. The hierarchical structure continues at each level such that a node at level n has its successors at level $n+1$. Terms such as parent, child, sibling, descendent and ancestor are commonly used to specify the relationship of different nodes in the hierarchical structure. A subtree is defined as a node taken together with all of its descendant nodes.

Algorithms that map naturally onto a tree structure would run efficiently on a parallel computer whose processors are hierarchically interconnected. These algorithms decompose

naturally onto the hierarchical structure of tree machines and include: searching, sorting, and other dictionary and database operations image and signal processing operations and neural networks [1][8][14]. Tree machines are also well suited for reduction, divide-and-conquer, and dataflow programming paradigms [8][10][14]. Examples of tree machines include DADO2 and Non-Von [8] and the Connection Machine 5 [13].

In Section 3.1 the characteristics common to all oblivious algorithms as they pertain to the proposed approach are discussed. Detailed mappings of three tree routing algorithms are presented in the following sections. In Section 3.2 we study an oblivious implicit routing in the binary tree network. This forms the foundation for the mapping of the other tree interconnection networks. Mapping a routing algorithm for the hypertree network is developed by extension of the mapping of the binary tree in Section 3.3. A fault-tolerant hypertree routing algorithm is examined in Section 3.4. We have selected these examples to show the capabilities of the proposed scheme. It is possible to map the routing algorithms of other topologies. Routing algorithms for 37 interconnection networks have been mapped onto the proposed router and reported in [20]. In Section 3.5 we present a summary of the results for the tree interconnection networks.

3.1 Routing on the Bit-Pattern Associative Router

A routing algorithm always yields the same path for a given source-destination node pair and network status (for adaptive and fault-tolerant routing). This deterministic nature of routing algorithms imposes a fixed priority of the alternative paths. Thus, the selection function needs to implement a priority approach. In each entry of the bit-pattern match unit a different bit-pattern (based on the current node's address) is stored. Figure 3 shows how the patterns and the port assignments are stored in the router. The numbers that are shown at the left of the patterns represent the fixed priority; the bit-pattern with the highest priority is given label 1 while the lowest is assigned label M. The highest priority is usually assigned to the most specific of the matched bit-patterns and the lowest to the most general. The representation shown in Figure 3 is used in the rest of this study. For the sake of simplicity the destination address and the output port specifier will not be shown.

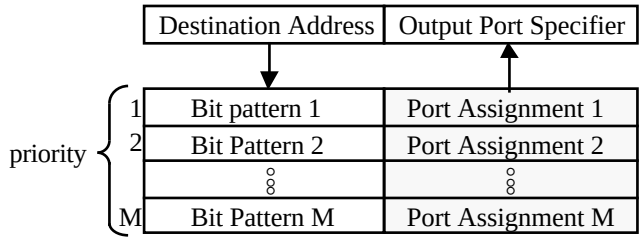


Figure 3: Bit-pattern and assignment entries and their priority.

3.2 Mapping of an Oblivious Binary Tree Routing Algorithm

The binary tree structure is the fundamental building block for most of the tree networks [10]. The binary tree structure is distinguished from that of the generalized tree structures by the

fact that no node has more than two children. These two children are commonly referred as the left and right child of the node. The left (or right) child and its descendants are referred to as the left (or right) subtree of the node. Without losing generality we have adopted the odd-even addressing scheme proposed by Horowitz and Zorat [10]. In this scheme, each node address has the format $(0, \ldots 0, 1, c_{k-1}, \ldots c_0)$, where the leftmost '1' bit is referred to as the leading 1 and the bits $(c_{k-1}, \ldots c_0)$ are referred to collectively as the significant portion of the node address. The root is assigned the address of $(0...0, 1)$. As the tree is traversed downward, the address of each parent node is modified and passed on to its children. The leading 1 of the parent's address is replaced with a '0' if the address is being determined for the left child or with a '1' if it is for the right child. A new leading 1 is then appended before the significant portion of the node address. A labeled binary tree with four levels is shown in Figure 4(a), where the levels are numbered starting with the root level as zero.

The routing algorithm for the binary tree is described in [10]. In this algorithm, a message is first routed up the tree to a subtree containing the destination node. Once the correct subtree is reached, the algorithm sends the message down through this subtree towards it destination. We refer to these as the upward and downward phases of the algorithm. For a message arriving at node C $(0, \ldots 0, 1, c_{k-1}, \ldots c_0)$ on level k destined for another node D $(0, \ldots 0, 1, d_{n-1}, \ldots d_0)$ at level n a distributed routing algorithm is shown in program-like fashion below.

```
BEGIN
  CASE_OF_k
    (k = n): IF(D = C)
                THEN: assign(thisnode)                    Assignment 1
                ELSE: assign(parent)                      Assignment 2
    (k > n): assign(parent)                               Assignment 3
    (k < n): IF ( D is in the subtree of C )
                THEN: IF( D is in the left subtree of C )
                          THEN: assign(left)              Assignment 4
                          ELSE: assign(right)             Assignment 5
                ELSE: assign(parent)                      Assignment 6
  END_CASE
END.
```

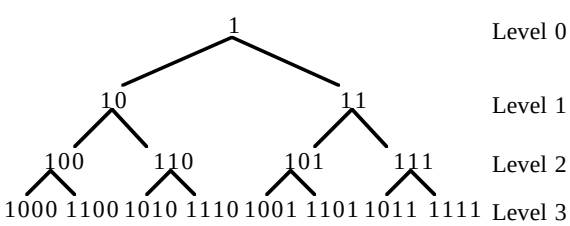
The algorithm consists of conditional program statements. Each of these program statements is associated with a potential output port assignment that could be made by the algorithm. These potential assignments are shown to the right of the algorithm. To map this routing algorithm onto the bit-pattern associative router, each conditional program statement needs to be expressed as a bit-pattern. This is accomplished using the properties of the addressing scheme. The conditions required by this algorithm include:

- *Comparing the levels of the source and destination node addresses.* The number of bits in the significant portion of a node address is equal to the level number. Thus the position of the leading 1 indicates a node's level in the tree. For node C $(0, \ldots 0, 1, c_{k-1}, \ldots c_0)$ at level k the leading 1 occurs at position k. If nodes D and C are at the same level $(k = n)$, then their leading 1 must occur at the same bit position. If node D is higher (lower) in the tree than node C, then the bit position of the leading 1 in the address of D is to the right (left) of the leading 1 in the address of C.

- *Determining if a destination node is in the current node's subtree.* In this addressing scheme, the child node inherits the significant portion of the parent's address. All nodes in the subtree of node C $(0, \ldots 0, 1, c_{k-1}, \ldots c_0)$ will have the bits $(c_{k-1}, \ldots c_0)$ in common. Furthermore, a node in the left subtree of C will have $(c_k = 0)$ and a node in the right subtree will have $(c_k = 1)$.

The bit-pattern entries for this binary tree routing algorithm are given in Figure 4(b). Assignment 1 in the algorithm routes messages destined for the current node. The bit pattern corresponding to this condition is the current node's address. This is shown in the first entry in Figure 4(b). Assignment 2 requires the destination node to be on the same level in the tree hierarchy; thus $(k = n)$ and the destination address word is led by 0's up to d_{k+1} with $(d_k = 1)$. The remaining bit positions are not important to the determination of the output port and are set to the "don't care" condition. This results in the bit-pattern $(0, \ldots 0, 1, X_{k-1}, \ldots X_0)$. Assignment 3 requires the destination node to be higher in the hierarchy $(k > n)$; in this case the destination address word is led by 0's at least up to d_k. This results in the bit-pattern $(0, \ldots 0, 0, X_{k-1}, \ldots X_0)$. Since assignments 2 and 3 result in the same output port and the bit-patterns differ only in position d_k they can be combined into one pattern with d_k set to "don't care". This is shown as entry 2 in Figure 4(b).

Assignments 4 and 5 (entries 3 and 4 in Figure 4(b), respectively) require the destination node to be within the current node's subtree $(d_{k-1}, \ldots d_0 = c_{k-1}, \ldots c_0)$. Here, the bit d_k is checked to assign the output port (left or right). The remaining bit positions are set to "don't care". If none of the above conditions are met, then the destination is at or below the level of the current node and falls within a different subtree. In this case the message is sent to the parent port (assignment 6). The last bit-pattern will be selected if none of the other bit-patterns match.



(a) Node addressing in a binary tree.

	m	k	k-1 ... 0	
1	0 ... 0	1	c_{k-1} ... c_0	this node
2	0 ... 0	X	X ... X	parent
3	X ... X	0	c_{k-1} ... c_0	left
4	X ... X	1	c_{k-1} ... c_0	right
5	X ... X	X	X ... X	parent

(b) Binary tree bit-patterns.

Figure 4: Mapping of an oblivious routing algorithm for the binary tree.

The same set of bit-patterns can be used to route messages in a fat tree interconnection network [18]. For a general m-ary tree, each bit c_i in Figure 4(b) is replaced by $\lceil \log_2(m) \rceil$ bits. The number and structure of the patterns would remain the same.

3.3 Mapping an Oblivious Hypertree Routing Algorithm

A hypertree interconnection topology is a binary tree network augmented with additional horizontal connections called n-cube links [9]. The hypertree structure is shown in Figure 5.

An n-cube link provides an interconnection path between two nodes in different subtrees at the same level of the tree structure. The addresses of these two nodes differ in only one bit. Using the same addressing scheme as the binary tree, we have that for a node C at level k $(0, \ldots 0, 1, c_{k-1}, \ldots c_{i+1} c_i c_{i-1}, \ldots c_0)$ the node connected by an n-cube link to node C will have address $B(0, \ldots 0, 1, b_{k-1}, \ldots b_{i+1} b_i b_{i-1} \ldots b_0)$, where $b_i \neq c_i$ and $b_j = c_j$ for all $j \neq i$. We refer to B as the hypernode of C. The bit position in which the two node addresses differ is based on the type of hypertree as well as the level in the tree at which the nodes reside. Details can be found in [9].

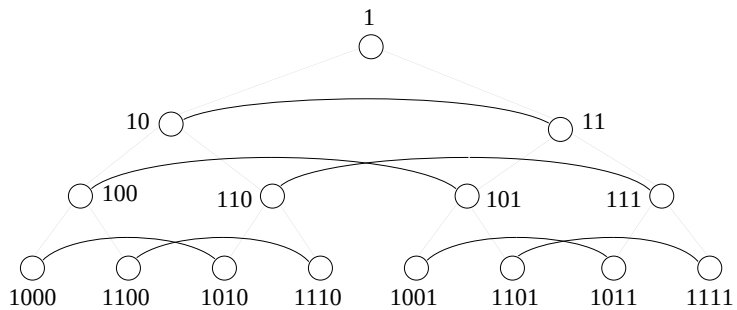


Figure 5: Hypertree interconnection network.

We have adopted the oblivious routing algorithm for hypertree presented in [9]. This routing algorithm is based on the binary tree with provision made for the n-cube links. As in the basic binary tree algorithm, messages are routed up the tree to a subtree containing the destination node. During this upward phase of the algorithm, however, an n-cube link is traversed if it reduces the Hamming distance between the source and destination addresses. This has the effect of shortening the path between source and destination nodes.

To map this routing algorithm onto the pattern associative router the binary tree bit-patterns in Section 3.2 can be used to implement the binary tree portion of the algorithm. These bit-patterns are augmented to allow the use of an n-cube link during the upward phase of the algorithm. The conditions required to use an n-cube link are:

- *The Hamming distance between the addresses of the current node C and the destination node D must be greater than that between the addresses of B (the hypernode of C) and D.* The addresses of nodes B and C differ only in bit position i. Let $d(x, y)$ represent the Hamming distance between the addresses of nodes x and y. Then $d(B, D) < d(C, D)$ requires $d_i = b_i$ or, alternatively, $d_i \neq c_i$.

- *The routing algorithm must be in its upward phase.* For the routing algorithm to be in its downward phase, node D must be in the subtree of node C. This requires that the bits of D and C corresponding to the significant portion of the address of C be equal. Since bit position i falls within the significant portion of the address of C, $d_i \neq c_i$ is sufficient to ensure that the algorithm is in it upward phase.

The condition to use an n-cube link can thus be represented by the bit-pattern $(X, \ldots X\bar{c}_i X, \ldots X)$. The set of bit-patterns to implement this hypertree routing algorithm are shown in Figure 6.

1	0 ... 0	1	c_{k-1}	. . .	c_0	this node
2	X ... X	X	X . . .X	$\overline{c_i}$	X . . .X	n-cube link
3	0 ... 0	0	X_{k-1}	. . .	X_0	parent
4	X ... X	0	c_{k-1}	. . .	c_0	left
5	X ... X	1	c_{k-1}	. . .	c_0	right
6	X ... X	X	X	. . .	X	parent
	m	k	k-1		0	

Figure 6: Hypertree Bit pattern set.

3.4 Fault-Tolerant Hypertree Routing Algorithm

In an oblivious routing algorithm the n-cube links in the hypertree structure are used to provide shorter paths between nodes as well as to distribute message traffic more evenly throughout the network [9]. The n-cube links can also be used to provide alternate paths to implement adaptive and fault-tolerant routing algorithms. In this section we consider the mapping of a fault-tolerant algorithm for the hypertree interconnection network onto the bit-pattern associative router. This algorithm has been included to show the flexibility of the proposed approach. For simplicity, a partially fault-tolerant algorithm has been chosen; this algorithm is tolerant to any single link failure in the upward phase of the oblivious hypertree algorithm. The partially fault-tolerant algorithm is based on an algorithm in [9] that is tolerant to any single link fault; the mapping of this algorithm can be found in [20].

The fault-tolerant routing algorithm considered here is based on the oblivious hypertree routing algorithm presented in Section 3.3. This algorithm is tolerant to any single link failure during the upward phase of the hypertree routing. Under normal network conditions, a message is routed according to the hypertree routing algorithm. When a faulty link is encountered during the upward phase of the algorithm the message is sent on a detour around the fault. If a faulty n-cube link is encountered in the upward phase of the routing, the message is routed around the fault by reverting to the binary tree routing algorithm (it is sent to the parent of the current node). At the next node on the path the message resumes the normal hypertree routing. If a faulty tree link is encountered during the upward phase, the message is routed first to the hypernode and then one level up the tree. With the fault bypassed, the normal hypertree routing is resumed. Note that unlike the detour around a faulty n-cube link, the detour around a faulty tree link is a two hop detour. The hypernode on the detour must know that the message it has just received is on a detour. Without this information, the message would be returned to the node with the fault (via the normal hypertree routing algorithm) and a livelock condition would exist.

To map this routing algorithm onto the bit-pattern associative router the hypertree bit-patterns in Section 3.3 will be augmented with bit-patterns to provide support for the detours. A fault-tolerant algorithm requires knowledge of the fault status of the node links. This algorithm requires information of faults local to the current node. In any routing approach, this information must be provided to the router. The condition required by the fault-tolerant algorithm is:

- *Determining whether a message is on a detour.* Although this information could be explicitly carried in the message header, this requires modifying message headers. Another approach is to infer whether a message is on a detour based on the input port from which the message has arrived as well as information on the destination node address. In this routing algorithm the only multiple node detour is that around a tree link. When the message arrives at the hypernode of the node with the faulty link, the hypertree routing algorithm will tend to send the message back across the n-cube link. Since this condition could never occur in the oblivious hypertree algorithm, its existence implies that the message is on a detour.

The bit-patterns for the fault-tolerant hypertree algorithm are shown in Figure 7. In addition to the destination address, the link fault status as well as the input port from which the message has arrived are used as input to the router. It should be noted that this additional information requires no hardware modification to the proposed router approach; additional information is simply provided as input to the router. Routing around a faulty n-cube link is accomplished by bit-pattern entry 3. If the n-cube link is faulty and the condition for using the n-cube link in the hypertree algorithm is met, then the message is routed to the parent of the current node. Routing around a faulty tree link is handled by entries 1, 5, and 9. Entries 5 and 9 route the message to the n-cube link if the parent link is faulty and the destination address meets one of the conditions for using the parent link in the hypertree routing algorithm. Entry 1 matches the condition that the message has arrived from the n-cube link and the destination address of the message meets the condition for using the n-cube link. This indicates that the message is on a detour around a faulty tree link and should be routed up the tree. The remaining bit-pattern entries correspond to the oblivious hypertree routing algorithm. This partially fault-tolerant algorithm requires 10 bit patterns to implement. A fully fault-tolerant algorithm for the hypertree network [20] requires 15 bit-patterns to map onto the bit-pattern associative router. Both algorithms map into $O(degree)$ bit-patterns.

Adaptive routing algorithms are mapped onto the bit-pattern associative router using a method similar to mapping fault-tolerant algorithms. In this case, the input to the router would consist of the destination address and network status information required by the algorithm. An example can be found in [20].

3.5 Summary

In Table 1 we include the number of patterns required to implement an oblivious routing algorithm as well as the degree for 10 tree interconnection networks we have studied. The details of the mapping of these networks can be found in [18][19].

We have considered the output port to the current node as contributing to the node degree since it represents a routing alternative. Using these values we have computed the ratio between the number of patterns and the degree. The degree of a node represents all potential routing alternatives. This in turn represents the minimum number of patterns that a routing algorithm can be mapped onto. The number of patterns required to implement an oblivious implicit routing is $O(degree)$ for each the studied networks. The results for 37 interconnection networks which include the tree, cube, array and multistage interconnection

	Fault	Input Port						
1	X	n-cube	X ... X	X	X . . . X	$\overline{c_i}$	X . . . X	parent
2	X	X	0 ... 0	1	c_{k-1}	. . .	c_0	this node
3	n-cube	X	X ... X	X	X . . . X	$\overline{c_i}$	X . . . X	parent
4	X	X	X ... X	X	X . . . X	$\overline{c_i}$	X . . . X	n-cube link
5	parent	X	0 ... 0	0	X_{k-1}	. . .	X_0	n-cube link
6	X	X	0 ... 0	0	X_{k-1}	. . .	X_0	parent
7	X	X	X ... X	0	c_{k-1}	. . .	c_0	left
8	X	X	X ... X	1	c_{k-1}	. . .	c_0	right
9	parent	X	X ... X	X	X	. . .	X	n-cube link
10	X	X	X ... X	X	X	. . .	X	parent
			m	k	k-1		0	

Figure 7: Bit pattern set for fault-tolerant hypertree routing algorithm.

networks can be found in [20]. The average ratio between the number of patterns and degree for all the studied interconnection networks is 1.35 [20]. The average ratio for the tree interconnection networks listed in Table 1 is 1.72. This clearly indicates that the proposed router approach tends to map implicit routing algorithms onto a very small number of bit-patterns.

Table 1: Bit-pattern and degree characteristics of tree network topologies [20].

Topology	Patterns	Degree	Ratio
Binary tree, Fat tree	5	4	1.25
General m-ary tree	m+3	m+1	≈1.0
Hyper tree	6	5	1.2
Flip tree	max(trees+1,8)	max(trees,4)	2.0
Tree w. ring	9	6	1.5
Tree w. half ring	8	5	1.6
Quadtree	7	6	1.17
Diamond	9	4	2.25
Tree of Meshes	9	5	1.8

4 A Comparison with Other Flexible Routers

In this section we compare the proposed bit-pattern associative router with other flexible routers. These flexible routers include the dedicated processor, look up table, and interval routers. In the dedicated processor routing approach [15][17] a customized processor is used to execute routing algorithms. The ability to store programs and execute algorithms allows this type of router to accommodate a large number of interconnection networks. The time to compute a routing algorithm is not only long due to the sequential execution

but also is unpredictable due to the dynamic instruction count variations. An oblivious routing processor architecture that provides flexibility to support over 40 interconnection networks has been reported in [17]. In this approach the average static instruction count required to implement a routing algorithm is of the order of 20 instructions. This figure does not account for the variation in program length due to network size [17]. In addition, the routing algorithm execution delay can vary significantly with each invocation of the same algorithm since the dynamic instruction count is dependent on program input and the program sequence. The dedicated processor routing approach yields larger execution delays with higher hardware requirements than the other flexible routers. Since the routing algorithm execution lies on the critical path [2], the latency of this approach tends to be dominated by the this delay. For these reasons we do not consider this approach any further.

Look up table routers require tables of $O(N)$ entries at each node, where N is the number of nodes in the system [4], to store a routing path to each possible destination node. To reduce the size of the look up tables, van Leeuwen and Tan [21] introduced the interval routing. This method allows the table size to be scaled down to $O(d)$, where d is the node degree. This method relies on adapting a suitable labeling scheme. The nodes have to be labeled in an order that allows the output ports of a node to uniquely identify ranges of addresses. Thus an interval of addresses can be associated to each output port. The destination address of a message is evaluated to determine which range contains that address. The output port is then assigned based on the results of the comparisons with the limits of each range. Figure 8 illustrates a binary tree addressed and labeled according to an interval routing scheme. The nodes are assigned sequential addresses, starting with 1, using a depth-first-search technique. Interval routing is considered as an explicit routing algorithm since each port has been assigned with a specific set of interval addresses. The topology of the interconnection network is not taken into account when selecting the output port. The interval routing algorithm is being used by INMOS to form the interconnection network for multiprocessor Transputer-based systems [11]. Potential drawbacks of the interval routing approach include: minimum distance path is only possible for a subset of interconnection networks [4], inserting a new node in the network usually results in a new labeling of the nodes and new assignment of ranges [21], and single assignment from a number of potential routing paths is obtained [4].

The bit-pattern associative router can execute any interval routing. The bit-patterns associated with each output port are based on the node ranges assigned by the interval routing. Specifically, each output port is assigned a switching function $f_P()$ of the n bits of the destination address D such that

$$fp(d_{n-1}, d_{n-2}, \ldots d_0) = \begin{cases} 0 & \text{if } D \text{ is not in an interval associated with port P,} \\ 1 & \text{if } D \text{ is in an interval associated with port P.} \end{cases}$$

When placed into a sum-of-products form, the function can be seen as the composed condition of several compound bit fields logically ORed together. As shown in Section 2, this structure can be programmed onto the bit-pattern associative router by assigning a bit-pattern to each product term of the expression. The expressions can be minimized to reduce the number of entries.

As example of this approach we show the mapping of the interval routing for node 4. The interval routing table is summarized in Figure 9(a) and the bit-patterns to implement this

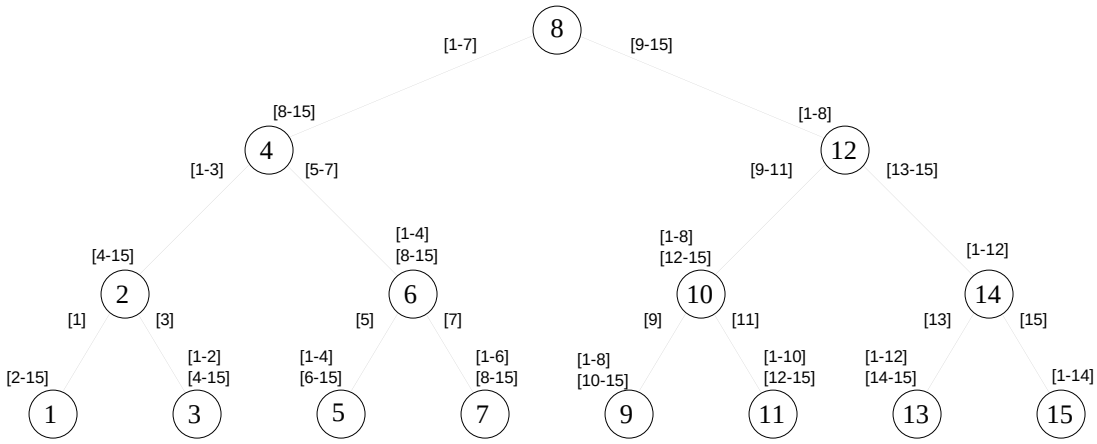


Figure 8: Binary tree addressed and labeled according to an interval routing scheme.

interval routing are in Figure 9(b). The first interval represents routing to the left child node and the function for that node can be specified as $f_{left}(d_{n-1}, d_{n-2} \ldots d_0) = 1; D \in \{1, 2, 3\}$. Given that address (0) is not used in this interval routing and can be considered as a "don't care", $f_{left}(D) = (0000 + 0001 + 0010 + 0011) = 00XX$. This corresponds to bit-pattern 1 in Figure 9(b). The second interval is a single address which specifies the current node. Thus $f_{current}(D) = 0100$, shown as the second bit-pattern in Figure 9(b). The third interval specifies that $f_{right}(D) = (0101 + 0110 + 0111)$. , corresponding to bit-pattern 3. Note that since this is an oblivious routing, the priority function eliminates node address 4 from further consideration, allowing it to be used to simplify the expression for $f_{right}(D)$. Thus, the third interval can be specified as $f_{right}(D) = (0100 + 0101 + 0110 + 0111) = 01XX$. The last interval defines $f_{parent}(D) = 1XXX$, shown as the last bit-pattern in Figure 9(b). Using this method, any routing algorithm for any node addressing scheme and network topology can be mapped onto the proposed router.

1	1 ≤ D ≤ 3	left
2	4	this node
3	5 ≤ D ≤ 7	right
4	8 ≤ D ≤ 15	parent

(a) interval table.

1	0 0 X X	left
2	0 1 0 0	this node
3	0 1 X X	right
4	1 X X X	parent

(b) corresponding bit patterns

Figure 9: Mapping an interval routing on the bit-pattern associative router.

Table 2 provides a comparative list of features for the bit-pattern associative, interval, and lookup table routings. It can be observed that the topology of the interconnection network plays a key role in the determination of the output port for the pattern based approach; this topology is stored in the bit-patterns. This in turn provides expandability to our approach; more nodes could be added to the network with minor modifications. These modifications

Table 2: Comparison of the flexible routing schemes.

Feature	Bit-Pattern Associative	Processor	Lookup Table	Interval
Routing algorithm	implicit	implicit	explicit	explicit
Stored data	bit-patterns	instructions	addresses	addresses
Storage Requirements	O(degree)	Program Size	O(network size)	O(degree)
Expandability requirements	minor	minor	reprogram & expand tables	relabel all node addresses & ranges
Assignment delay	1 comparison and 1 read	Requires sequential algorithm execution	1 decode and 1 read	1 comparison and 1 read
Programmability	yes	yes	yes	yes
Alternative paths	yes	yes	no	no

occur mainly at the nodes that the expansion nodes are connected to. For the interval and look up table routing approaches, all the intervals or look up tables need to be modified for all the nodes because of a complete relabeling of the entire interconnection network. Using the pattern based approach it is possible to generate alternative paths; in this paper, we have not discussed this feature. The three approaches have a similar assignment delay, which makes them have a comparable performance as the dedicated routers. The look up table, due to the large number of stored entries, requires much more hardware as the network becomes large. The bit-pattern associative approach, with its programming capabilities, imposes fewer restrictions than the interval or look-up table approaches.

5 Concluding Remarks

In this paper we have presented a novel routing algorithm decoding scheme that is applicable to a wide range of interconnection network topologies. To show the capabilities of the proposed approach, we have studied implicit oblivious and fault-tolerant routing algorithms for tree interconnection network topologies. A list of the main features of the proposed bit-pattern associative router is provided below.

- *A wide range of implicit distributed routing algorithms can be implemented.* In this paper, implicit tree routing algorithms have been mapped onto bit-patterns. However, we have performed a broader study of 37 interconnection networks which includes cube, array, multistage, and tree network topologies [20]. The output port is determined on the basis of the current node address, destination node address, and the interconnection network structure.

- *Routing algorithms are mapped onto O(d) bit-patterns.* For all the studied networks, the number of bit-patterns required to implement the routing algorithms is $O(d)$, where d is the node degree. Each reachable non-redundant port in a node has to be considered for assignment at least once in any routing scheme. This in turn imposes a minimum bound on the number of entries equal to the node degree. The average ratio between the number of patterns and degree for the tree topologies is 1.72; however, for all the

studied interconnection networks is 1.35 [20]. Thus, the proposed approach tends to map routing algorithms into a number of entries very close to the smallest number.

- *The pattern-based router requires one comparison and one read delays.* Since the bit-pattern match can be done in parallel, the proposed approach requires 1 comparison delay. Once a match is found 1 read delay is required to get the assigned output port. Within the flexible and programmable router schemes, the proposed bit-pattern associative is one of the fastest approaches.

- *Network expandability needs no address reassignment or bit-pattern modification.* For all the studied topologies, generic bit-pattern sets have been developed to accommodate the specified routing algorithm for arbitrary network sizes. Network expansion (as allowed by the topology) requires no modification of the bit-pattern sets or node addresses.

We have introduced the bit-pattern associative routing approach in this paper. As an example of its applicability the mappings of a number of routing algorithms for tree topologies have been reported. This approach is not restricted to oblivious and fault-tolerant routing algorithms. Adaptive routing algorithms [2][3][7] can be mapped as well by using the status of the output ports as part of the input to the bit-pattern associative router. This router has been implemented on a custom VLSI chip and tested; results are reported in [6].

Acknowledgment

The authors are grateful to the anonymous referees of this paper for their helpful comments and suggestions, which greatly improved the quality of the paper.

References

[1] S. G. Akl, *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[2] A. A. Chien, "A Cost and Speed Model for k-ary n-cube Wormhole Routers," *Proceedings of Hot Interconnects '93*, Palo Alto, CA, August 5-7, 1993.

[3] W. J. Dally and H. Aoki, "Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466-475, April 1993.

[4] U. De Carlini and U. Villano, *Transputers and Parallel Architectures: Message Passing Distributed Systems.* New York: Ellis Horwood, 1991.

[5] J. G. Delgado-Frias, H. Ding, S. Vassiliadis, and D. Summerville, "A Survey on Interconnection Networks for Parallel Processor Organizations," *IBM Technical Report*, IBM, Endicott, NY 13790, July, 1997.

[6] J. G. Delgado-Frias, J. Nyathi, C. L. Miller, and D. H. Summerville, "A VLSI Interconnection Network Router using a D-CAM with Hidden Refresh," *IEEE Sixth Great Lakes Symposium on VLSI*, Ames, Iowa, pp. 246-251, March 1996.

[7] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320-31, Dec. 1993.

[8] R. Duncan, "A Survey of Parallel Computer Architectures," *Computer*, vol. 23, pp. 5-16, February, 1990.

[9] A. Esfahanian, L. M. Ni, and B. E. Sagan, "The Twisted N-Cube with Application to Multiprocessing," *IEEE Trans. on Computers*, vol. 40, no. 1, pp. 88-93, Jan. 1991.

[10] E. Horowitz and A. Zorat, "The Binary Tree as Interconnection Network: Applications to Multiprocessor Systems and VLSI," *IEEE Trans. on Computers*, vol.30, no.4, pp. 247-253, April 1981.

[11] INMOS Ltd., *The 9000 Transputer Products Overview Manual. INMOS document number: 72 TRN 228 00*, 1991.

[12] T. Leighton, "Average Case Analysis of Greedy Routing Algorithms on Arrays," *2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 2-10, Crete, Greece, 1990.

[13] C. E. Leiserson et. al., "The Network Architecture of the Connection Machine CM-5," *SPAA Symposium on Parallel Algorithms and Architectures*, June, 1992.

[14] C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison-Wesley Publishing, 1980.

[15] W. G. P. Mooij and A. Ligtenberg, "Architecture of a Communication Network Processor," *PARLE '89: Parallel Architectures and Languages Europe, E. Odijk, M Rem, and J-C. Syre (Eds.), Lecture Notes in Computer Science 365*, Berlin: Springer-Verlag, 1989.

[16] L. M. Ni and D. K. Panda, "A Report of the ICPP '94 Panel on Sea of Interconnection Networks: What's Your Choice?," *IEEE Computer Architecture Technical Committee Newsletter*, pp. 31-34, Winter 1994-1995.

[17] J. Park, S. Vassiliadis, and J. G. Delgado-Frias, "Flexible Oblivious Router Architecture," *IBM Journal of Research and Development*, vol. 39, no. 3, pp. 315-334, May 1995.

[18] D. H. Summerville, J. G. Delgado-Frias, and S. Vassiliadis, "A Pattern-Based Router for Tree Topology Implicit Routing Algorithms," *IBM Technical Report TR 51.0804*, IBM, Austin, TX 78758, Nov. 1993

[19] D. H. Summerville, J. G. Delgado-Frias, and S. Vassiliadis, "A High Performance Pattern Associative Oblivious Router for Tree Topologies," *IPPS '94: The 8th International Parallel Processing Symposium*, pp. 541-545, Cancun, Mexico, April 1994.

[20] D. H. Summerville, J. G. Delgado-Frias, and S. Vassiliadis, "A High Performance Flexible Router for Multiple Interconnection Networks," *IBM Technical Report*, IBM, Endicott, NY 13790, July, 1997.

[21] J. Van Leeuwen and R. B. Tan, "Interval Routing," *The Computer Journal*, vol. 30, no. 4, pp. 298-307, 1987.

Douglas H. Summerville received a BE degree in 1991 from The Cooper Union for the Advancement of Science and Art, New York, NY and MS and Ph.D. degrees in 1994 and 1997, respectively, from the State University of New York at Binghamton, all in electrical engineering. He is currently an assistant professor in the Electrical Engineering Department at the University of Hawaii at Manoa.

Previously, he has held positions as adjunct lecturer, teaching assistant, and research assistant at the State University of New York at Binghamton. In 1995 he received the Graduate Student Award for Excellence in Teaching. He holds memberships in the IEEE, Association for Computing Machinery, American Society for Engineering Education, Eta Kappa Nu, and Sigma Xi. His research interests include interconnection networks, parallel computer architecture and design, and VLSI design.

José G. Delgado-Frias received a BS degree from the National Autonomous University of Mexico, an MS degree from the National Institute for Astrophysics, Optics and Electronics, Mexico and a Ph.D. degree from Texas A&M University, all in electrical engineering.

He is with the Electrical Engineering Department at the State University of New York (SUNY) at Binghamton where he is an associate professor. He has held academic positions at the University of Oxford, England (as a post-doctoral research fellow) and the National Autonomous University of Mexico (as an assistant professor). His research interests include parallel computer architecture, interconnection networks, VLSI design, computer hardware organization, neural network computing machines, and optimization using genetic algorithms. He has co-authored over eighty technical papers and co-edited three books. He has been granted eight US patents.

He has been the co-chairman of three international workshops and a program committee member of a number of international conferences. In 1994, he received the State University of New York System Chancellor's Award for Excellence in Teaching. He is a senior member of the IEEE and a member of the Association for Computing Machinery, American Society for Engineering Education, and Sigma Xi.

Stamatis Vassiliadis is a professor in the Electrical Engineering Department at Delft University of Technology, The Netherlands. He has been also in the faculty of Cornell University and the State University of New York at Binghamton.

Previous working experience includes 10 years at IBM in the Advanced Workstations and Systems laboratory in Austin TX, the Mid-Hudson Valley laboratory in Poughkeepsie NY, and the Glendale laboratory in Endicott NY. He was involved in the design and implementation of the IBM 9370 model 60. A number of his inventions have been implemented in commercial systems and processors including the IBM POWER II, the IBM AS/400 Models 400, 500, and 510, Server Models 40S and 50S, and the IBM AS/400 Advanced 36. He received numerous awards including 23 levels of Publication Achievement Awards, 15 levels of Invention Achievement Awards and an Outstanding Innovation Award for Engineering/Scientific Hardware Design in 1989. Six of his patents have been rated with the

highest patent ranking in IBM and in 1990 he was awarded the highest number of patents in IBM.

His research interests include computer architecture, hardware design and functional testing of computer systems, parallel processors, computer arithmetic, EDFI for hardware implementations, and software engineering. He received a Dr. Eng. degree in electronic engineering from the Politecnico di Milano, Milan, Italy and a Ph.D. degree in computer science from the University of Namur (F.U.N.D.P.), Belgium. Dr. Vassiliadis is an IEEE Fellow.