Optimally Adaptive, Minimum-Distance, Circuit-Switched Routing in Hypercubes

AUSIF MAHMOOD
University of Bridgeport
and
DONALD J. LYNCH and ROGER B. SHAFFER
Washington State University at Tricities

In circuit-switched routing, the path between a source and its destination is established by incrementally reserving all required links before the data transmission can begin. If the routing algorithm is not carefully designed, deadlocks can occur in reserving these links. Deadlock-free algorithms based on dimension-ordered routing, such as the *E-cube*, exist. However, *E-cube* does not provide any flexibility in choosing a path from a source to its destination and can thus result in long latencies under heavy or uneven traffic. Adaptive, minimum-distance routing algorithms, such as the *Turn Model* and the *UP Preference* algorithms, have previously been reported. In this article, we present a new class of adaptive, provably deadlock-free, minimum-distance routing algorithms. We prove that the algorithms developed here are optimally adaptive in the sense that any further flexibility in communication will result in deadlock. We show that the *Turn Model* is actually a member of our new class of algorithms that does not perform as well as other algorithms within the new class. It creates artificial hotspots in routing the traffic and allows fewer total paths. We present an analytical comparison of the flexibility and balance in routing provided by various algorithms and a comparison based on uniform and nonuniform traffic simulations. The *Extended-UP Preference* algorithm developed in this article is shown to have improved performance with respect to existing algorithms. The methodology and the algorithms developed here can be used to develop routing for other schemes such as wormhole routing, and for other recursively defined networks such as k-ary n-cubes.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Achitectures—*interconnection architectures*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*network communications*; *network topology*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Adaptive routing, circuit switched, hypercube, interconnection networks, k-ary n-cube, parallel processing, routing algorithms

This research was funded in part by a grant from the National Science Foundation Center for Design of Analog and Digital Integrated Circuits under grant CDADIC 92-4.
An initial version of this article was presented at the 1995 International Parallel Processing Conference.
Authors' addresses: A. Mahmood, Computer Engineering, University of Bridgeport, Bridgeport, CT 06601; D. J. Lynch and R. B. Shaffer, Washington State University at Tricities, Richland, WA 99352.

1. INTRODUCTION

Hypercubes, i.e., binary n-cube interconnection systems, have been extensively studied in recent years [Chiu et al. 1994; Choi and Somani 1993; Gaughan and Yalamanchili 1993; Kim and Das 1994; Ohring and Das 1996; Seitz 1985]. An n-dimensional (n-D) hypercube network has $N = 2^n$ nodes with each node assigned an n-bit address $(x_{n-1}, x_{n-2}, \ldots, x_0)$ between 0 and $N - 1$. Two nodes are considered neighbors if their addresses differ only in one bit. Each node is thus connected to exactly n neighbors through n pairs of unidirectional busses or links. An n-D hypercube can be constructed recursively from two $(n - 1)$-D hypercubes.

Hypercubes have a regular structure with high connectivity and low diameter, allowing flexible and efficient communication between any two processors in the system, and the ability to tolerate faults in the system. These advantages have led to numerous hypercube multiprocessor implementations, e.g., Intel iPSC-2, iPSC-860, nCUBE, nCUBE-10, etc. nCUBE corporation is actively developing newer hypercube-based designs. Recently, product networks based on a Cartesian product of one network to another are being investigated, e.g., a folded Peterson cube [Ohring and Das 1996] involves Cartesian product of a folded Peterson graph and the hypercube. The routing strategy in such a product network involving a hypercube requires the use of an existing hypercube routing algorithm. Thus, development of improved routing schemes in the hypercube interconnection system has important implications for such systems.

There are many different methods of message communication in interconnection networks such as a hypercube, e.g., packet switching, circuit switching, virtual cut-through, wormhole switching, pipelined circuit switching, etc. An explanation of the different switching techniques can be found in Gaughan and Yalamanchili [1993]. In this article, we focus only on the circuit-switched routing. However, as we explain later, the algorithms developed here are easily modifiable to work with other switching techniques, including wormhole routing.

The circuit-switched scheme completely reserves the path from a source s to its destination d before message transmission begins. The links in the path are released only after the message transmission is completed. In reserving links in an s-d path, a preferred approach (e.g., used in Intel's iPSC-860) is to use the *reserve-and-hold* policy. The reserve-and-hold scheme waits at an intermediate node until the partial reserved path can be advanced toward the destination node by reserving an appropriate outgoing link. Thus, it is possible that it can lead to a deadlock state if there are cyclic dependencies among the sources reserving the partial paths toward their destinations. A simple example of deadlock for a 2-D hypercube is shown in Figure 1. Suppose that each of the messages at a processor node shown in Figure 1(a) simultaneously tries to reserve links from source to destination clockwise around the hypercube and that each succeeds in reserving the first link (indicated by darker lines in Figure 1(b)). The hypercube in Figure 1(b) is in a deadlocked state, since no message can

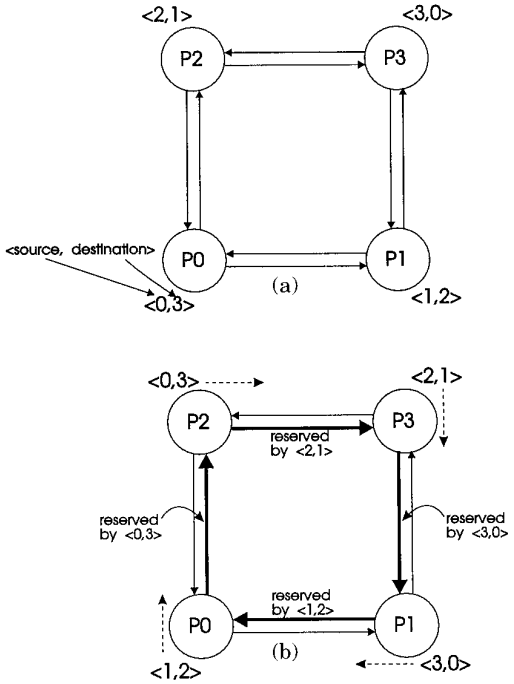

Fig. 1. (a) message requests in a 2D hypercube; (b) a deadlock in reserving links for the messages in (a).

completely reserve the path to its destination in order to complete its transmission and release the links it has reserved.

Various algorithms have been designed to achieve deadlock-free, circuit-switched routing. These algorithms can be classified as either oblivious (nonadaptive) or adaptive. An adaptive algorithm provides many possible routes between a source and its destination. Adaptive algorithms can be *minimal* or *nonminimal*. Minimal algorithms allow only minimum-distance paths between a source and a destination node, while nonminimal algorithms allow paths of any length, possibly even of infinite length. Minimum-distance routing algorithms generally result in a simpler router design and thus have been used in most existing hypercube implementations. In this article, we focus on minimal, adaptive, circuit-switched routing algorithms using the reserve-and-hold strategy.

2. EXISTING HYPERCUBE ROUTING ALGORITHMS

One popular, deadlock-free, circuit-switched routing method used in hypercubes is known as the *E-cube* algorithm [Sullivan and Brashkow 1977]. It reserves the links in a strictly increasing (or decreasing) order of the dimensions in which the binary representations of the source and destination nodes differ. This algorithm is minimal and oblivious. The *E-cube* routing algorithm has the advantage of simplicity. However, since there is

only one prescribed path for a source-destination (s-d) pair, much of the hypercube's flexibility is not utilized.

Some hypercube routing algorithms have been developed for wormhole routing [Dally and Seitz 1987; Duato 1994; Gravano et al. 1992; Lin and Lin 1994a; Yang and Tsai 1993]. It has been shown that any circuit-switched routing algorithm is easily modifiable to work for the wormhole routing [Boppana et al. 1992; Gaughan and Yalamanchili 1993]. In wormhole routing, a message is broken into small fixed-size blocks called flits. Each node has a buffer space of only one flit per incoming link. The flits are pipelined through the network as links are incrementally acquired toward the destination. A link is released after the last flit in the message goes through it. Wormhole routing often uses virtual channels which are time multiplexed over a physical channel. Dally and Seitz [1987] introduced the concept of a channel dependency graph and showed that an algorithm for wormhole routing can be deadlock free if the dependency graph is acyclic. The channel dependency graph depicts the dependencies between the channels (links) of the interconnection system as implied by the allowed routing. Dally and Seitz also introduced a methodology to develop deadlock-free routing algorithms. In their scheme, physical channels corresponding to cycles are split into a group of virtual channels; the virtual channels are ordered; and routing is restricted to visit channels in decreasing order to eliminate cycles in the channel dependency graph.

A minimum-distance, adaptive wormhole routing algorithm reported by Gravano et al. [1992] uses four virtual channels per physical link to achieve deadlock-free routing. Another algorithm reported by Lin and Lin [1994a] uses $(n + 1)/2$ virtual channels per physical link to achieve minimal, adaptive deadlock-free wormhole routing in an n-D hypercube. Recently, an improved approach to designing adaptive wormhole routing has been presented by Duato [1994]. The general drawback of the virtual-channel approach is that the addition of virtual channels adds extra overhead in message communication and setup. In addition, it has been shown that this mechanism perturbs the network's balance, resulting in nonuniform traffic flow through the network and degraded performance [Bolding 1992; Chien 1993].

An adaptive, minimal routing algorithm has been presented by Konstantinidou [1990]. This algorithm has the drawback that it causes the node whose address is all ones to become a hotspot, creating a critical problem in system performance [Chiu et al. 1994]. Another algorithm, presented by Li [1994], allows $k + 1$ simultaneous disjoint paths from a source to a destination node that is a Hamming distance k from the source. A number of nonminimal, wormhole routing algorithms have been proposed by Bolding et al. [1994], Fulgham [1993], Kim and Shin [1993], and Ngai and Seitz [1989]. A nonminimal algorithm has the potential to cause infinite-length paths known as livelocks, and thus some mechanism is needed to avoid these. Although it has been shown that the chaos router reduces network

Table I. An Example Showing Allowed Paths in the *Turn Model* and *UP Preference* Algorithms

PATH (by node order)	PATH (by transition order)	Allowed by *Turn Model*	Allowed by *UP Preference*
3→2→0→4	$D_0 \rightarrow D_1 \rightarrow U_2$	Yes	Yes
3→2→6→4	$D_0 \rightarrow U_2 \rightarrow D_1$	No	Yes
3→7→6→4	$U_2 \rightarrow D_0 \rightarrow D_1$	No	Yes
3→1→0→4	$D_1 \rightarrow D_0 \rightarrow U_2$	Yes	No
3→1→5→4	$D_1 \rightarrow U_2 \rightarrow D_0$	No	No
3→7→5→4	$U_2 \rightarrow D_1 \rightarrow D_0$	No	No

congestion [Bolding et al. 1994; Fulgham 1993], the livelock prevention implementation has been difficult for circuit-switched applications [Chiu et al. 1994]. Thus in this work we focus only on minimal algorithms.

Of the known existing, minimal, circuit-switched routing algorithms, the *Turn Model* [Glass and Ni 1994] and the *UP* (or *DOWN*) *Preference* [Chiu et al. 1994] algorithms have previously shown the greatest flexibility in providing deadlock-free paths from a source to its destination. The *Turn Model* algorithm [Glass and Ni 1994] is developed by analyzing the directions in which packets can turn in a network and the cycles they form. By removing the turns that can cause deadlocks (cycles in the associated channel dependency graph), a deadlock-free routing is obtained. The *Turn Model* and *UP Preference* algorithms are defined in terms of up and down transitions. If bit x_i in dimension i in the source address changes from 0 to 1 in order to proceed toward its destination, it is considered an up transition U_i. Similarly, a down transition D_i changes x_i from 1 to 0. The *Turn Model* algorithm operates in two phases:

(1) PHASE 1 (down phase): All down transitions required between a source and its destination can be traversed in any order.
(2) PHASE 2 (up phase): The up transitions are completed in any order.

The *UP Preference* algorithm [Chiu et al. 1994] specifies that any up transition U_i can be made at any point in the order of required transitions, but a down transition D_i can be made only after all lower-dimension transitions have been completed. Table I shows all possible paths from node 3 to node 4 and whether or not each path is allowed by the *UP Preference* and the *TURN Model* algorithms. The last three paths in Table I are not allowed by the *UP Preference* algorithm because D_1 occurs before the D_0 transition.

Chiu et al. [1994] have shown that the *UP Preference* algorithm is deadlock free because it yields acyclic channel dependency graphs. Note that the *UP Preference* scheme is not optimal in terms of the flexibility in allowed paths. For example, in a 3D hypercube, the path 3→1→5→4 is not allowed by the *UP Preference* rule. However, if this is added to the set of allowed paths, the resulting channel dependency graph is still acyclic. As the next section shows, a new class of routing algorithms developed in this

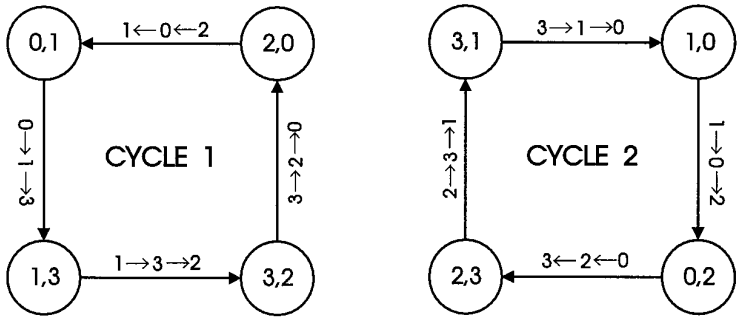

Fig. 2. Channel dependency graph for the 2D hypercube with unrestricted routing.

article improves Chiu et al.'s algorithms by providing an increased number of allowed paths, while still remaining deadlock free. In fact, the set of paths allowed by one of the new algorithms is a superset of those allowed by the *UP Preference* rules. Thus we designate this new algorithm as the *Extended-UP Preference* algorithm. Not all members of our new class of algorithms perform equally well. In particular, the *Turn Model* algorithm is a member of this new class which is shown to perform poorly, and it allows fewer *s-d* paths in a hypercube than, for example, the *Extended-Up Preference* algorithm.

3. EXTENDED-UP PREFERENCE ALGORITHM

The *Extended-UP Preference* algorithm is based on extending the deadlock-free routing in a 2D hypercube, as defined by the *UP Preference* rule, to general n-D hypercubes but relaxing the global restriction that down transitions cannot precede lower-dimension transitions. That is, the *Extended-UP Preference* algorithm uniformly applies the *UP Preference* rule to each of the 2D subcubes traversed in an *s-d* path within a larger hypercube. This results in a deadlock-free routing with much more flexibility than the original *UP Preference* algorithm. The reasons for treating the overall routing in terms of 2D subcube traversals are that a 2D cube is the smallest network that contains physical loops and that the cycles in the channel dependency graph are related to these physical loops. After eliminating cycles in a 2D hypercube by removing the minimum possible number of edges in the channel dependency graph, we extend the deadlock-free routing scheme to general n-D hypercubes.

The channel dependency graph D defined by Dally and Seitz [1987], for a given interconnection network I and routing function R, is a directed graph, $D = G(C, E)$. The vertices of D are the channels of I. The edges of D are the pairs of channels connected by R as

$$E = \{(c_i, c_j) | R(c_i, n) = c_j \text{ for some } n \in N\}$$

where c_i and c_j are the channels in I; and N is the set of processing nodes in the interconnection network. Figure 2 shows the channel dependency

graph for a 2D hypercube without any restrictions in routing messages from a source to a nonadjacent destination. Each vertex in Figure 2 corresponds to a physical link or channel in the 2D hypercube and is identified by the nodes that it connects. For example, the vertex labeled 1,3 corresponds to the channel from node 1 to node 3. An edge in the graph in Figure 2 exists if that path is allowed by the routing criterion. For example, the edge labeled 1→3→2 indicates that the path 1→3→2 is allowed by the routing algorithm and that it uses the channels 1,3 and 3,2.

Since there are two cycles created in the channel dependency graph shown in Figure 2, *unrestricted* routing can deadlock. In order to remove the potential for deadlocks, at least one edge must be removed from each of the two cycles. If the top edges, labeled 2→0→1 and 3→1→0, are removed, then each of the two cycles in Figure 2 is broken. This results in one possible deadlock-free routing scheme in the 2D hypercube. Incidentally, this particular deadlock-free routing is the same as that allowed by the *UP Preference* rule in a 2D hypercube.

Notice that there are 16 possible deadlock-free routing solutions, depending upon which combination of edges is removed from the two cycles in the channel dependency graph for a 2D hypercube. By uniformly applying one of these deadlock-free routing solutions to all the 2D subcubes in a larger hypercube, a deadlock-free routing algorithm can be obtained. Section 3.1 elaborates on the various members of this new class of algorithms. One member of our new class of algorithms is obtained when the *UP Preference* rule on a 2D hypercube is *extended* by uniformly applying it to all the 2D subcubes in an n-D hypercube. The resulting routing is deadlock free and has greater flexibility than the original *UP Preference* algorithm of Chiu et al. [1994].

It should be noted that Duato's recent theory in developing adaptive wormhole routing [Duato 1994] can allow cyclic dependencies among channels as long as there exists an escape route which is fully connected. In Duato's scheme, usually one virtual channel implements the *E-cube* routing algorithm (escape route), while the other virtual channels use fully adaptive routing. Deadlock freedom is guaranteed by the existence of an acyclic channel dependency graph for the escape route only when considering all direct, indirect, and cross dependencies between the escape and other routes. It is important to note that our article concerns circuit-switched routing without using any virtual channels. Thus, in our case it is required that the entire channel dependency graph be acyclic for deadlock-free operation.

Extended-UP Preference Routing Criterion. The *Extended-UP* (*Ex-UP*) *Preference* rule is equivalent to applying the standard *UP Preference* rules uniformly to all 2D subcubes in a larger hypercube, without any additional restriction. Thus, in the sequence of up and down links (transitions) traversed in the path from a source to its destination within an n-D hypercube, the *UP Preference* rule must be satisfied for any two consecutive transitions in the path. As an example of this rule, the path 3→1→5→4,

which was not allowed by the *UP Preference* rule, is allowed under the *Ex-UP Preference* scheme. This can be seen by the transitions involved in 3→1→5→4, which are $D_1 \rightarrow U_2 \rightarrow D_0$. Each of the consecutive 2D subcubes traversed in this routing do satisfy the *UP Preference* rule, i.e., $D_1 \rightarrow U_2$ and $U_2 \rightarrow D_0$ are both individually allowed under the *UP Preference*, and thus collectively $D_1 \rightarrow U_2 \rightarrow D_0$ (path 3→1→5→4) is legal under the *Ex-UP Preference* rule.

In general, any path created from a sequence of allowed 2D subpaths (by the *UP Preference* rule) is an allowed path in the *Ex-UP Preference* scheme. The only constraints are "local," i.e., within each 2D face. Consequently, the *Ex-UP Preference* algorithm falls within the category of greedy optimization algorithms. A formal description of the *Ex-UP Preference* rule follows from defining the set of up transitions $UT(s, d)$ and the set of down transitions $DT(s, d)$ needed in traversing a path from a source s to a destination d:

$$UT(s, d) = \{i | 0 \leq i \leq (n-1),\ s_i = 0,\ d_i = 1\}$$

$$DT(s, d) = \{i | 0 \leq i \leq (n-1),\ s_i = 1,\ d_i = 0\}$$

For an s-d pair of nodes (X, Y), suppose a path has been established up to an intermediate node I as $(X, \ldots, H, I, \ldots, Y)$. In establishing the path from H to I, if a transition in dimension *prev_dim* is taken, then the following are set:

If $I = X$, then *previous_transition* = UP, *prev_dim* = -1
If an up transition occurred in the link from H to I
  previous_transition = UP else *previous_transition* = DOWN
If (*previous_transition* = UP)
  lock_dimension = -1 else *lock_dimension* = *prev_dim*

The following rules are applied to determine if the next link of dimension greater than *lock_dimension* can be reserved from I:

—A j-dimension link ($j >$ *lock_dimension*) from I can be reserved if $j \in UT(I, Y)$
—A j-dimension link ($j >$ *lock_dimension*) from I can be reserved if $j \in DT(I, Y)$ AND [{ $m | m > j$ AND $m \in UT(I, Y)$} OR
  { m, *lock_dimension* $\leq m < j$ AND $m \notin (UT(I, Y) \cup DT(I, Y))$}]
—A j-dimensional link from I cannot be reserved if $j \notin (UT(I, Y) \cup DT(I, Y))$

Note the added flexibility in *Ex-UP Preference* routing as compared to the *UP Preference* rule; the *Extended* concept does not necessarily restrict a down transition in dimension i if the transitions in dimensions less than i remain, in order to proceed to the destination. Such a down transition can be taken in the *Ex-UP Preference* scheme (but not in *UP Preference*) if an up transition of dimension greater than i is remaining. In order to prevent deadlocks, all transitions whose dimensions are less than i are considered to be locked once a down transition in dimension i is taken. When an up

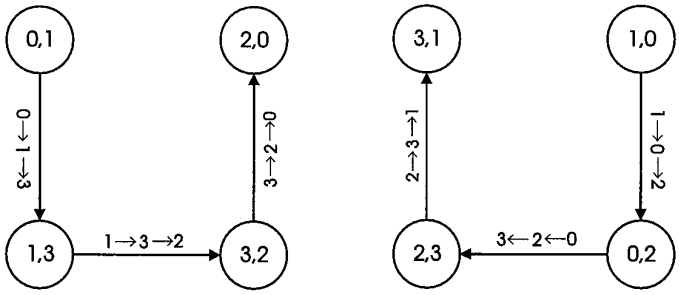

Fig. 3. Channel dependency graph for the 2D hypercube under *Ex-UP Preference* rule.

transition of dimension higher than the *lock_dimension* occurs, it unlocks, i.e., allows all remaining transitions in any dimension. Theorem 3.1 formally demonstrates that the *Ex-UP Preference* scheme results in a deadlock-free routing.

THEOREM 3.1. *If an algorithm based on the Ex-UP Preference criteria is used to route messages in an n-D hypercube* ($n \geq 2$)*, no deadlocks in routing can occur.*

PROOF. We use induction to prove this theorem. The associated channel dependency graphs are derived at each step to show that they are acyclic.

Base Case: $n = 2$. For the 2D hypercube, the *Ex-UP Preference* rule disallows the paths 2→0→1 and 3→1→0. The resulting channel dependency graph is shown in Figure 3. Since it is acyclic, the *Ex-UP Preference* routing in a 2D hypercube is deadlock free.

Inductive Step. We assume that the theorem holds for hypercubes of dimension 2, 3, . . . , n, and we show that it holds for a hypercube of dimension $n + 1$.

Consider that the hypercube of dimension $n + 1$ is partitioned along dimension n into two subcubes, N_0 and N_1, each of dimension n. All processor nodes in N_0 have addresses with a zero in the nth-dimension bit (i.e., $(n + 1)$th bit). Similarly, the processor nodes in N_1 have addresses with a one in the most significant bit. Let C_0^n and C_1^n be the channel dependency graphs for N_0 and N_1, respectively, corresponding to the *Ex-UP Preference* rule. By the induction hypothesis, C_0^n and C_1^n are acyclic.

When N_0 and N_1 are connected to form the $(n + 1)$-D hypercube, there are 2^{n+1} additional links (channels) formed between the two n-D subcubes. Half of these, corresponding to the up links from N_0 to N_1, are denoted as the set

$$UL_{n+1} = \{(0, 2^n), (1, 2^n + 1), \ldots, (2^n - 1, 2^{n+1} - 1)\}.$$

The other half corresponds to the down links from N_1 to N_0 forming the set of links

$$DL_{n+1} = \{(2^n, 0), (2^n + 1, 1), \ldots, (2^{n+1} - 1, 2^n - 1)\}.$$

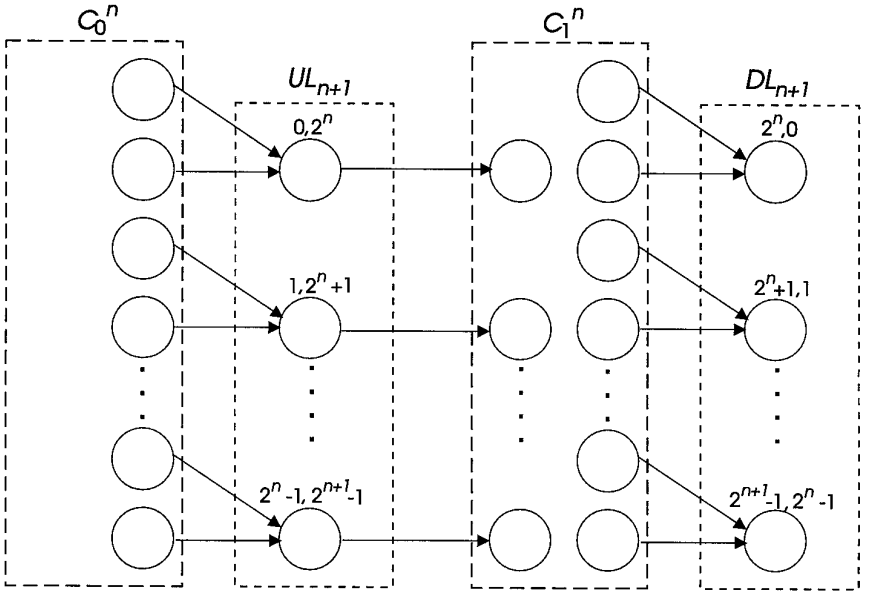

Fig. 4. Channel dependency graph for the $(n + 1)$-D hypercube under *Ex-UP Preference* rule.

The channel dependency graph for the $(n + 1)$-D hypercube based on the *Ex-UP Preference* routing scheme is shown in Figure 4, where channel dependencies internal to C_0^n and C_1^n are omitted. The links (channels) in UL_{n+1} can appear as intermediate nodes in the channel dependency graph for a path connecting a processor node in N_0 to a processor node in N_1. However, since a path beginning in N_1 cannot continue as a path in N_0 under the *Ex-UP Preference* rule, the channels in DL_{n+1} cannot appear as intermediate nodes in the channel dependency graph for a path connecting a processor node in N_1 to a processor node in N_0. This condition results from the fact that a down transition in dimension n (bit position $(n + 1)$) can only be the last transition in the *Ex-UP Preference* rule, as otherwise it will lock all lower dimensions, and there is no higher up transition left to unlock these lower dimensions. Thus, the links in DL_{n+1} are not dependent on any other links, as shown in Figure 4. Therefore the channel dependency graph for the $(n + 1)$-D hypercube is acyclic and deadlock-free. □

3.1 The Class of *Extended* Algorithms

Since the *Extended* concept applies a deadlock-free solution obtained for a 2D hypercube to all the 2D subcubes traversed in an s-d path in a larger hypercube, many deadlock-free routing algorithms can be devised. The unrestricted routing in a 2D hypercube results in two independent cycles in the channel dependency graph, as shown in Figure 2. A deadlock-free routing algorithm in a 2D hypercube must disallow these cycles by removal of one edge from each. Thus, there are 16 possible deadlock-free routing schemes in a 2D hypercube.

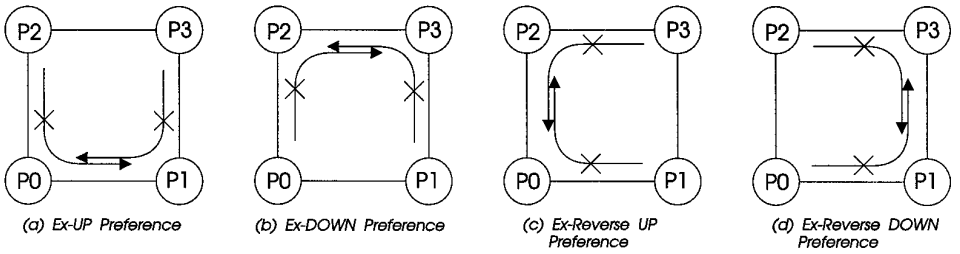

Fig. 5. Disallowed paths in the *Extended-UP/DOWN Preference* group.

As shown earlier, the removal of the top edges in the two cycles in Figure 2 corresponds to disallowing the paths 2→0→1 and 3→1→0 (shown graphically in Figure 5(a)) and is equivalent to the *UP Preference* algorithm of Chiu et al. [1994] in a 2D hypercube. If the bottom two edges (1→3→2 and 0→2→3) in Figure 2 are removed, the *DOWN Preference* solution results. Similarly, the removal of the right and left edges in Figure 2 results in *Reverse UP Preference* (disallowing paths 3→2→0 and 1→0→2) and *Reverse DOWN Preference* (disallowing paths 0→1→3 and 2→3→1) algorithms, respectively. A *Reverse UP Preference* algorithm allows up transitions in any order, but a down transition can be taken only if all the higher transitions have been completed. Similarly, the *Reverse DOWN Preference* algorithm allows an up transition only if all higher-dimension transitions have been completed. The disallowed paths in these four types of algorithms are graphically shown in Figure 5. All other *s-d* paths in Figure 5 are allowed.

Figure 5 demonstrates that all the four different algorithms are closely related, since they have the same pattern of disallowed paths. Further, by processor relabeling, one case can be obtained from the other. For example, if the processor address bits are all inverted in the *UP Preference* scheme, the *DOWN Preference* scheme results. Similarly, the *DOWN Preference* and the *Reverse DOWN Preference* schemes are related by a transposition of the processor address bits. Because of the close relationship of these four 2D hypercube routing solutions, their performance when extended in a uniform manner to a larger hypercube will also be similar to each other. They are therefore considered to be equivalent; only the *Extended-UP Preference* case will be studied in later sections.

Following similar reasoning, the remaining 12 deadlock-free solutions in the 2D hypercube are also divided in three groups with four solutions in each group.

Extended Hotspot Group. If the bottom edge in cycle 1 and the left edge in cycle 2 of Figure 2 are removed by disallowing paths 1→3→2 and 2→3→1, respectively, a solution from the hotspot group results, as shown in Figure 6. It is termed hotspot because the communication between processor 1 and processor 2 is forced to go through processor 0. Under heavy traffic, this will cause a larger message flow through node 0, resulting in an artificial hotspot. For this reason, the solutions from this

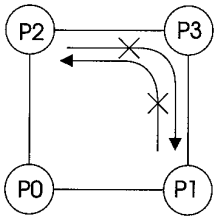

Fig. 6. Disallowed paths in the *Extended* hotspot group.

group are not good candidates for use in routing implementations. This solution corresponds exactly to the *Turn Model* algorithm applied to a hypercube (i.e., *P-cube* "negative-first" algorithm of Glass and Ni [1994]) and performs poorly under traffic simulations because of its hotspot properties, as will be seen in Sections 4 and 5.

Extended Isolation Group. If the top edge in cycle 1 and the left edge in cycle 2 of Figure 2 are removed by disallowing paths 2→0→1 and 2→3→1, respectively, a solution from the isolation group results, as shown in Figure 7. In this scheme processor 2 cannot pass any messages to processor 1, and thus the solutions from this group are not useful in deadlock-free routing design.

Extended Inverse UP/DOWN Preference Group. If the left edge in cycle 1 and the right edge in cycle 2 of Figure 2 are removed by disallowing paths 0→1→3 and 1→0→2, respectively, a solution from the *Inverse UP/DOWN Preference* group results, as shown in Figure 8. This scheme is similar to the *Extended-UP/DOWN Preference* group, and thus its performance should also be similar.

3.2 Optimality of *Extended* Routing Algorithms

We show that the *Extended* class of routing algorithms results in an optimal flexibility in the number of allowed paths in the sense that the addition of any new path to the allowed set of paths by an extended algorithm will result in deadlocks. To prove their optimality, we restrict ourselves to the class of *coherent* routing functions as defined by Duato [1994]. A routing function R for a given network is coherent iff, for every path P that can be established by R, all subpaths of P are also paths of R. All well-defined minimal-distance routing algorithms on the hypercubes fall in this category.

THEOREM 3.2.1. *For an n-D hypercube with* $n \geq 2$, *the* Extended *algorithms are optimal in terms of the number of paths allowed while remaining deadlock free.*

PROOF.

Case 1: 2D Hypercube. All *s-d* paths of length 1 (single link to neighbors) are allowed in a 2D hypercube, since they cannot cause a deadlock. For the paths of length 2, the unrestricted routing causes two independent

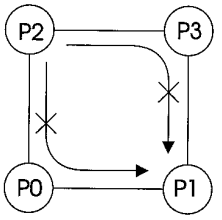

Fig. 7. Disallowed paths in the *Extended* isolation group.

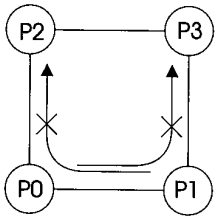

Fig. 8. Disallowed paths in the *Extended Inverse UP/DOWN Preference* group.

cycles in the channel dependency graph as was shown in Figure 2. The *Extended* rule removes only one edge from each of the two cycles to make the routing deadlock free (Figure 3). Thus the algorithm provides optimal flexibility in routing in a 2D hypercube.

Case 2: n-D Hypercube ($n > 2$). All paths of length-l links ($l > 2$) are composed of a sequence of overlapping two-link paths. Allowing a length-l path ($l > 2$) implies that all of the two-link paths it contains are also allowed. A length-l path is disallowed in an *Extended* algorithm if any two-link paths it contains are disallowed. Thus if a disallowed length-l path ($l > 2$) is added to the set of allowed paths, then this path contains at least one disallowed two-link path because the overall path is coherent. This will cause the routing to deadlock, since all of the 2D subcubes are already optimal in terms of allowed two-link paths. □

For example, in a 3D hypercube, the path 3→7→5→4 is disallowed in the *Ex-UP Preference* rule. Allowing the path 3→7→5→4 will imply that the two-link path 7→5→4 is also allowed, but that will cause a deadlock in the 2D subcube where the most significant bit is always a one (the path 7→5→4 is disallowed by the *UP Preference* rule in this 2D subcube). Further, it has also been proved recently by Lin and Lin [1994b] that the *Turn Model* is optimal in the sense that it restricts the minimum number of turns to avoid deadlock. We have shown earlier that the *Turn Model* is one of the members of our new *Extended* class of algorithms. It is not useful in practice, since it creates bottlenecks in routing traffic. Of the class of extended algorithms, the algorithms belonging to the *Extended UP/DOWN Preference* group and the *Extended Inverse UP/DOWN Preference* group have the highest flexibility in terms of allowed *s-d* paths. Section 4 develops an exact comparison of the allowed *s-d* path counts among different algorithms.

4. ANALYTICAL COMPARISON OF ROUTING ALGORITHMS

One measure of the adaptivity provided by a minimum-distance routing algorithm is the count of all allowed x-hop s-d paths in an n-D hypercube where $(2 \leq x \leq n)$. The one-hop path count is not interesting in the comparison, since every routing algorithm allows all possible one-hop paths, i.e., a path to the neighboring nodes. For a single quantitative measure of the adaptivity provided by a minimum-distance routing algorithm, we define the flexibility of an algorithm to be

$$Flexibility_{alg} = \frac{\left(\frac{N_{2h-alg}}{N_{2h-Ecube}} + \frac{N_{3h-alg}}{N_{3h-Ecube}} + \ldots + \frac{N_{nh-alg}}{N_{nh-Ecube}} \right)}{n-1} \tag{1}$$

where N_{ih-alg} and $N_{ih-Ecube}$ are the number of allowed paths of length i-hops in a given algorithm and the *E-cube* algorithm, respectively. The flexibility as defined in (1) is a count of all allowed x-hop paths $(2 \leq x \leq n)$ in an algorithm, normalized by the path counts of the *E-cube* algorithm. The flexibility provided by the *E-cube* algorithm according to Eq. (1) is thus 1, indicating that the *E-cube* algorithm provides a single path from a source node to a destination node. A higher flexibility count is desirable for an algorithm, since a greater number of alternate s-d paths would lessen congestion under heavy or uneven traffic loads.

In order to calculate the flexibility provided by an algorithm according to (1), separate counts of distinct allowed x-hop s-d paths must be determined. Note that in unrestricted routing, the number of n-hop s-d paths (where the destination is the complement of the source) in an n-D hypercube is $2^n \cdot n!$. For example, a 2D hypercube has eight two-hop paths, i.e., two paths originating from each of the four nodes. It can be verified that the number of $(n-1)$-hop s-d paths is $(2^n \cdot n!)/1!$; the number of $(n-2)$-hop s-d paths is $(2^n \cdot n!)/2!$; and so on. In general, the x-hop s-d path count in unrestricted routing is given by Eq. (2):

$$N^{n\text{-D}}_{xh-unrest} = \frac{2^n \cdot n!}{(n-x)!} \quad \text{where } (2 \leq x \leq n-1) \text{ and } n > 2. \tag{2}$$

The superscripts in Eq. (2) indicate the dimension of the hypercube. Thus the ratio of x-hop path counts in an n-D and $(n-1)$-D hypercube is given by

$$\frac{N^{n\text{-D}}_{xh-unrest}}{N^{(n-1)\text{-D}}_{xh-unrest}} = \frac{2n}{(n-x)}. \tag{3}$$

Similar reasoning shows that for any coherent, restricted, minimum-distance routing algorithm which scales uniformly according to the dimension of the hypercube, the increase in the number of allowed x-hop s-d

paths from $(n - 1)$-D to n-D hypercube is also given by the factor of $2n/(n - x)$. Equation (4) states this and shows how the x-hop path count can be recursively determined for any coherent path algorithm:

$$N_{xh-alg}^{n\text{-D}} = N_{xh-alg}^{(n-1)\text{-D}} \cdot \left(\frac{2n}{n - x} \right) \tag{4}$$

For example, the number of distinct two-hop paths in unrestricted routing in a 2D hypercube is 8 (see Figure 2). In a 3D hypercube, since there are six 2D subcubes in it, this number increases by a factor of 6, as indicated by Eq. (4). As another example, the three-hop paths allowed by the *Turn Model* in a 3D hypercube are 24, whereas in a 4D hypercube it is 24×8, as there are eight 3D subcubes in a 4D hypercube. The term $2n/(n - x)$, when $n = x + 1$, is essentially the number of $(n - 1)$-D subcubes in an n-D hypercube. When $n > x + 1$, the relative increase in the number of x-hop paths in an n-D hypercube as compared to the $(n - 1)$-D hypercube is less, due to duplication or sharing of some of the smaller subcubes in the larger hypercube.

Returning to Eq. (4), we see that the starting point for building the recursion is the count of all allowed two-hop s-d paths in a 2D hypercube. Since Eq. (4) is valid only up to $(n - 1)$-hop counts, expressions for n-hop s-d path counts allowed by various algorithms are needed to compute their flexibility. An n-hop s-d path in a minimum-distance routing algorithm traverses all dimensions in linking a source to its destination. Thus, for each of the 2^n nodes in an n-D hypercube, each source node has only one destination node that is n hops away. The n-hop path count is then the aggregate of all the paths allowed by an algorithm in establishing a link from each node in the hypercube to its complement node. The *E-cube* algorithm, being nonadaptive, allows only one such s-d path per node. Thus there is a total of 2^n n-hop s-d paths (because there are 2^n nodes) in an n-D hypercube denoted by $N_{nh-Ecube}^{n\text{-D}}$:

$$N_{nh-Ecube}^{n\text{-D}} = 2^n \tag{5}$$

The total number of allowed n-hop s-d paths in the *UP Preference* and the *Turn Model* is similar and given by Eq. (6).

$$N_{nh-UP-Preference}^{n\text{-D}} = N_{nh-Turn}^{n\text{-D}} = (n + 1)! \tag{6}$$

For example, the number of allowed two-hop paths in a 2D hypercube in the *Turn* and *UP Preference* algorithms is 6 (Figures 3 and 6); three-hop counts in a 3D hypercube is 24; four-hop counts in a 4D hypercube is 120. The increase in n-hop count in an n-D hypercube as compared to the $(n - 1)$-D hypercube is by a factor of half the number of $(n - 1)$-D subcubes in an n-D hypercube. The expression for the n-hop s-d path counts allowed in

the *Ex-UP Preference* algorithm is given recursively by (7),

$$N^{n\text{-D}}_{nh-Ex\text{-}UP\text{-}Pref} = 2 \cdot N^{(n-1)\text{-D}}_{(n-1)h-Ex\text{-}UP\text{-}Pref} + \sum_{i=1}^{n-1} \left[\frac{(n-1)!}{(i-1)!(n-i)!} \right] N^{i\text{-D}}_{ih-Ex\text{-}UP\text{-}Pref} \qquad (7)$$

where $N^{0\text{-D}}_{1h-UP\text{-}Preference} = 1$.

Equation (7) is obtained by induction on n-D hypercubes for $n > 1$. The first term in Eq. (7) represents the allowed n-hop paths that correspond to extensions, by an additional (terminating) hop, of the allowed $(n - 1)$-hop paths in the $(n - 1)$-D hypercubes it contains. The second term in the equation represents the allowed n-hop paths created by inserting an allowed intermediate link between two allowed subpaths of length i and $(n - i - 1)$. It is easy to verify that the n-hop count for the *Ex-UP Preference* algorithm in 2D, 3D, 4D, and 5D hypercubes is 6, 26, 150, and 1082, respectively.

The unrestricted routing flexibility can also be computed to provide an upper ceiling on the flexibility provided by a minimum-distance routing algorithm. The n-hop s-d path count in unrestricted routing (which can deadlock) is given by Eq. (8).

$$N^{n\text{-D}}_{nh-unrest} = (n!)(2^n) \qquad (8)$$

Flexibility defined by Eq. (1) can now be computed for algorithms in this study. The $\log_2$ plot of the flexibility of different algorithms for varying hypercube sizes is shown in Figure 9. The *Ex-UP Preference* algorithm is approximately five times more adaptive for a hypercube size of 10 than the *UP Preference* and the *Turn Model* algorithms and about 100 times more adaptive for a hypercube size of 20. The *E-cube* algorithm, which provides a flexibility of 1 for all hypercube sizes, is not shown in Figure 9. The flexibility provided by unrestricted routing (which does deadlock) is also plotted in Figure 9, for comparison with the deadlock-free algorithms.

Although the *UP Preference*- and the *Turn Model*-based algorithms provide the same count of x-hop s-d paths and thus the same flexibility as defined by Eq. (1), their routing paths are different. As explained in Section 3, the *Turn Model* forces more messages through certain nodes and creates artificial hotspots in routing the traffic. Thus, the flexibility count alone is not a complete indicator of an algorithm's performance. A good algorithm should provide high flexibility and *balance* the traffic through different nodes in a hypercube.

In order to quantitatively measure traffic balance under a routing algorithm, we analyze the Intermediate Node Traffic Count (*INTC*) through each node in the hypercube. Suppose the allowed paths between an s-d pair P-Q are P→X→Y→Q and P→X→W→Q. Because the message has to pass through node X to get from P to Q, the *INTC* for X is 1. However, to proceed further toward Q, the message could either go through Y or W. Hence, the *INTC* for Y and W is 0.5, meaning there is a 50% chance of the message

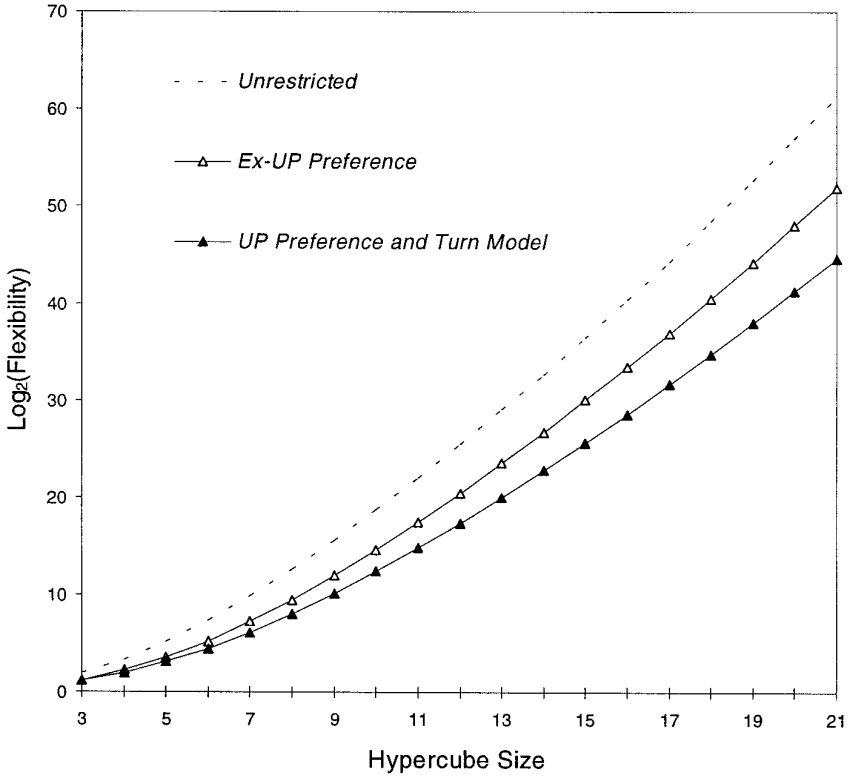

Fig. 9. Comparison of flexibility of different minimum-distance routing algorithms.

going through each if the *s-d* pair is P-Q. Similarly, if the routing algorithm also allowed an additional *s-d* path from P to Q as P→L→M→Q, the *INTC* would become 2/3 for node X and 1/3 for nodes L, M, W, and Y.

By considering all *s-d* pairs for a given size hypercube and enumerating the paths allowed by a routing algorithm, the *INTC* through each node can be computed. The standard deviation of *INTC* for all nodes is then a good measure of the *balance* in routing provided by an algorithm. Overall, the desirable attributes in an adaptive routing algorithm are *high flexibility and low standard deviation of INTC*. Table II lists the flexibility (as given by Eq. (1)) and the standard deviation of *INTC* for hypercube sizes of up to 7. It can be seen from Table II that the *Ex-UP Preference* has the highest flexibility of all the deadlock-free routing algorithms and a relatively low standard deviation of the *INTC* values. Although *E-cube* has a 0 standard deviation, its performance will be limited, since its flexibility is only 1. The mean of *INTC*, also shown in Table II, depends only on the hypercube size and is independent of the routing algorithm.

Despite the fact that the *UP Preference* and the *Turn Model* routing algorithms have identical flexibility counts, the standard deviation of *INTC* for the *Turn Model* is much higher. Figure 10 shows a plot of the *INTC* through all nodes in a 7D hypercube for the three algorithms of interest.

Table II. Flexibility and Standard Deviation of *INTC* for Different Algorithms

Hypercube Size	Mean of *INTC*	*E-Cube* (Flex., S.D.)	*UP Preference* (Flex., S.D.)	*Turn Model* (Flex., S.D.)	*Extended-UP Preference* (Flex., S.D.)	*Unrestricted* (deadlocks) (Flex., S.D.)
3	4	(1, 0)	(2.25, 2.29)	(2.25, 3.51)	(2.38, 2.19)	(4, 0)
4	17	(1, 0)	(4, 7.84)	(4, 12.45)	(4.71, 7.18)	(10.67, 0)
5	49	(1, 0)	(8.62, 23.47)	(8.62, 38.08)	(11.98, 20.57)	(38, 0)
6	129	(1, 0)	(22.65, 64.91)	(22.65, 107.21)	(38.86, 54.52)	(174.40, 0)
7	321	(1, 0)	(71.38, 170.36)	(71.38, 286.18)	(155.54, 137.35)	(985.33, 0)

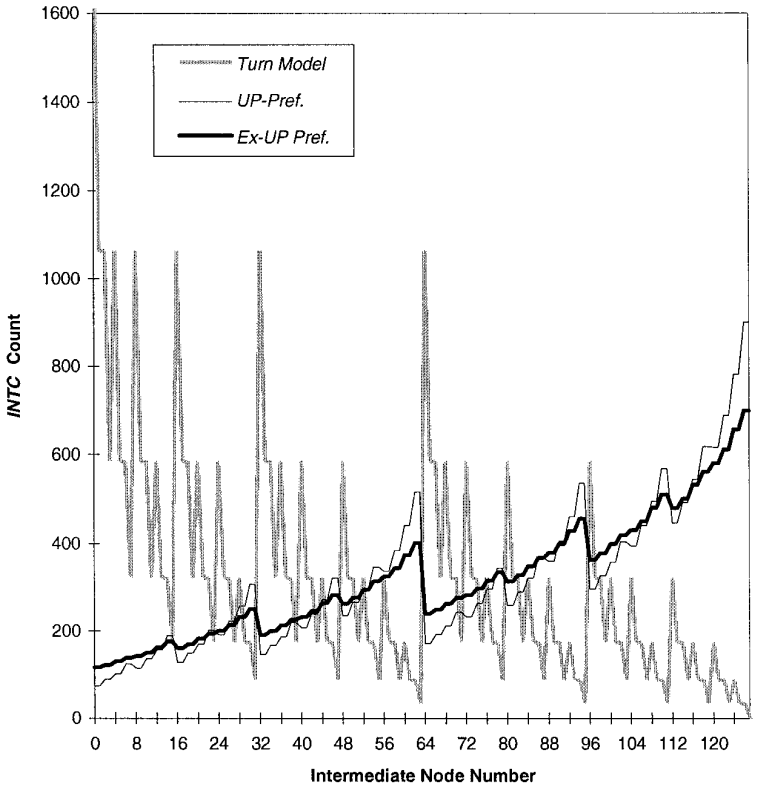

Fig. 10. Comparison of *INTC* for different algorithms in a 7D hypercube (a high value of *INTC* indicates high traffic through a node).

The higher standard deviation of *INTC* in the *Turn Model* indicates its poor balance in routing the traffic. Thus, its performance is expected to be poor in comparison to *UP Preference* and *Ex-UP Preference* algorithms. The traffic simulations in the next section confirm this result.

5. SIMULATION STUDY OF ROUTING ALGORITHMS

A discrete-event simulator was developed to study the performance of various routing algorithms on a hypercube multiprocessor. The simulator

models the hypercube by two unidirectional links between all neighboring nodes, i.e., one outgoing link and one incoming link, to and from each neighbor, respectively. Each node contains a message queue and an associated router controller. If the message queue is not empty, the controller attempts to reserve a nonbusy link according to the routing algorithm being implemented. It is assumed that it takes one time unit to reserve a link if it is available. Once all links between a source and its destination have been established, the message transmission takes place, consuming an amount of time proportional to the size of the message. All of the links involved in a transmission are released when the message transmission is completed.

In our study, the message transmission times can vary between 100 and 900 time units. Both a uniform distribution of message lengths (with a mean of 500) and a Poisson distribution of message lengths (with a mean of 250 time units) are included in the simulation model. The Poisson distribution allows fewer long messages than the uniform distribution. It has been used in related simulation studies of routing schemes, such as in Chiu et al. [1994], and represents many practical applications in a circuit-switched hypercube system. In gathering statistics, 16,000 messages are generated in an 8D hypercube. Of these, the first 3000 and the last 3000 messages are ignored in order to ensure that routing performance results are independent of initial loading and final unloading effects. The choice of an 8D hypercube and 16,000 total messages allows the simulation to execute in a reasonable amount of CPU time with stable results.

In the circuit-switched mode of routing, a source node has to finish the transmission of a message before it can start another request. A node which is not originating a message at a given time is considered to be idle. In accordance with the message injection rate chosen, the simulator for this study periodically determines the idle nodes and randomly picks one of these nodes to become the source node for a message request. The destination node is then chosen according to the type of traffic being simulated. Various traffic patterns were studied, including random destinations, bit-complement, transpose, bit-reverse, geometric, and hotspot types.

The simulator keeps track of all nodes that are busy in originating a message in every simulation time unit. The percent of busy nodes averaged over the simulation time interval is defined in this work as the "traffic load." This load can be adjusted indirectly by changing the periodic message injection rate into the system. For circuit-switched communication the setup time, i.e., the time involved to reserve the links between a source and destination, is directly proportional to the overall communication throughput achievable. The average message size in the system affects this setup time, since the links in the reserved path are occupied for a duration of time that corresponds to the size of the message. For a given traffic type, the simulator records the setup time for each message and keeps a running average of the setup times for all messages. The average setup time for the 10,000 messages is normalized by dividing it with the average message size.

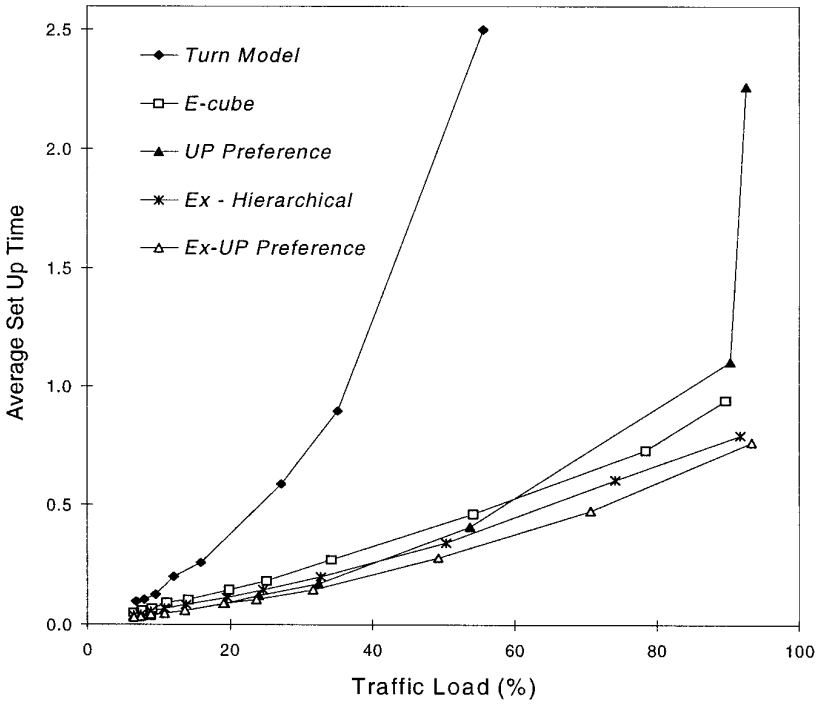

Fig. 11. Comparison of different routing algorithms under uniform distribution of destination nodes with randomly distributed message lengths.

Figure 11 shows the average setup time for a random distribution of destination nodes, where a source node is randomly selected from the list of idle nodes and where the destination node is randomly picked to be any node in the hypercube excluding the source node itself. The size of messages for the simulation shown in Figure 11 is set randomly to be between 100 and 900 with a mean of 500.

As can be seen from Figure 11, the *Ex-UP Preference* algorithm has the lowest average setup time over the entire range of network traffic loads. The *Turn Model* shows the worst performance of all the algorithms studied because it creates bottlenecks in routing the messages, which is also consistent with the results of Section 4. The *UP Preference* algorithm performs worse than the *E-cube* algorithm for high traffic loads. This effect was also noticed by Chiu et al. [1994]. This behavior occurs because uniform traffic favors an algorithm with better traffic load-balancing properties. Since *E-cube* creates perfect balance in routing (its standard deviation of *INTC* is 0), it performs better than the *UP Preference* algorithm. However, the *E-cube* does not perform as well as the *Ex-UP Preference* algorithm. The *Ex-UP Preference* algorithm has a much higher flexibility that compensates for a small loss in balance in routing the traffic. Figure 11 also includes a graph for the *Hierarchical* version of the *Ex-UP/DOWN Preference* algorithm. In the hierarchical scheme, the n-D hypercube is divided in two $n/2$-D subcubes. The lower $n/2$-D subcube

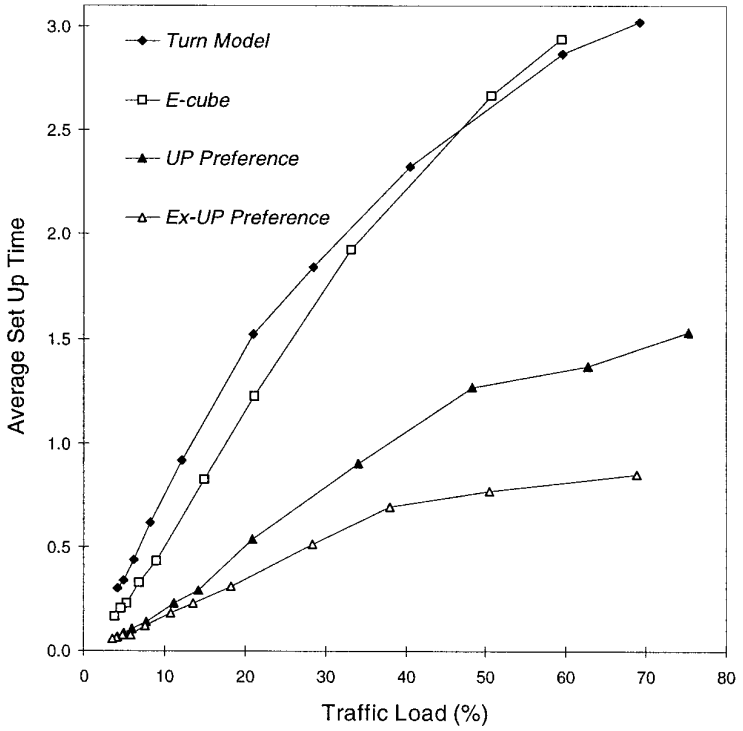

Fig. 12. Comparison of different routing algorithms under bit-reverse traffic with Poisson distribution of message lengths.

routes the messages according to the *Ex-UP Preference* rule while the upper $n/2$-D subcube follows the *Ex-DOWN Preference* rule. This scheme has a potential to balance the traffic better, but it looses some flexibility, since all required transitions in the lower $n/2$-D subcube have to be completed before moving to the upper $n/2$-D subcube to avoid deadlocks. Figure 11 shows the *Ex-Hierarchical* scheme performs better than other algorithms, but not as well as the *Ex-UP Preference*.

Figures 12 and 13 show the traffic simulation results for the bit-reverse and bit-transpose traffics, respectively. In the bit-reverse traffic, a source node with binary address $(x_{n-1}, x_{n-2}, \ldots, x_0)$ sends a message to the destination node with address $(x_0, x_1, \ldots, x_{n-1})$. In the bit-transpose traffic, the destination is selected to be the node with address $(x_{n/2-1}, x_{n/2-2}, \ldots x_0, x_{n-1}, x_{n-2}, \ldots, x_{n/2})$. Both bit-reverse and bit-transpose traffics occur in many practical computations and can cause worst-case behavior in oblivious routers for hypercubes [Fulgham 1993]. Figures 12 and 13 show the superior performance of the *Ex-UP Preference* algorithm as compared to other algorithms studied. The message lengths are based on a Poisson distribution in these figures. The *Ex-UP Preference* algorithm performs even better when the message lengths are uniformly distributed or when a larger mean for the message lengths is used.

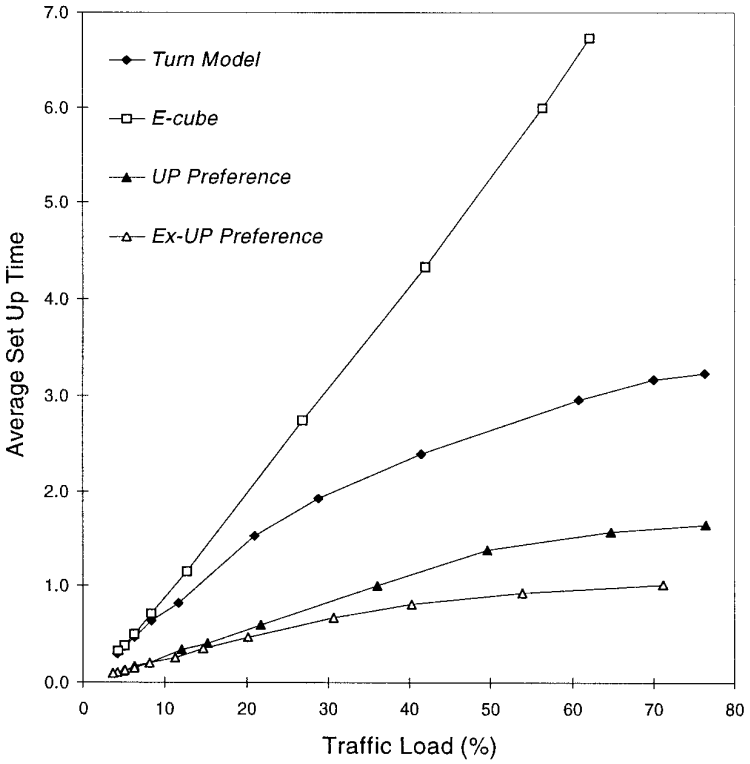

Fig. 13. Comparison of different routing algorithms under bit-transpose traffic with Poisson distribution of message lengths.

Figure 14 shows a comparison of different algorithms under hotspot traffic. A source node selects the destination node to be either the hotspot node or a randomly picked node. For this example the hotspot node is selected 15% of the time.

In Figure 14, the hotspot node is the node with the highest address, i.e., 11 . . . 1. The results in this case also demonstrate the superior performance of the *Ex-UP Preference* algorithm over other algorithms. Under high traffic load, the *E-cube* algorithm approaches the performance of the *Ex-UP Preference* algorithm. However, under light-to-medium traffic loads, the *E-cube* algorithm has a poor setup time.

We also simulated a comparison of different algorithms under geometric distribution of destinations. In this traffic, 50% of the destination nodes are randomly chosen to be one-hop distance away from the source node, 25% two-hops distance, 12.5% three-hops distance, and so on. This type of traffic represents those applications in which communication takes place mostly to neighboring nodes. Note that this traffic does not expose the flexibility of a routing algorithm to its full extent. However, the *Ex-UP Preference* algorithm still performs better than other algorithms, although the difference between *UP Preference* and *Ex-UP Preference* algorithms is relatively small.

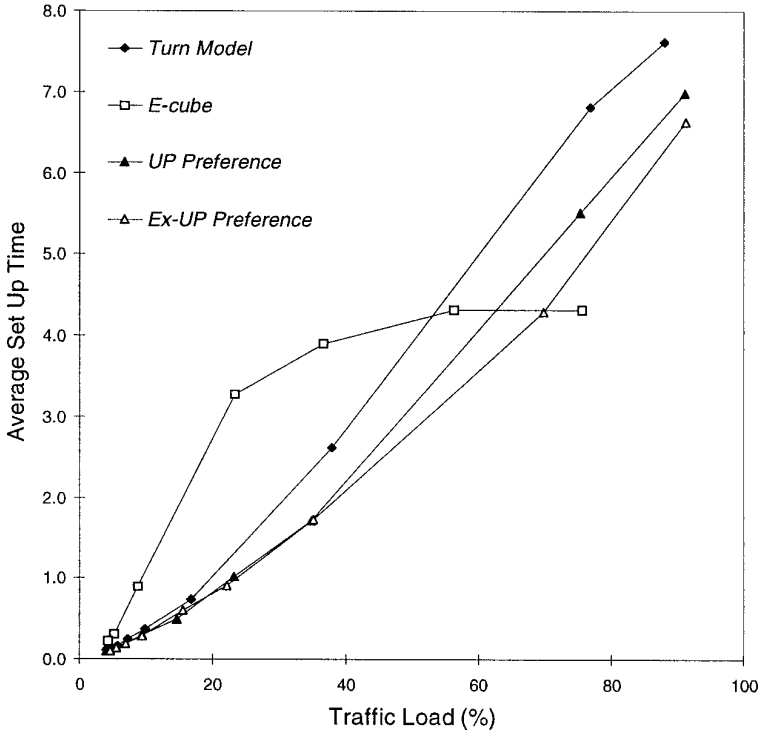

Fig. 14. Comparison of different routing algorithms under hotspot traffic (15% hotspot intensity) with Poisson distribution of message lengths.

A bit-complement traffic was also studied for the different algorithms. Except for the *Turn Model*, the remaining algorithms perform very well and are able to achieve the minimum possible setup time. This result is expected, since bit-complement traffic allows each message to reach its destination without causing a conflict with another message [Fulgham 1993]. In summary, all the various traffic simulations studied indicate the superior performance of the *Ex-UP Preference* algorithm over the existing popular algorithms.

6. APPLICATIONS TO OTHER NETWORKS/ROUTING SCHEMES

Although we have focused on the hypercube network in this article, the approach followed here to develop optimal, minimum-distance, deadlock-free routing algorithms can be applied to any recursively defined or well-defined network. By removing the fewest paths needed to eliminate cycles in the channel dependency graph in the smallest dimension (or order) network with physical loop(s), a deadlock-free routing can be obtained. By uniformly applying this routing to higher-order networks, a deadlock-free routing algorithm can be devised. As an example of application of the ideas developed in this article to other popular networks, we

outline an improved routing algorithm for the k-ary n-cube interconnection network in the following subsection.

6.1 Circuit-Switched Routing in the k-ary n-Cube Network

A k-ary n-cube network is a powerful class of networks of which the hypercube (two-ary n-cube) is one member. It also includes many other useful networks such as 2D toroidal mesh (k-ary two-cube), 3D mesh (k-ary three-cube), etc. In general, a k-ary n-cube network has k^n nodes, such that there are k nodes in each of the n dimensions. Each node is represented by n digits where each digit is in base k. There are $2n$ outgoing connections from each node, $\pm 1 \bmod k$ connections for each of the n digits representing the node.

In developing the minimum-distance, deadlock-free, circuit-switched routing in a k-ary n-cube network, note that the k nodes in each dimension form a physical loop when $k > 2$ (for $k = 2$ which is the hypercube, this is an immediate one-hop loop and does not cause any problems in routing, as previously shown in this article). Since deadlocks are related to the physical loops, we need to restrict the routing in the k nodes within each dimension. In addition, there are physical loops being created by the interaction of different dimensions when $n > 1$. Thus, routing also has to be restricted when we traverse different dimensions in an s-d path to make it deadlock free. In order to describe our improved deadlock-free algorithm, we use the same terminology as used by Boppana et al. [1992] to develop an equivalent circuit-switching algorithm for the *UP Preference* scheme on the k-ary n-cube network (the algorithm is referred to as f-cube routing by Boppana et al.).

All nodes in the k-ary n-cube network use four outgoing links in each dimension, termed as *Sigma high*, *Sigma low*, *Eta high*, and *Eta low* [Boppana et al. 1992]. Two extra links are needed in this case over normal networks to break the deadlock in each dimension [Boppana et al. 1992]. *Sigma* and *Eta* refer to the $(+1 \bmod k)$ and $(-1 \bmod k)$ connections, respectively. For each of the *Sigma* or *Eta* connections, there are two outgoing links from each node termed as the *high* and *low* links. For each message originating from a source node s with address $(s_{n-1}, \ldots, s_0)$ for a destination d labeled $(d_{n-1}, \ldots, d_0)$, first the required links in the s-d path are determined from the following algorithm:

Algorithm for determining required link transitions:

```
For i = 0 to (n − 1)      /* n = number of dimensions, k = nodes in one
dimension */
  length1 = |s_i − d_i|; length2 = |k − length1|; /* to see if Sigma or Eta link
  is a shorter path */
  if (s_i < d_i) {
    queue[i] = high
    if (length1 <= length2)
      link[i] = Sigma
```

```
    else link[i] = Eta }
  else if (s_i > d_i) {
    queue[i] = low
    if (length1 <= length2)
      link[i] = Eta
    else link[i] = Sigma }
  else do nothing      /* because s_i = d_i */
```

For example, in a 10-ary 2-cube system, if source node s is (8, 2), and the destination node d is (1, 6), the required link transitions as computed from the above algorithm are four *Sigma-high* links in dimension 0 and three *Sigma-low* links in dimension 1. Note that the above algorithm restricts choices of links within each dimension such that no deadlock is possible. If within the same dimension, source node is higher than the destination node, the *low* links will always be taken. Depending on which distance is shorter to the destination, the *Sigma* (+1) or *Eta* (−1) directions are pursued.

In order to remove the deadlock in routing due to interaction of different dimensions, the f-cube in Boppana et al.'s algorithm (which is similar to the *UP Preference* algorithm for the hypercube) states that a *Sigma-high* hop can be taken at any point in the order of required transitions. However, a *Sigma-low* or *Eta-high* or *Eta-low* hop in dimension i can be taken only if all required hops in dimension 0 to $(i - 1)$ have already been taken. Thus, in a way very similar to the *Ex-UP Preference* development, we can improve upon the f-cube algorithm by making it less strict in the order of allowed transitions. The pseudocode for our improved algorithm as would be implemented in a router in the k-ary n-cube network is as follows:

```
Improved routing algorithm for the k-ary n-cube system:
For each message request in the message queue in the router {
  highest_Sigma-high_dimension = highest remaining Sigma-high
transition's dimension      (0 if no Sigma-high transition left)
  If (message originated from this node) previous transition = Sigma-
  high; /* initialize */
  If (previous transition was Sigma-high)
      start_dimension = lowest remaining transition's dimension;
    else start_dimension = previous transition's dimension; /* since
         previous transition was an Eta or Sigma-low, all dimen-
         sions < previous dimension are locked */
  If (a link in start_dimension is available AND link type is needed)
    { Reserve the link and update the remaining transitions needed
      Send message request onward to node connected to this link }
  else {
    For (i = start_dimension to highest remaining transition's dimen-
    sion)
```

If (a link in dimension i is available AND link type is needed AND

(transition *i Sigma-high* OR ($i <$ *highest_Sigma-high_dimension*)))

{ Reserve the link and update the remaining transitions needed

Send message request onward to node connected to this link } } }

The router can take a *Sigma-high* transition without any restriction, but a *Sigma-low* or an *Eta* link is reserved only if all required lower-dimension transitions have been completed *or* if a higher-dimension *Sigma-high* transition is remaining. If a *Sigma-low* or an *Eta* transition is taken, it locks all lower dimensions, and only a higher-dimension *Sigma-high* transition when taken can unlock these. The router finds the next required link (>= lock dimension) that does not lock a needed dimension forever. If no link can be reserved, the router waits for this request and moves on to process the next request.

6.2 Application to Other Routing Schemes

While the focus of this article has been on the circuit-switched routing, the algorithms developed here are applicable to other routing schemes as well. For example, the *Ex-UP Preference* algorithm is directly applicable to packet-switched routing. In packet-switched routing, the resources are the packet buffers in each node, instead of the links. Thus, a deadlock-free packet-switched routing requires that there be no cyclic dependencies in reserving the buffers in each node. By replacing the link reservations in the *Ex-UP Preference* algorithm with the buffer reservation in a node, the algorithm will work for the packet-switched case.

The *Ex-UP Preference* algorithm has important implications in the case of wormhole routing. Currently, the most effective technique in wormhole routing is by Duato [1994] which uses two virtual channels per physical link. One of the virtual channels uses fully adaptive (unrestricted minimum-distance) routing while the other channel provides an escape route by using the deadlock-free *E-cube* algorithm. Thus, it is possible to use our new *Ex-UP Preference* algorithm in this article as the escape route, provided some additional constraints are specified for messages when they switch between the escape and the nonescape channels. Since our new algorithm is much more adaptive than the escape route originally used in Duato's scheme, it has the potential to further improve traffic throughput for wormhole routing. Similarly, it can also be applied to improve pipelined circuit-switched routing [Gaughan and Yalamanchili 1993]. Our future work lies in exploring these ideas.

7. CONCLUSIONS

A new class of minimum-distance deadlock-free algorithms for the hypercube network has been introduced in this article. A member of this new

class, termed as the *Ex-UP Preference* algorithm, is shown to be optimal in terms of the number of allowed s-d paths, with superior performance in comparison to existing algorithms. An existing algorithm, known as the *Turn Model*, also belongs to our new class of algorithms. However, it does not perform as well as other members of the new class, because it creates congestion in routing the traffic.

An analytical measure for judging the performance of adaptive minimum-distance routing algorithms has also been developed in this study. It takes into account the flexibility provided in routing as well as the balance of traffic load through different nodes in the hypercube network. This measure, when applied to the new algorithms developed in this article and other existing algorithms, indicates the relatively high flexibility of the new *Ex-UP Preference* algorithm. Further, the *Ex-UP Preference* algorithm has low standard deviation of the *INTC*, indicating a good balance in routing the traffic. Various types of traffic were simulated on a hypercube network employing different routing algorithms. In all traffic simulations, the *Ex-UP Preference* algorithm achieves the lowest s-d path setup time of all existing algorithms, supporting the analytical comparisons.

The ideas developed in this article are applicable to developing improved routing for other well-defined networks. An example of an improved routing scheme in the k-ary n-cube network has been presented. Recently, a class of networks termed as product networks is drawing considerable research interest. Usually one of the networks in a Cartesian product of two networks is the hypercube network, due to its many useful properties, e.g., the folded Peterson cube [Ohring and Das 1996], which is a product of the Peterson graph and the binary hypercube. The routing in such a case can be improved upon by employing the optimal hypercube routing algorithms developed in this article.

REFERENCES

BOLDING, K. 1992. Non-uniformities introduced by virtual channel deadlock prevention. Tech. Rep. UW-CSE-92-07-07, Univ. of Washington, Seattle, Wash.

BOLDING, K., FULGHAM, M. L., AND SNYDER, L. 1994. The case for chaotic adaptive routing. Tech. Rep. CSE-94-02-04, Univ. of Washington, Seattle, Wash.

BOPPANA, R. V., CHALASANI, S., AND RAGHAVENDRA, C. S. 1992. Adaptive packet-routing algorithms for k-ary n-cubes. Tech. Rep. UTSA-CS-92-105, Univ. of Texas, San Antonio, Tex.

CHIEN, A. A. 1993. A cost and speed model for k-ary n-cube wormhole routers. In *Proceedings of the 1993 Hot Interconnects Conference* (Aug.).

CHIU, G.-M., CHALASANI, S., AND RAGHAVENDRA, C. S. 1994. Flexible routing criteria for circuit-switched hypercubes. *J. Parallel Distrib. Comput. 22,* 279–294.

CHOI, S. B. AND SOMANI, A. K. 1993. Rearrangeable circuit-switched hypercube architectures for routing permutations. *J. Parallel Distrib. Comput. 19,* 2 (Oct.), 125–130.

DALLY, W. J. AND SEITZ, C. L. 1987. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput. C-36,* 5, 547–553.

DUATO, J. 1994. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In *Proceedings of the 1994 International Conference on Parallel Proceedings.* I-142–I-149.

FULGHAM, M. L. 1993. Performance of chaos and oblivious routers under non-uniform traffic. Tech. Rep. CSE-93-06-01, Univ. of Washington, Seattle, Wash.

GAUGHAN, P. T. AND YALAMANCHILI, S. 1993. Routing protocols for hypercube interconnection networks. *IEEE Comput.* (May), 12–23.

GAUGHAN, P. T. AND YALAMANCHILI, S. 1995. A family of fault-tolerant routing protocols for direct multiprocessor networks. *IEEE Trans. Parallel Distrib. Syst. 6* (May), 482–497.

GLASS, C. J. AND NI, L. M. 1994. The Turn model for adaptive routing. *J. ACM 41,* 5, 874–902.

GRAVANO, L., ET AL. 1992. Adaptive deadlock-free wormhole routing in hypercubes. In *Proceedings of the 16th International Symposium on Parallel Processing.* 512–515.

KIM, J. AND DAS, C. R. 1994. Hypercube communication delay with wormhole routing. *IEEE Trans. Comput. 43,* (July), 806–814.

KIM, J. AND SHIN, K. G. 1993. Deadlock-free fault tolerant routing in injured hypercubes. *IEEE Trans. Comput. 42,* 9 (Sept.), 1078–1088.

KONSTANTINIDOU, S. 1990. Adaptive, minimal routing in hypercubes. In *Proceedings of the 6th MIT Conference on Advanced Research in VLSI.* 139–153.

LI, Q. 1994. A dual-channel binary hypercube network. In *Proceedings of the 13th International Conference on Computers and Communications.* 268–274.

LIN, C.-C. AND LIN, F.-C. 1994a. Minimal fully adaptive wormhole routing in hypercubes. *Inf. Process. Lett. 50,* 6 (June), 297–301.

LIN, C.-C. AND LIN, F.-C. 1994b. Minimal turn restrictions for designing deadlock-free adaptive routing. In *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing.* IEEE Computer Society, Washington, D.C., 680–687.

NGAI, J. N. AND SEITZ, C. L. 1989. A framework for adaptive routing in multicomputer networks. In the *ACM Symposium on Parallel Algorithms and Architectures.* ACM, New York, 1–9.

OHRING, S. AND DAS, S. K. 1996. Folded Peterson cube networks: New competitors for the hypercubes. *IEEE Trans. Parallel Distrib. Syst. 7,* 2 (Feb.), 151–168.

SEITZ, C. L. 1985. The Cosmic Cube. *Commun. ACM 28*, 7 (July), 22–33.

SULLIVAN, H. AND BRASHKOW, T. R. 1977. A large scale homogeneous machine. In *Proceedings of the 4th Annual Symposium on Computer Architecture.* 105–124.

YANG, C. S. AND TSAI, Y. M. 1993. Adaptive and fault tolerant wormhole routing in hypercube network. *J. Inf. Sci. Eng. 9,* 2 (June), 253–270.

Received December 1996; revised March 1997; accepted April 1997