in *IEEE Transactions on Parallel and Distributed Systems*, vol.7, no. 5, pp. 477-495, May 1996. ©1996 IEEE[1].

A Flexible Bit-Pattern Associative Router for Interconnection Networks

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

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

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

Abstract

A programmable associative approach to execute implicit routing algorithms is presented. Algorithms are mapped onto a set of bit-patterns that are matched in parallel. We have studied and mapped a large number of routing algorithms for a wide range of interconnection network topologies. Here we report three cases that illustrate the capabilities of the router scheme. For the studied topologies, the number of required bit-patterns is of the same order as the topology degree. The proposed approach is one of the fastest routers and requires a very small amount of hardware.

Keywords

Routing Algorithm Execution, Interconnection Networks, Bit-Pattern Associative Memories, Oblivious Routing, Adaptive Routing, and Flexible Routers.

I. INTRODUCTION

A number of interconnection network topologies have been proposed and used [10]; each network has features that make it suitable for a set of applications and algorithms. The interconnection network is often considered as the critical element in a multiprocessor machine due to the machine's performance sensitivity to network latency and throughput. 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 usually consist of two major components: the header (H), which contains the destination node address, and the body (B), which carries the message. Figure 1 shows a 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.

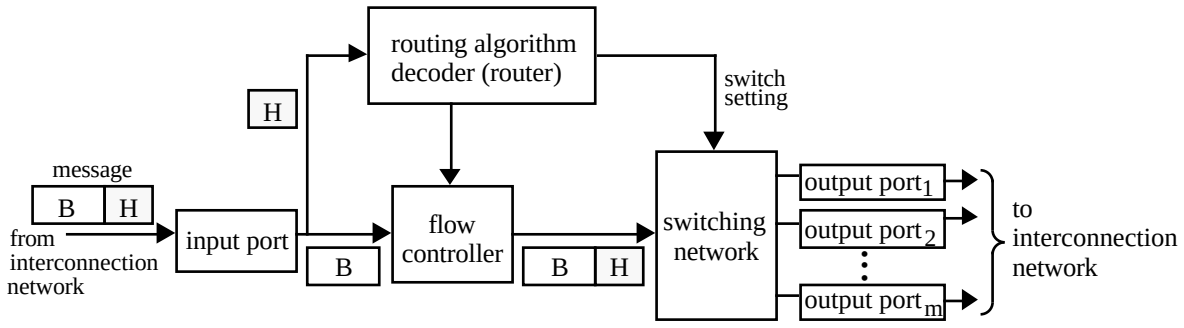

Fig. 1. Router system.

A crucial component of any large scale parallel machine is the routing algorithm [8]. Given the large number of interconnection network topologies, it is highly desirable to have a reconfigurable router. Such a router should provide flexibility to accommodate various network topologies and support to carry out a number of different routing algorithms. In addition to the routing algorithm execution, a flexible router has to be able to accommodate a number of routing

requirements including: expeditious determination of the destination port (to be comparable to specialized routers), 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 [9][11] and look-up tables [13]. A dedicated processor allows 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. To decrease the number of entries, the interval routing approach was proposed [16]. 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 [2].

In this paper we present a novel router approach that provides not only flexibility to execute implicit routing algorithms but also 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. In this paper we report a study of this approach as it applies to a large number of interconnection network routing algorithms. The set of studied interconnection networks includes four major categories: tree, cube, array, and multistage. It is shown that the same approach 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 II. In Section III we show the capabilities of the proposed approach with detailed descriptions of the mapping of both oblivious and adaptive routing algorithms onto the proposed router architecture. A comparison with other flexible routing approaches is provided in Section IV. Some concluding remarks are included in Section V.

II. 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. Since the topology is inherent in the addressing scheme, the current and destination node addresses contain sufficient information to make a routing decision.

A. 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 in [3][15]). 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.

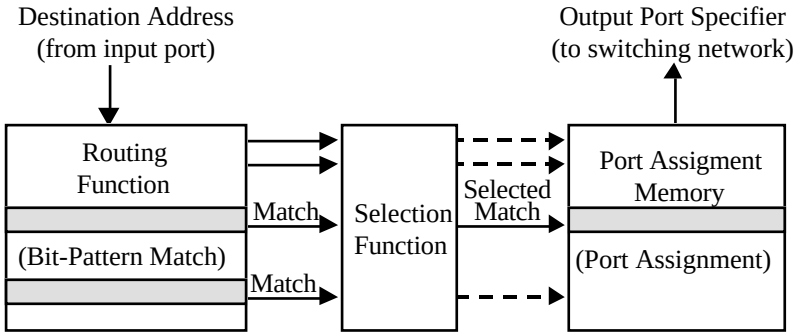

Fig. 2. Bit-pattern associative router scheme.

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 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.

B. Routing Function Programming Using a Bit-Pattern Associative Approach

To provide the flexibility required to support multiple interconnection networks and routing algorithms the routing function must be programmable. 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 the mapping of some basic programming statements that are present in most implicit routing algorithms [11]. 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. (1)

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). 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 requires two IF_THEN statements to execute as follows:

IF condition_1 THEN assign_port_A
IF condition_2 THEN assign_port_A (2)

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 requires two bit-patterns to execute, one for the IF condition and one for the ELSE condition. The structure would be similar to expression (2) with condition_2 set to the ELSE condition and the port assignment changed accordingly. 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 structure may contain several nested IF_THEN_ELSE statements. A CASE statement can be coded onto the bit-pattern associative router using this IF_THEN_ELSE structure coding. Thus these mappings can be considered as an extension of expression (2).

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.

C. Deterministic Execution of Routing Algorithms

An oblivious routing algorithm always yields the same path for a given source-destination node pair. Given the interconnection network status and the destination as input, most adaptive routing algorithms perform a deterministic assignment of the output ports. This deterministic nature of these routing algorithms imposes a fixed priority on the alternative paths. Thus the selection function needs to implement a static 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.

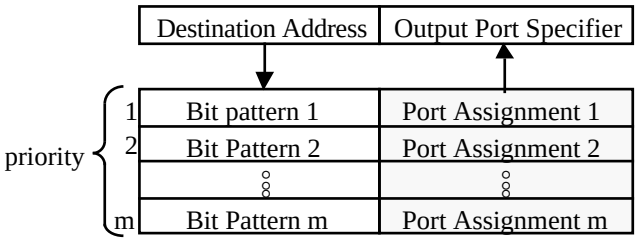

Fig. 3. Bit-pattern and assignment entries and their priority.

III. MAPPING OF IMPLICIT ROUTING ALGORITHMS

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. Detailed mappings of two oblivious routing algorithms are presented in subsections III-A and III-B. In subsection III-A we study an oblivious implicit routing for the binary tree network. A hypercube routing algorithm is examined in subsection III-B. To show the generality of the proposed approach, an adaptive routing algorithm for the hypercube is presented in subsection III-C. We have selected these examples because they are widely used and show the capabilities of the proposed scheme. However, 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 [14][15].

A. Mapping of an Oblivious Binary Tree Routing Algorithm

The binary tree structure is the fundamental building block for most of the tree networks [6]. 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 of the root and its descendants are referred to as the left (or right) subtree of the root. A full or complete binary tree has all of its leaves on the bottom level. Without losing generality we have adopted the odd-even addressing scheme proposed by Horowitz and Zorat [6]. In this addressing scheme the root is assigned the address of 1. As the tree is traversed downward, the address of each parent node is modified and passed on to its children. The high order bit 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 high-order bit of one is then appended. A labeled binary tree with four levels is shown in Figure 4.

Levels are numbered starting with the root level as zero. One consequence of this addressing scheme is that the network is split, with the even addresses falling into the left subtree of the root and the odd addresses into the right, hence the name odd-even addressing.

To efficiently implement a routing algorithm, the properties of the addressing scheme must be identified. The scheme proposed by Horowitz and Zorat posses the following properties:

- *Position of the leftmost 1 indicates the node's level in the tree.* Any node in the binary tree has the following address format $(0, \ldots 0, 1, c_{k-1}, \ldots c_1, c_0)$ where 'c' could be either 0 or 1 and k indicates the bit position. The leftmost bit equal to 1 is at position k; this in turn indicates that this node is at level k in the tree structure. The leftmost 1 bit is called the leading 1.
- *A node's lineage is contained in its address.* As addresses are passed from parent to child, the child node inherits all the parent's address bits except the leading 1 bit. The immediate descendants of node $(0, \ldots 0, 1, c_{k-1}, \ldots c_0)$

have the address format $(1, a, c_{k-1}, \ldots c_0)$ where a is 0 for the left child and 1 for the right child. The bits c_{k-1} to c_0 are inherited by the children from the parent. The determination of whether a given node is a descendent or an ancestor of another node is based on a logical bitwise comparison of the least significant bits of node addresses. With this addressing scheme any generation of descendants or ancestors of a node can be easily determined.

- *The traversal path downward is completely specified by the destination node's address.* As the tree is traversed downwards, the high-order bit of a parent node's address is replaced with a 0 if it is passed to the left child or with a 1 if to the right. Thus the tree traversal from a node at level k $(0, \ldots 0, 1, c_{k-1}, \ldots c_0)$ to a destination node in its subtree at level n $(0, \ldots 0, 1, d_{n-1}, \ldots d_i, \ldots d_k, c_{k-1}, \ldots c_0)$ is specified by the bits d_{n-1} to d_k. Specifically, at each level i, $k \leq i \leq n-1$, the decision to go left or right is determined by bit d_i of the destination node's address, where $d_i = 0$ indicates left and a $d_i = 1$ right.

The basic concept used in binary tree routing is that of the subtree. A message is first routed to the destination subtree and then down the subtree to its destination node. To communicate between two different subtrees it is necessary to go through a common ancestor node. Therefore, any message not destined for a node in the current subtree must be sent up the tree. Once the correct subtree is reached, the algorithm must determine how to send the message through the subtree to its destination. Assuming $(0, \ldots 0, 1, c_{k-1}, \ldots c_0)$ and $(0, \ldots 0, 1, d_{n-1}, \ldots d_0)$ as current and destination nodes, respectively, a distributed routing algorithm is shown in program-like fashion below, where the assignments to the right represent potential port assignments.

```
CASE_OF_k
    (k = n): IF(c_{k-1}, ... c_0 = d_{k-1}, ... d_0)
                THEN: deliver message to current node                    Assignment 1
                ELSE: send the message to the parent of the current node Assignment 2
    (k > n): send the message to the parent of the current node          Assignment 3
    (k < n): IF (c_{k-1}, ... c_0 = d_{k-1}, ... d_0)
                THEN: IF(d_k = 0)
                        THEN: send the message to current node's left subtree  Assignment 4
                        ELSE: send the message to current node's right subtree Assignment 5
                ELSE: send the message to the parent of the current node      Assignment 6
END_CASE
```

The bit-pattern entries for this binary tree routing algorithm are given in Figure 5. The top priority in the selection of ports is given to the present node (assignment 1). Selection of this port requires $(k = n)$ and $(d_{n-1}, \ldots d_0 = c_{k-1}, \ldots c_0)$. Assignment 2 requires the destination node be on the same level in the tree hierarchy; thus $(k = n)$ and the destination address word is led by 0's at least up to d_{k+1} with $(d_k = 1)$. 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 up to d_k. In both cases the remaining bit positions are not important to the determination of the output port. These bit positions are set to the "don't care" condition. Since these two assignments 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 5.

Assignments 4 and 5 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.

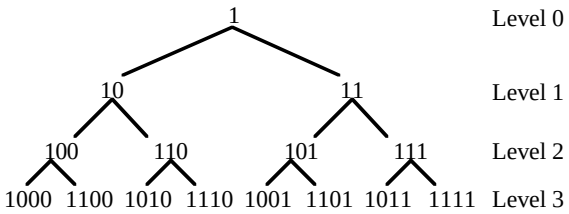

Fig. 4. Node addressing in a binary tree.

The same set of bit-patterns can be used to route messages in any m-ary tree. In the general case, each bit c_i in Figure 5 is replaced by $\lceil log_2(m) \rceil$, bits. The number and structure of the patterns in Figure 5 would remain the same. The detailed mapping of other tree interconnection networks can be found in [14].

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
	m	k	k-1	0	

Fig. 5. Binary tree bit-patterns.

B. Mapping of an Oblivious Hypercube Routing Algorithm

The hypercube network [12] consists of 2^n nodes organized in a multidimensional structure with n dimensions. In this structure, each node has one connection to every dimension; a node $(c_{n-1}, \ldots c_{i+1}, c_i, c_{i-1}, \ldots c_0)$, where c_i is 1 or 0, has a link to node $(c_{n-1}, \ldots c_{i+1}, \overline{c_i}, c_{i-1}, \ldots c_0)$ for $0 \leq i \leq n-1$. As a result, the degree of each node is n.

Messages are sent across one of the dimensions in which the source and destination addresses differ. This routing algorithm is referred to as the bit correction algorithm because one bit of the current node address is corrected at each intermediate node until the current node is the destination. This method reduces by one the distance between the message and its destination at each node along the route. The minimum route length for a given source-destination pair is given by the Hamming distance between the source and destination node addresses, which is also equal to the number of equally short disjoint paths between the two nodes [4]. Given that an oblivious routing is deterministic, a dimension ordering is used to select from these potential routing alternatives. Without losing generality we have assumed that the routing is performed in order of decreasing dimension; any other oblivious scheme can be implemented. This hypercube routing algorithm can be implemented with the set of bit-patterns given in Figure 6. The first bit-pattern compares the destination address to the current address $(c_{n-1}, \ldots c_{i+1}, c_i, c_{i-1}, \ldots c_0)$ to determine if the message has arrived at its destination. Each remaining pattern compares one bit of the destination address with the inverse of the corresponding bit in the current node's address. A match indicates that the dimension corresponding to that bit position is a potential routing alternative. The priority ensures that the dimension ordering is preserved. Any dimension ordering can be implemented by rearranging the order of the bit-patterns. It can be observed that the number of bit-patterns required is $n+1$, where n is the dimension of the hypercube; thus, the number of entries is $O(degree)$.

1	c_{n-1} c_{n-2} c_{n-3} c_{n-4}	...	c_1	c_0	this node
2	$\overline{c_{n-1}}$ X_{n-2} X_{n-3} X_{n-4}	...	X_1	X_0	link n-1
3	X_{n-1} $\overline{c_{n-2}}$ X_{n-3} X_{n-4}	...	X_1	X_0	link n-2
	⋮				⋮
n	X_{n-1} X_{n-2} X_{n-3} X_{n-4}	...	X_1	$\overline{c_0}$	link 0
	n-1			0	

Fig. 6. Bit-patterns for an oblivious hypercube routing algorithm.

C. Adaptive Hypercube Routing Algorithm

In an adaptive routing the path taken by a message depends on the status of the interconnection network as well as the source and destination nodes. An adaptive routing algorithm can be mapped onto the bit-pattern associative router by considering the interconnection network status as input to the bit- pattern matching unit. The algorithm will then produce the same path for a given source node, destination node, and network status.

We have chosen an adaptive routing algorithm to show how this type of algorithm can be implemented on the proposed approach. Below we present the mapping of the minimal p-cube routing algorithm [5] for the hypercube interconnection network to show one way in which alternative paths for blocked messages can be implemented in the bit-pattern associative router.

The p-cube routing algorithm is based on the turn-model for adaptive routing [5]. Routing algorithms based on the turn model prevent deadlock and livelock by restricting the turns a message can make along the path from source to destination. The p-cube routing algorithm is an example of the negative-first algorithm which prohibits a message from making a turn from a positive direction of travel to a negative direction. This implies a two-phase algorithm in which a message is first adaptively routed in those dimensions requiring a negative direction of travel and then adaptively routed in dimensions that require a positive direction of travel. For the hypercube, the first phase routes the message along any dimension in which $c_i = 1$ and $d_i = 0$ and the second phase routes along any dimension for which $c_i = 0$ and $d_i = 1$.

Figure 7 shows the mapping of the p-cube routing algorithm. The bit-patterns for each node are based on the node address. Without losing generality, we have chosen to show the bit patterns for a node in a six-dimensional hypercube

whose address ($c_5c_4c_3c_2c_1c_0$) is 101100. The input to the matching unit consists of the destination address from message header and the status (blocked or free) of each output port on the current node. In this example the algorithm specifies a negative direction of travel for phase one; each selected bit in an entry for phase one corresponds to a 1 bit in the current node's address. Thus phase one will route the message along the first dimension i for which $c_i = 1$, $d_i = 0$ and channel i is free. Each entry in phase two corresponds to a 0 in the current node's address and a message is routed by the first entry for which $c_i = 0$, $d_i = 1$ and channel i is free. Note that the bit positions in the current node's address which can be corrected in phase one are required to match the destination node address in phase two. This ensures that all phase one routing is completed before phase two begins.

If all the potential output channels are blocked the last entry indicates that no assignment can be made at this time. In this case an attempt to assign a port might be done at a later time. Another alternative is to add entries to the bit-pattern approach to implement oblivious routing. These entries are added in place of the last entry in Figure 7 and consist of the first six entries with the channel status bits set to "don't care".

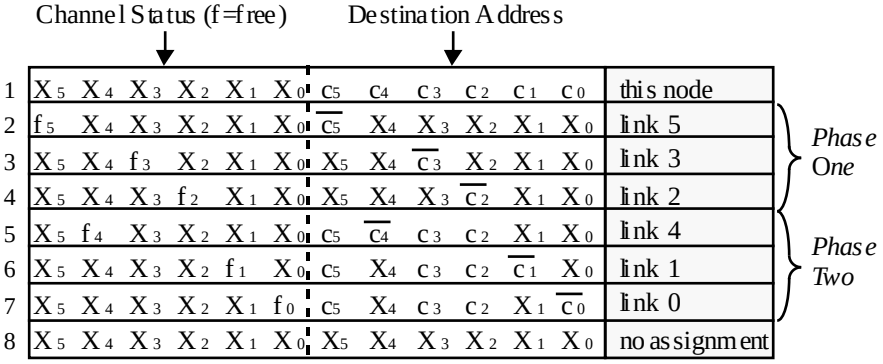

Fig. 7. Bit patterns for an adaptive hypercube routing algorithm.

D. Mapping of Other Interconnection Networks

Other interconnection network routing algorithms can be implemented on the bit-pattern associative router. We have reported the mapping of oblivious routing algorithms for 37 interconnection networks in [14][15]. These networks fall into four major categories: tree, cube, array, and multistage interconnection networks. Routing in most tree interconnection networks is based on an extension of a m-ary tree routing reported in this paper; thus, the mappings for these networks resemble the binary tree mapping with provisions made for the additional links found in these networks. In general, an oblivious routing in an array or cube network is based on a dimension ordered routing. Mapping these networks is similar to mapping the e-cube routing algorithm for the hypercube; each dimension is programmed in the order specified by the algorithm. Most multistage network routing algorithms map directly onto the proposed approach. For example, routing in a delta class network is digit controlled; the switch at stage i is directly controlled by the ith base-b digit of the destination address.

IV. 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 [9][11], 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 [11]. 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 [11]. 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 [1], 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 $O(N)$ size, where N is the number of nodes in the system [2]. An example of a working router utilizing look up tables is Hnet [13]. To reduce the size of the look up tables, van Leeuwen and Tan [16] 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. 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 multi-processor Transputer-based systems [7]. Potential drawbacks of the interval routing approach include: minimum distance path is only possible for a subset of interconnection networks [2], inserting a new node in the network usually results in a new labeling of the nodes and new assignment of ranges [16], and single assignment from a number of potential paths is obtained [2].

Table I 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 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. Interconnection networks with highly irregular structures or node addressing schemes may require a large number of entries in both the interval and the bit-pattern associative routers. Complex adaptive routing algorithms tend to be mapped onto a large number of entries.

TABLE I
COMPARISON OF THE FLEXIBLE ROUTING SCHEMES.

Feature	Bit-Pattern Associative	Interval	Lookup Table
Routing algorithm	implicit	explicit	explicit
Stored data	bit-patterns	addresses	addresses
Number of entries	$O(degree)$	$O(degree)$	$O(networksize)$
Expandability requirements	minor	relabel all node addresses & ranges	reprogram & expand lookup tables
Assignment delay	1 comparison and 1 read	1 comparison and 1 read	1 decode and 1 read
Programmability	yes	yes	yes
Alternative paths	yes	no	no

V. CONCLUDING REMARKS

In this paper we have presented a novel routing algorithm decoding approach for a wide range of interconnection network topologies. To show the capabilities of the proposed approach, we have studied implicit routing algorithms for 37 interconnection network topologies in four major categories; these are: tree, cube, array, and multistage. In this paper we include a full description of three interconnection network routing algorithms. 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.* Implicit algorithms were mapped onto bit-patterns for the tree, cube, array, and multistage structures. The output port is determined on the basis of the current node address, destination node address, and the interconnection network structure. In the case of adaptive routing, the interconnection network status is included to assign an output port.
- *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 was $O(d)$, 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 all the studied interconnection networks is 1.35. Thus, the proposed approach tends to map routing algorithms into the smallest number of entries.
- *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 were 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.

The bit-pattern associative router provides a flexible and programmable approach to execute interconnection network routing algorithms with a high-speed port assignment comparable to topology specific routers. An implementation of this router is reported in [3].

Acknowledgment

The authors are grateful to the anonymous referees of this paper and Professor Lionel M. Ni for their helpful comments and suggestions.

References

[1] 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.

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

[3] J. G. Delgado-Frias, R. Sze, D. Summerville, V. Aikens, "A VLSI CAM-Based Router for Multiprocessor Organizations," *Proceedings Fourth Great Lakes Symposium on VLSI*, Notre Dame, IN, pp. 124-129, March 1994.

[4] 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.

[5] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," *Journal of the Association for Computing Machinery*, vol. 41, no. 5, pp. 874-902, Sept. 1994.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] J. Park, S. Vassiliadis, and J. G. Delgado-Frias, "Flexible Oblivious Router Architecture," *IBM Technical Report TR01.C749*, IBM, Endicott, NY 13760, Sept. 1993. Also to appear in *IBM Journal of Research and Development*, July 1995.

[12] C. L. Seitz, "The Cosmic Cube," *Comm. of the ACM*, vol. 28, no. 1, pp. 22-33, Jan. 1985.

[13] D. Smitley, F. Hady and D. Burns, "Hnet: A High-performance Network Evaluation Testbed," *Proceedings of the 1992 International Conference on Parallel Processing*, vol. 1, pp. 276-279, Aug. 1992.

[14] 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.

[15] 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, June 1995.

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