Adaptive Deadlock- and Livelock-Free Routing in the Hypercube Network

Gustavo D. Pifarré, Luis Gravano, Gustavo Denicolay, and Jorge L. C. Sanz, *Fellow, IEEE*

***Abstract*—This paper consists of two parts. In the first one, two new algorithms for wormhole routing on the hypercube network are presented. These techniques are adaptive and are ensured to be deadlock- and livelock-free. These properties are guaranteed by using a small number of resources in the routing node. The first algorithm is adaptive and nonminimal, and will be referred to as *Nonminimal*. In this technique, some moderate derouting is allowed in order to alleviate the potential congestion arising from highly structured communication patterns. The second algorithm, dubbed *Subcubes*, is adaptive and minimal, and is based on partitioning the hypercube into subcubes of smaller dimension. This technique requires only two virtual channels per physical link of the node. In the second part of the paper, a wide variety of techniques for wormhole routing in the hypercube are evaluated from an algorithmic point of view. Five partially adaptive algorithms are considered: the *Hanging* algorithm, the *Zenith* algorithm, the *Hanging-Order* algorithm, the *Nonminimal* algorithm, and the *Subcubes* algorithm. One oblivious algorithm, the *Dimension-Order*, or E-Cube routing algorithm, is also used. Finally, a *Fully Adaptive* Minimal algorithm is tried. A simple node model was designed and adapted to all the algorithms. For those algorithms that require fewer virtual channels per physical link, the extra logical channels are used as extra *lanes*. As a result of this, the storage and routing capabilities of the algorithms are equalized. For the empirical performance evaluation, several dynamic injection loads are used on a hypercube of 2^{10} nodes.**

***Index Terms*—Adaptivity, deadlock freedom, hypercube, livelock freedom, parallel communication, parallel computer, performance simulation, wormhole routing**

I. Introduction

MESSAGE routing in large interconnection networks has attracted a great deal of interest in recent years. Different underlying machine models have been used and proposed [2], [7], [14], [24], [27], [28], [31], [32], [36], [39]–[42]. Some fundamental distinctions among routing algorithms involve the length of the messages injected in the network, the static or dynamic nature of the injection model, special assumptions on the semantic of the messages, architecture of the network and router, degree of synchronization in the hardware, and others.

In terms of message length, several issues have been studied concerning the ways to handle long messages (of potentially unknown size) and very short messages (typically of 150 to 300 bits). In packet-switching routing, the messages are of constant (and small) size, and they are stored completely in every node they visit. In [18], a survey of some packet routing algorithms has been presented. In [16], simulation results have been shown comparing a number of different oblivious packet routing schemes on the hypercube. In wormhole routing [14], messages of unknown size are routed in the network. These messages are never stored completely in a node. Only pieces of the messages, called flits, are buffered when routing. For a review of recent wormhole methods, see [35]. In between packet-routing and wormhole routing lie some hybrid approaches [1], [3]. In [25], a message is routed by using a wormhole technique until it gets blocked in a node by traffic. In this case, the message is buffered completely in the node if buffers are large enough with respect to message length.

Two subjects of long-standing interest in routing are deadlock and livelock freedom. Techniques that perform without deadlocks or livelocks have been shown on different models. Some algorithms succeed in accomplishing deadlock-free or livelock-free routing only in a probabilistic sense [28], [38]. In other algorithms, deadlock freedom is guaranteed in a deterministic sense [19], [33]. Several techniques achieve this by defining an ordering on the critical resources and allowing each message to progress throughout the network by occupying resources in a strictly monotonic fashion (see, e.g., [5], [14], [26], [39], [40]). This idea results in a generation of a directed acyclic graph (DAG) of the resources.

A desirable feature of routing algorithms is adaptivity, i.e., the ability of messages to use multiple paths toward their destinations [13]. In this way, alternative paths can be followed based on factors that are local to each node, such as conflicts arising from messages competing for the same resources (e.g., buffers, crossbar access), faulty nodes, or links, among others. For example, consider the 16-node hypercube shown in Fig. 1. A message starting from node (0, 0, 0, 0) having (1, 0, 1, 1) as its destination can move to any of the following nodes as its first step: (1, 0, 0, 0), (0, 0, 1, 0), and (0, 0, 0, 1). An adaptive routing function will allow more than one choice among the three possible nodes. The decision about which node is actually selected is based on an arbitration mechanism that attempts to optimize the use of available resources. If one of the involved links out of (0, 0, 0, 0) is congested

Manuscript received November 9, 1992; revised November 30, 1993.

G. D. Pifarré is with the Departmento de Computación, Universidad de Buenos Aires, Buenos Aires, Argentina, and Advanced Solutions and Innovative Technologies Department, IBM Argentina, Buenos Aires, Argentina.; e-mail: pifarre@buevm2.vnet.ibm.com.

L. Gravano is with the Computer Science Department, Stanford University, Stanford, CA 94305 USA; e-mail: gravano@cs.stanford.edu.

G. Denicolay is with ESLAI, Escuela Superior Latino Americana de Informática, Buenos Aires, Argentina.

J. L. C. Sanz is with the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA, and Advanced Solutions and Innovative Technologies Department, IBM Argentina; e-mail: sanz@buevm1.vnet.ibm.com.

IEEE Log Number 9403097.

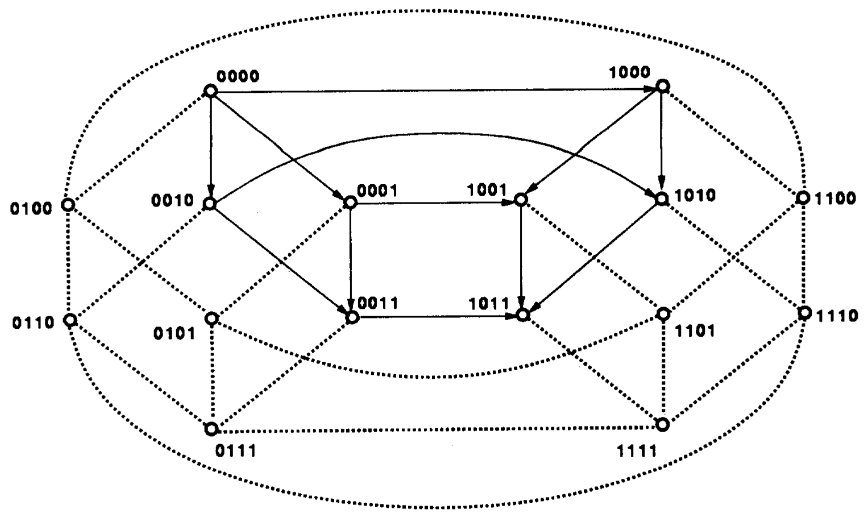


Fig. 1. The paths available in the 4-hypercube between node (0, 0, 0, 0) and node (1, 0, 1, 1).

or faulty, a message can use another link to progress to its destination. Thus, adaptivity may potentially yield more throughput by distributing the traffic over the resources of the network better. For example, if (1, 0, 0, 0) is chosen, the next hop can be made to nodes (1, 0, 1, 0) or (1, 0, 0, 1), and so on. Recent simulation results for packet switching on the 2-D mesh [17] have shown that adaptivity improves the performance of dynamic injection routing when compared to oblivious methods. Also, the same conclusion was obtained by [11] for several adaptive routing techniques for k-ary n-cube networks. However, finding deterministic and probabilistic bounds for static models of message injection in adaptive routing is still an open problem for all cube-type networks.

A *fully adaptive minimal* routing scheme is one in which all possible minimal paths between a source and a destination are of potential use at the time messages are injected into the network. Paths followed by the message depend on the traffic congestion found in the nodes of the network. For example, the minimal routing functions presented in [5] and [26] are *not* fully adaptive, because several minimal routes are not allowed to take place. Full adaptivity is a feature from which one can hope to obtain the best possible performance if no source of randomization is used. Full adaptivity has been used by Upfal in [41] to produce a deterministic optimal algorithm for routing in the multibutterfly. Multibutterflies are extremely rich in terms of the number of minimal paths between any pair of nodes.

In [37], fully adaptive algorithms for the hypercube and mesh have been presented for packet-switching routing. These algorithms are deadlock-free and can be implemented using only two queues per node. Recently, the principles shown in [37] have been used for the same routing mode in n-dimensional torus networks by using three queues per node [8], and this result is optimal for the given model and network [9]. Unfortunately, the methodology of [8] and [37] does not apply to wormhole routing. On the other hand, new algorithms for deadlock-free wormhole routing have been reported in [4], [11], [12], [15], [21], and [30].

In [12], a technique based on the use of the multiple independent *lanes* associated with each physical link in a routing node is shown. Given a fixed amount of storage space allocated to each physical channel, it is shown that breaking the storage into several buffers is a convenient methodology for improving network performance. In [17], a fully adaptive minimal wormhole routing technique for the bidimensional torus network was shown, requiring eight virtual channels per bidirectional link. In [21], fully adaptive wormhole algorithms were introduced for a variety of networks, including the hypercube, together with a methodology for the design of deadlock-free wormhole routing techniques. The same fully adaptive wormhole algorithm for the hypercube and a very similar design methodology were independently presented in [15].

This paper consists of two parts. In the first one, two algorithms for wormhole routing in the hypercube are shown.[1] Although other topologies have received increasing attention in recent years, the hypercube remains important, because several parallel machines were constructed with this topology [10]. Furthermore, the study of hypercube topologies is still relevant, because as the technology continues to improve, it may be possible to build even larger hypercubes in the future.

The techniques addressed in this paper are adaptive, deadlock-free, and livelock-free, and require a very moderate amount of resources in the routing nodes. The hypercube algorithm of Section III is adaptive, but nonminimal in the sense that some derouting is permitted. This nonminimality, though restricted, is aimed at providing each message with a wealth of paths for arriving at its destination. An important feature of this technique is that moderate connectivity is

[1] In fact, it should be noted that even though the presentation of the routing techniques in this paper is carried out in terms of wormhole routing, all of the techniques can easily be adapted for the packet-switching routing model.

necessary between the virtual channels of the same routing node. More precisely, an input virtual channel needs to reach only four output virtual channels. This is relevant in the hypercube because of the logarithmic growth of physical links per routing node. Also, eight virtual channels are needed for each bidirectional link.

This algorithm has another interesting property. Although it is possible to find traffic patterns for which deterministic performance of the algorithm is not proportional to the diameter of the hypercube, it is likely that the method performs well for many communication patterns, because at each step, every message will suffer some (limited) enforced potential derouting that is hoped to break many structured communication patterns. However, this derouting is limited in an attempt to achieve minimum latency, as well as freedom from deadlock and livelock, with moderate resources at each node. This technique is referred to as *Nonminimal* [20].

In Section IV, another deadlock-free adaptive wormhole-based routing algorithm for the hypercube interconnection, dubbed *Subcubes*, is presented [20]. This algorithm, unlike the one shown in Section III, is minimal. The central idea in this hypercube algorithm consists of decomposing the n-dimensional cube into disjoint subcubes of dimension $n/2$. Routes are generated by considering the hypercube as *hung* from one of the subcubes of dimension $n/2$. The idea of decomposing the hypercube into disjoint subcubes has also been used in [6]. In the *Basic Subcube* algorithm of Section IV-A, moderate hardware resources are necessary, because only two virtual channels per bidirectional physical link are necessary. This compares favorably with the three virtual channels per link needed by the natural wormhole extension of the packet-switched hypercube algorithm proposed in [26]. Also, the algorithm requires the same number of virtual channels per link as the deadlock-free oblivious routing algorithm for the hypercube presented in [14].

Although the new routing algorithm of Section IV is based on hanging the structure in a certain manner, it does not exhibit the same type of negative congestion analysis to which previous algorithms in the hypercube are subjected. Furthermore, the methodology allows the different alternatives when routing inside each of the subcubes of dimension $n/2$, and a choice of adaptive methods that perform well in a practical sense for low-dimensional hypercubes becomes appealing. (Remember that the size of each subcube is $\sqrt{N}$ for an N-node hypercube).

In the second part of the paper, a wide variety of algorithms for wormhole routing on the hypercube are evaluated. These techniques are adaptive, deadlock-free, and livelock-free, and require a very moderate amount of resources in the routing nodes. This part focuses on comparing the different routing strategies from an algorithmic point of view. Therefore, a simple hypercube node model was designed and adapted for all of the algorithms in order to obtain a meaningful comparison of the techniques. In this model, all of the nodes have the same number of input and output buffers per link in order to implement the different virtual channels that share each link. For those algorithms that require fewer virtual channels per physical link than those provided by this uniform node design, the extra logical channels are used as *lanes* [12]. As a result of this, the storage and routing capabilities of the different algorithms are equalized.

The comparison of wormhole routing algorithms for the hypercube involves techniques known from the literature, and also the new techniques. One oblivious algorithm, the *Dimension-Order* or *E-Cube* algorithm [14] is tried. Also, five partially adaptive algorithms are considered: the *Hanging* algorithm [5], [26], the *Zenith* algorithm [26], the *Hanging-order* algorithm [6], the *Nonminimal* algorithm (see Section III), and the *Subcubes* algorithm (see Section IV). Finally, a recently presented *Fully Adaptive* minimal algorithm is considered. This technique requires only four virtual channels per physical bidirectional link, and was developed independently by [15] and [21].

For the empirical performance evaluation, several dynamic simulations are tried. These simulations involve three different message sizes and four traffic patterns on a hypercube of 2^{10} nodes. Section VI describes the different simulations performed, as well as the results obtained.

II. Definitions

Definition 1: An n-dimensional hypercube is a network with 2^n nodes. The n dimensions of the hypercube are referred to as $n-1, n-2, \cdots, 1, 0$. Node $(x_{n-1}, \cdots, x_i, \cdots, 0)$ is connected to nodes $(x_{n-1}, \cdots, 1-x_i, \cdots, 0), 0 \leq i < n$.

Denoting by $(i_{n-1}, \cdots, i_0)$ the binary address of node I, and by $(j_{n-1}, \cdots, j_0)$ the corresponding address of node J, the bitwise exclusive or function of I and J shows the dimensions to be corrected in traversing from I to J. Sometimes a transition in dimension k, when moving from I to J, is labeled $0 \rightarrow 1$ if $i_k = 0$ and $j_k = 1$, and $1 \rightarrow 0$ if $i_k = 1$ and $j_k = 0$.

III. Algorithm *Nonminimal*

To improve network throughput, routing algorithms and their flow control should promote the effective use of the available physical links. Because contention in the output links of a node is likely to occur, algorithms may benefit from adaptive decisions in the way in which links are assigned to competing messages. A class of these algorithms is the one in which messages may be *derouted* away from their destination, thus following nonminimal paths. In general, if the amount of derouting is not controlled, messages may remain undelivered in the network forever, a situation known as *livelock*.

In this section, an adaptive, nonminimal, deadlock- and livelock-free routing algorithm for the hypercube is presented. Given a pair of nodes s and d, any route built by this function for a message going from s to d has length between n and $2n$ in an n-dimensional hypercube.

This nonminimal algorithm is inspired by routing on the multibutterfly: Multibutterflies correct dimensions starting from the highest, without paying attention to the lower ones. These lower dimensions will eventually be corrected at later stages. The experiments in Section VI show that the moderately larger routes created by the technique help improve the throughput in many situations.

The route of a message will consist of n phases. Each phase processes one dimension. Dimensions are processed from highest to lowest in an ordered manner. Each dimension has to be processed, regardless of whether it is already correct. During each of these phases, every message will have d alternative paths to take for some d fixed in advance. The least significant dimensions are a special case, as discussed below. The length of the paths to take at each phase will be 2 or 1, depending on the need to correct the current dimension or not, respectively. Messages will choose any one among these d paths adaptively. So, there are $\mathbf{O}(d^n)$ different (but not necessarily virtual-channel disjoint) paths between any pair of nodes. The **O** notation is needed because the least significant dimensions will have, in general, fewer than d paths associated.

The main idea behind this algorithm is to break the structured communication patterns by providing a wealth of paths between any source and destination, introducing a limited potential of derouting at each phase. At the start of phase i, dimensions $n-1$ through $i+1$ have already been corrected, and will never be modified again. When dimension i is being processed, a message m will either have to correct it or not. Regardless of this need, m will be sent first through a dimension $j < i$ (if i is not one of the least significant dimensions), regardless of whether this means *correcting* dimension j. Only after this will the message be sent through dimension i, if required. Dimension j will be chosen from within a set of d dimensions (again, except for the phases corresponding to the lowest dimensions, as discussed below). In principle, the set of d dimensions corresponding to a certain phase of the algorithm may be different for each node of the network, but it must be fixed in advance.

In other words, if a message m is at some node p, and m is at the ith phase of the routing algorithm, then m will leave p through any of the dimensions that node p will have associated with the ith phase. The set of dimensions corresponding to node p and phase i will be referred to as $\text{dims}(p,i)$.

Although the paths are not minimal, the number of derouting steps performed for each message is bounded; hence, the maximum length of the paths is fixed. Thus, if resources in the nodes are managed with fairness, the routing algorithm is livelock-free.

A. Virtual Channels Needed to Implement the Routing Algorithm

In this section, the set of virtual channels [14] needed to implement the routing algorithm in a deadlock-free manner is analyzed. Each virtual channel has one input and one output buffer, whereas several virtual channels share a physical link through multiplexing. A virtual channel is denoted by c_α, where α is a 4-tuple, as explained below. In the following, $p^{(i)}$ denotes the number that results from complementing the ith dimension in the binary representation of p.

Consider the node p, and a phase i of the routing algorithm. As stated above, p and i have associated with them a set $\text{dims}(p,i)$ of dimensions, $|\text{dims}(p,i)| = d$ if $i \geq d$, and $|\text{dims}(p,i)| = i$ otherwise.[2]

Thus, $|\text{dims}(p,i)|$ virtual channels are needed for derouting for node p and phase i. These channels are expressed as follows:

$$c_{i,1,i_1 p_{(i_1)}}, \cdots, c_{i,1,i_k,p_{(i_k)}},$$

where $dims(p,i) = \{i_1, \cdots, i_k\}$.

Furthermore, one virtual channel is needed for routing for each node p and phase i. This channel is used to correct the ith dimension of the address if this is necessary for the message to get closer to its destination. This virtual channel is denoted as follows:

$$c_{i,0,i,p_{(i)}}.$$

Channel $c_{i,d,j,p^{(j)}} (d = 0, 1)$ is associated with the physical link incident to p through dimension j. The input and output buffers associated with each virtual channel are referred to as routing or derouting buffers, according to the virtual channel type. In addition, each node p has an injection channel $c_{n,0,p}$ where messages are injected in the network.

B. The Routing Algorithm

As explained above, the routing algorithm for each message consists of n phases on an n-dimensional hypercube. Initially, each message is injected in the injection channel of its source node, and starts the routing process in phase $n-1$. Next phase i is described for a message m currently at a virtual channel of a node p.

Phase i:

```
/* Throughout the algorithm, $p$ is the node at which message $m$ resides */
/* Derouting Step First */
Send $m$ to any virtual channel in $\{c_{i,1,i_1,p^{(i_1)}}, \cdots, c_{i,1,i_k,p^{(i_k)}}\}$
where $\text{dims}(p,i) = \{i_1, \cdots, i_k\}$
/* Routing Step Follows */
If dimension $i$ needs to be changed
Then
Send $m$ to virtual channel $c_{i,0,i,p^{(i)}}$
Enter phase $i - 1$
```

Theorem 1: The routing algorithm described in Section III is correct and deadlock-free.

Proof: The proof can be found in the Appendix.

An important characteristic of this routing algorithm is the fact that each virtual channel has at most $d+1$ candidates as the next channel for any message that passes through it. Furthermore, these channels are fixed. So, each virtual channel has to be connected to at most $d+1$ other channels. This should be contrasted with the algorithm presented in [14] for routing on the hypercube.

[2] In fact, this does not need to be so rigid, as illustrated in Section III-C. For example, in the routing algorithm shown there for $d = 3$, $\text{dims}(p,i) = \emptyset$ for $0 \leq i \leq 3$, and for all nodes p.

Theorem 2: Injection channel $c_{n,0,p}$ has outdegree d ($d < n$), whereas channels $c_{i,1,j,p}$ and $c_{i,0,i,p}$ have outdegree $|\mathrm{dims}(p, i-1)| + 1$ and $|\mathrm{dims}(p, i-1)|$, respectively, if $j < i$.

Proof: The proof can be found in the Appendix.

Dimensions $(d-1), (d-2), \cdots, 1, 0$ are a special case. In general, these dimensions will have fewer than d dimensions associated with them in order to perform their corresponding derouting phase.

Now the main problem to solve is how to select for each node and for each phase the dimensions associated to have something at least *similar* to the *expansion property* [32], [41] while keeping the number of virtual channels associated with each physical link independent of the size of the network. Next an instance of the general algorithm presented above, with $d = 3$, is described.

C. An Instance of the Algorithm for $d = 3$

In this subsection, the routing algorithm presented in Section III is instantiated for $d = 3$, with some slight modifications. The set $\mathrm{dims}(p, i)$ has to be defined for each node p and for each dimension i. For the sake of uniformity, $\mathrm{dims}(p, i)$ will be equal to $\mathrm{dims}(p', i)$ $\forall p, p'$ nodes. As a result of this, the connections between the different virtual channels will follow a uniform pattern for every node of the network. However, by making the network uniform, the algorithm looses part of its potential ability for breaking the structure of the communication patterns.

For every node p, we have:

$$\begin{aligned}
&\mathrm{dims}(p, i) = \{i-2, i-4, i-6\} \text{ if } i \geq 6\\
&\mathrm{dims}(p, 5) = \{3, 1\}\\
&\mathrm{dims}(p, 4) = \{2, 0\}\\
&\mathrm{dims}(p, i) = \emptyset \text{ if } 0 \leq i \leq 3.
\end{aligned}$$

From the definition above, it follows that the lowest dimensions have fewer dimensions associated for performing the corresponding phase. Therefore, these dimensions have less nonminimal adaptivity in their corresponding phase of the routing algorithm, as discussed above.

Then an oriented physical link corresponding to dimension i ($2 \leq i \leq n-7$) has associated four virtual channels, a number that is independent of the size of the network. The other links have one, two, or three virtual channels associated, depending on their particular dimension.

In Figs. 2 and 3, the possible message transitions between input and output buffers of a node are shown. These figures show the derouting and the routing steps of the ith phase, respectively.

Fig. 2 shows the possible derouting moves from the routing and derouting buffers. A message beginning the derouting step of phase i may be in a derouting buffer or in a routing buffer. If the message did not need to correct dimension $i+1$, it will enter phase i immediately after the derouting step of phase $i+1$. For example, the message in the derouting buffer attached to virtual channel $c_{i+1,1,i+1-4,c}$ in Fig. 2 has ended the derouting step of phase $i+1$, so, if the message does not need to correct dimension $i+1$, it will start the derouting step of phase i. Thus, the message can leave the node through any of the following channels:

$$c_{i,1,i-2,c^{(i-2)}}, c_{i,1,i-4,c^{(i-4)}}, c_{i,1,i-6,c^{(i-6)}}.$$

On the other hand, a message that has ended the routing step of phase $i+1$ must also start the derouting step of phase i. The message in the routing buffer associated with the virtual channel $c_{i+1,0,i+1,c}$ in Fig. 2 has corrected dimension $i+1$, and now, in the derouting step of phase i, can go through any of the same three virtual channels as in the previous case.

Fig. 3 refers to the routing moves possible from a derouting buffer. A message can start the routing step of phase i only after it ends the derouting step of the same phase. Thus, a message residing in a derouting buffer after the derouting phase of phase i that needs to correct dimension i must be sent to the routing buffer of dimension i. For example, consider a message in the derouting buffer associated with the virtual channel $c_{i,1,i-4,c}$ in Fig. 3. If this message needs to correct dimension i, it must go to the routing output buffer associated with virtual channel $c_{i,0,i,c^{(i)}}$.

In summary, the algorithm that results from the definitions above needs at most four virtual channels per directed physical link. Each of these channels has to be connected to at most four other channels. As a result of this, no 1 to n switches are required. There are $3^{n-6}2^2$ different paths between any pair of nodes. For example, in Fig. 4, the 12 paths from node (0, 0, 0, 0, 0, 0, 0) to node (0, 0, 0, 0, 0, 1, 0) in a 7-D hypercube are shown. In this figure, each phase is presented as two steps: a derouting step and a routing step.

IV. Algorithm *Subcubes*

In this section, an adaptive, minimal, deadlock-free routing algorithm for the hypercube that requires only a moderate amount of resources for its implementation is described. This algorithm is based mainly upon considering the hypercube as a hierarchical network in which each node is a small hypercube. These small hypercubes are referred to as *subcubes*.[3] The routing functions over this network are defined as if the hypercube were *hung* from one of these subcubes [5], [26]. This hanging defines a leveled structure of subcubes. By considering the hierarchical network as hung from one of these subcubes, an order in which these subcubes will be visited can be defined. In addition, if each of these subcubes is visited in a deadlock-free manner, a deadlock-free routing algorithm on the whole network may be obtained.

Consider an n-dimensional hypercube with $N = 2^n$ nodes. First of all, the size of each of the subcubes mentioned above has to be chosen. Each subcube consists of 2^k nodes, $0 \leq k \leq n$. Then k dimensions $i_1, \cdots, i_k$ have to be chosen from the set $\{0, \cdots, n-1\}$. By fixing the $n-k$ remaining dimensions, $l_1, \cdots, l_{n-k}$, a subcube of size 2^k is defined. So, each subcube is identified by the value in each of these $n-k$ fixed dimensions. Therefore, by selecting these k dimensions, 2^{n-k} subcubes of size 2^k are defined.

For example, if $n = 10, k = 5$, and $i_j = 2(j-1), 1 \leq j \leq k$, then the subcube determined by $l_j = 0, 1 \leq j \leq n-k$

[3]The idea of defining a hierarchical topology using the hypercube has also been used in [6] and [24].

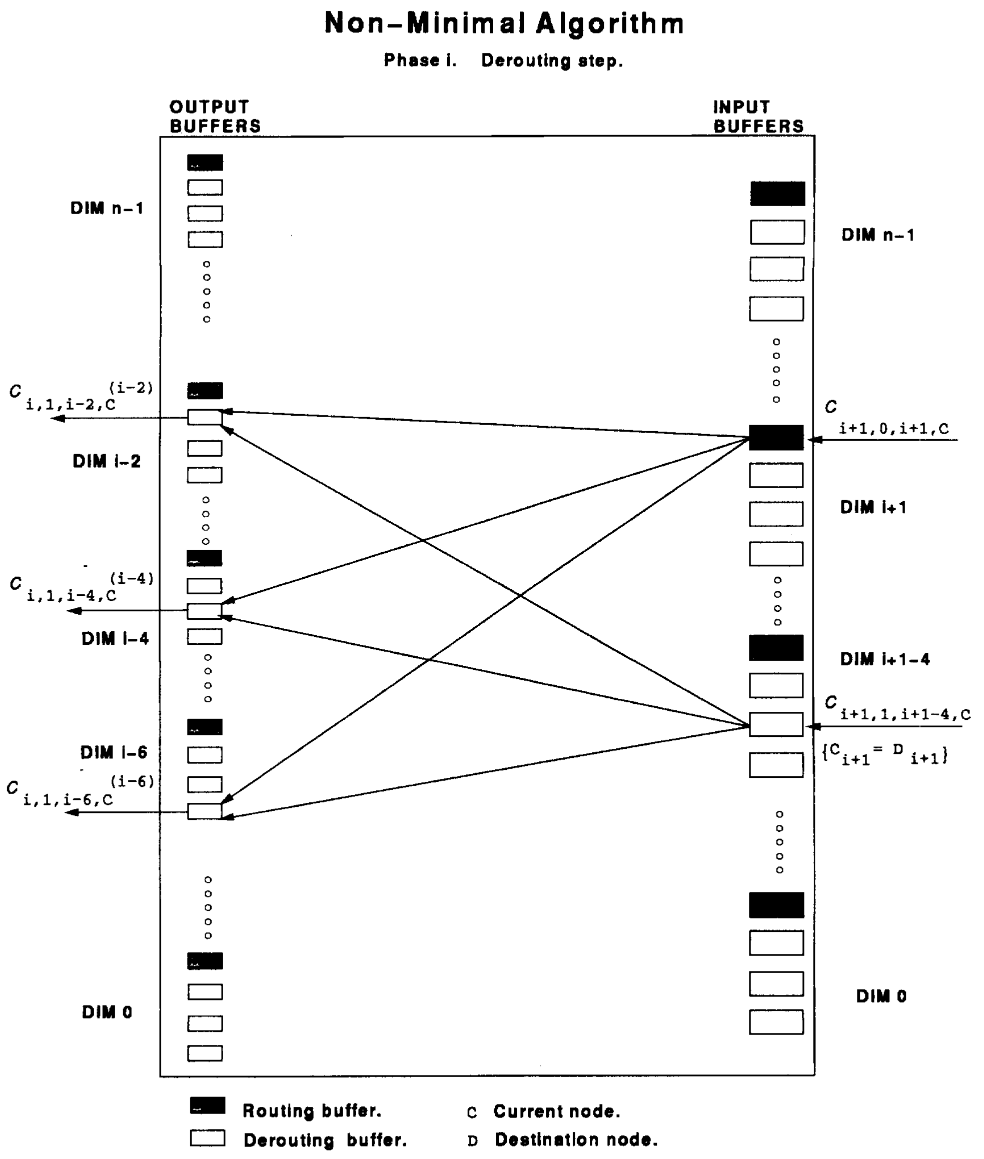


Fig. 2. The phase i derouting step for the algorithm of Section III-C on an n-hypercube.

consists of all those nodes $x_9 \cdots x_0$ such that $x_1 = x_3 = \cdots = x_9 = 0$, and is referred to as $S^{1,3,5,7,9}_{0,0,0,0,0}$.

In general, we have:

$$S^{l_1,\cdots,l_{n-k}}_{y_1,\cdots,y_{n-k}} = \{x_{n-1} \cdots x_0 : x_{l_j} = y_j \forall 1 \leq j \leq n-k\},$$

where $\forall 1 \leq j \leq n-k : 0 \leq l_j \leq n-1, y_j \in \{0,1\}$.

In order to define the routing function, how to visit each subcube has to be decided. As mentioned above, the main idea is to consider each subcube as a big node, and define a routing strategy for this new "hierarchical" network, a hypercube with 2^{n-k} big nodes.

The strategy chosen for this hierarchical network is the following. The hierarchical hypercube is regarded as hung from one of the subcubes. The path of a message going from one subcube to another subcube will consist of two phases. During the first hierarchical phase, the message will visit the nodes of the hierarchical network by moving downward, considering the network as hung from a fixed subcube. During the second hierarchical phase, the message will visit the nodes by traveling upward. Therefore, if the subcube from which to hang the network is $S^{l_1,\cdots,l_{n-k}}_{0,\cdots,0}$, the two phases are defined as follows. In the first phase, a dimension $l_i, 1 \leq i \leq n-k$, will be corrected only if that correction is a $0 \rightarrow 1$ transition. In the second phase, a dimension $l_i, 1 \leq i \leq n-k$, will be corrected only if that correction is a $1 \rightarrow 0$ transition. During the first phase, any $0 \rightarrow 1$ transition can be performed,

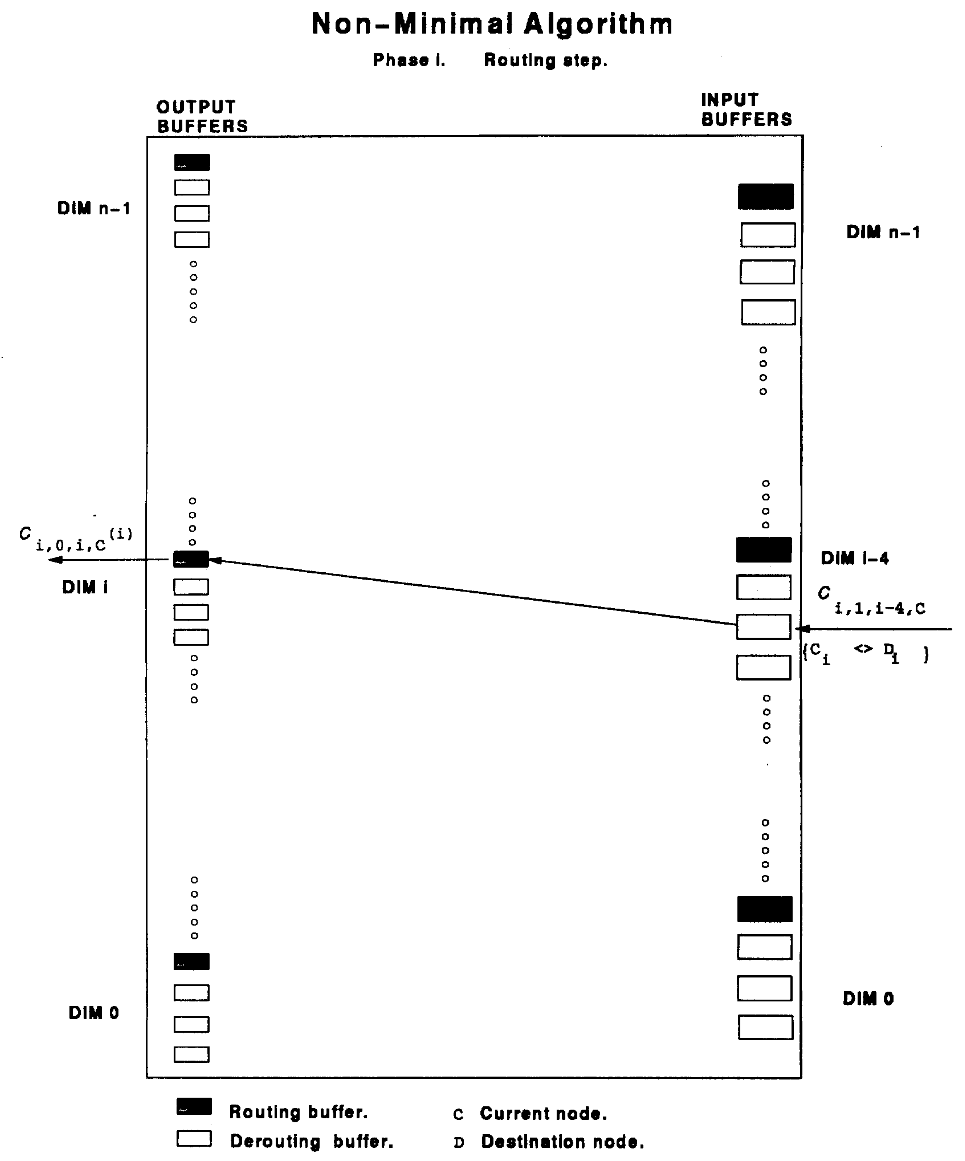


Fig. 3. The phase i routing step for the algorithm of Section III-C on an n-hypercube.

and in any order. So, the order in which these dimensions will be changed can be chosen adaptively depending on local congestion. Analogous remarks apply for the second phase. The only restriction is that a message will enter the second phase only once it has finished its first phase.

Note that in each phase of this strategy, each bidirectional link is used in a different direction. Each link will be used in only one direction in each phase. Therefore, to achieve deadlock freedom, this part of the routing algorithm can be implemented using two virtual channels [14] per bidirectional link. Each of these two virtual channels corresponds to one of the possible orientations of the underlying bidirectional link.

Hanging the hypercube from a single node (i.e. $k = 0$) is a well-known methodology for generating deadlock-free routing algorithms [5], [26], [37]. When $k = 0$ the two-phase algorithm used above is known to generate severe congestion at node $1 \cdots 1$ (if the network is hung from node $0 \cdots 0$) for moderate network sizes. If k is different from 0, the congestion phenomenon is distributed to all the nodes of the subcube labeled $S_{1,\cdots,1}^{l_1,\cdots,l_{n-k}}$. Because 2^k nodes are present in this subcube, congestion is less likely to arise at a single node.

Several routing algorithms can be defined on top of this routing strategy for routing between any pair of nodes of the hypercube. These routing algorithms differ in the way in which messages move within each subcube. An important point is that each subcube may use a different "internal" routing strategy. This will have an impact on the routing performance. As an illustration of the above ideas, several routing strategies

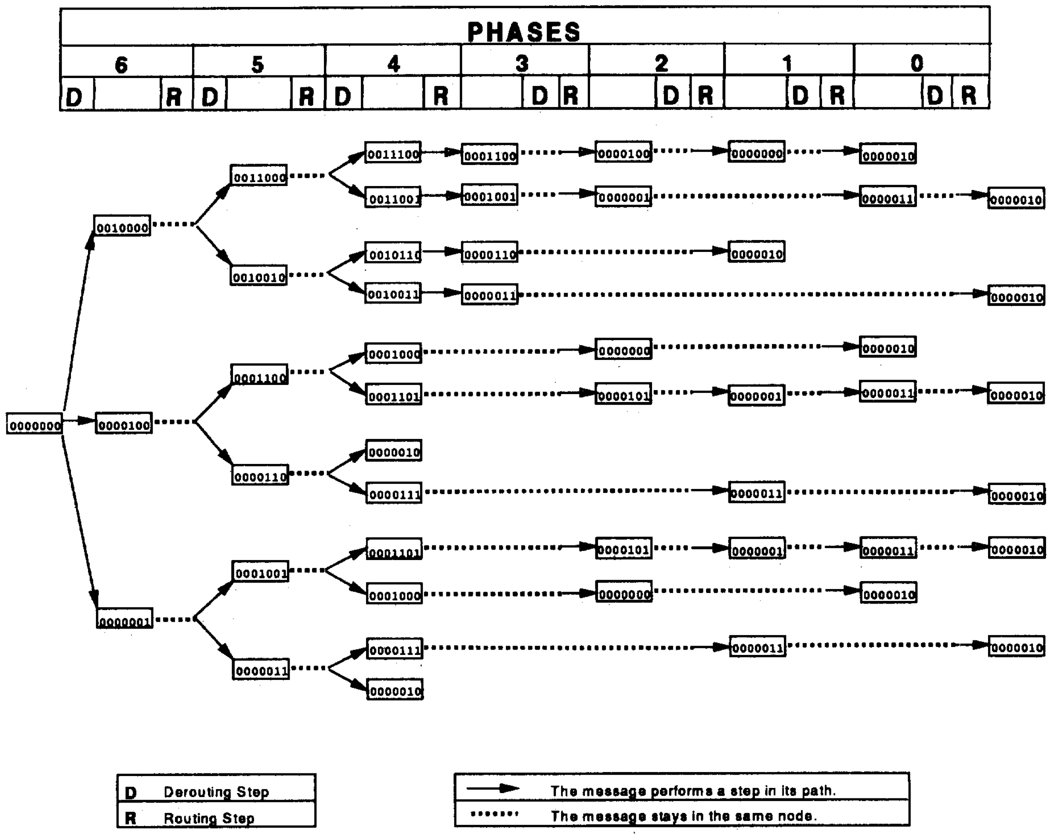


Fig. 4. The paths available on the 7-hypercube from node (0, 0, 0, 0, 0, 0, 0) to node (0, 0, 0, 0, 0, 1, 0) with the *Nonminimal* algorithm of Section III-C.

can be devised. The reader is referred to [22]. In the next section, one such algorithm is presented.

A. Basic Subcube Algorithm

This algorithm allows each message to move *within* the subcubes only during its first phase. A message will have as many chances to correct the k bits $i_1, \cdots, i_k$ as the number of hierarchical steps it has to take during the first phase, as described in Section IV. By taking any of these hierarchical steps, the message passes from one subcube to another, and different routing strategies can be followed within each subcube, as pointed out above. The message will have to have corrected all the dimensions $i_1, \cdots, i_k$ before starting the second hierarchical phase. If a message does not have to perform any step in the first hierarchical phase, it will start its path toward its destination by correcting all the dimensions $i_1, \cdots, i_k$ according to the routing strategy corresponding to its source subcube. After this it will start moving toward its destination subcube following the second hierarchical phase.

The routing strategy in each subcube can be chosen independently. The following are two possible choices:

- Routing by correcting dimensions in order [14], or
- Routing as if each subcube were hung from one node [5], [26], [37].

Either of these routing strategies requires two virtual channels per bidirectional link for its deadlock-free definition. In this way, the set of virtual channels needed at each node will be the same, regardless of the node's belonging to a particular subcube. Furthermore, each subcube can be hung from any node, or can choose any permutation of the set of dimensions $\{i_1, \cdots, i_k\}$ as the order in which these dimensions are corrected.

Figs. 5 and 6 depict the routing function for an instance of this algorithm. A 4-hypercube is considered. Suppose that $k = 2$ and the four associated subcubes are defined by the two most significant dimensions. Fig. 5 shows the available paths during the first phase, when a hanging strategy is chosen in each subcube. Notice that not all arrows within cubes can be taken at arbitrary moments. Fig. 6 shows the second phase of the algorithm. Fig. 7 shows all the available paths from node (0, 0, 0, 1) to node (1, 1, 1, 0). Note that all of these paths are realized during the first phase, so messages never reach the second phase. On the other hand, the paths from node (0, 1, 0, 1) to node (1, 0, 1, 0) shown in Fig. 8 involve both phases of the algorithm.

In summary, there are three sources of adaptivity.

1) Correcting dimensions $l_1, \cdots, l_{n-k}$ throughout the two hierarchical phases;

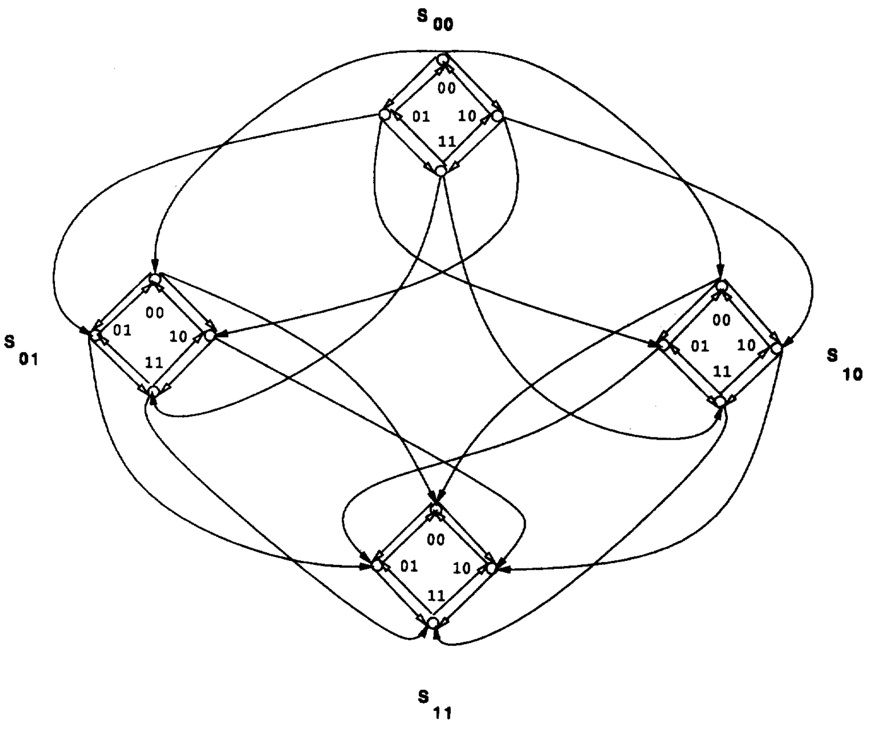


Fig. 5. The first phase of the *Basic Subcube* algorithm on a 4-hypercube.

2) Correcting dimensions $i_1, \cdots, i_k$ within *one* subcube; and
3) Correcting dimensions $i_1, \cdots, i_k$ within *different* subcubes: If a message is still in its first hierarchical phase, and it has to visit other subcubes before ending this phase, then this message will have more chances to correct the dimensions in $\{i_1, \cdots, i_k\}$ as it visits other subcubes; therefore, when correcting the dimensions in $\{i_1, \cdots, i_k\}$, and so when moving within a subcube, the message will be able to do the following:
   a) switch to the second phase of the subcube routing algorithm corresponding to hanging the subcube from a node, even if some dimensions that need to be corrected in this first phase have not been corrected yet [26], or
   b) leave some dimensions incorrect when it is these dimensions' turn to be corrected, if dimensions are corrected in order within this subcube.

All the dimensions that have not been corrected will have the chance to be changed within another subcube that will be visited before entering the second hierarchical phase.

B. Possible Sources of Randomness

Once the size of the subcubes has been chosen,[4] there are several parameters that can be set arbitrarily to incorporate randomness to the routing algorithm. All of the following parameters must be fixed before the routing process starts:

- The $n - k$ dimensions $l_1, \cdots, l_{n-k}$ that characterize the subcubes;
- The subcube from which to hang the hierarchical network (this choice determines the way in which the subcubes will be visited, and so it defines the order in which the different routing strategies to correct dimensions $i_1, \cdots, i_k$ will be applied.); and
- The node within each subcube from which to hang the subcube, or the order in which the dimensions are corrected within a subcube, depending on the routing strategy used within each subcube.

V. Wormhole Routing Algorithms to be Compared

In this section, the wormhole routing algorithms to be compared are briefly revisited. Table I summarizes the resources for each algorithm in terms of the number of virtual channels needed per bidirectional link.

A. Dimension-Order, or E-Cube, Algorithm

Dimension-Order, also referred to as *E-Cube*, works as follows. After a message has been injected in the network, it will be routed by correcting the dimensions from the most significant to the least significant. A message will not attempt to correct a dimension until all the higher ones have been corrected. This is an oblivious algorithm; there is no source

[4] $k = n/2$ seems to be a good choice, because it balances the size of the subcubes and the size of the hierarchical network.

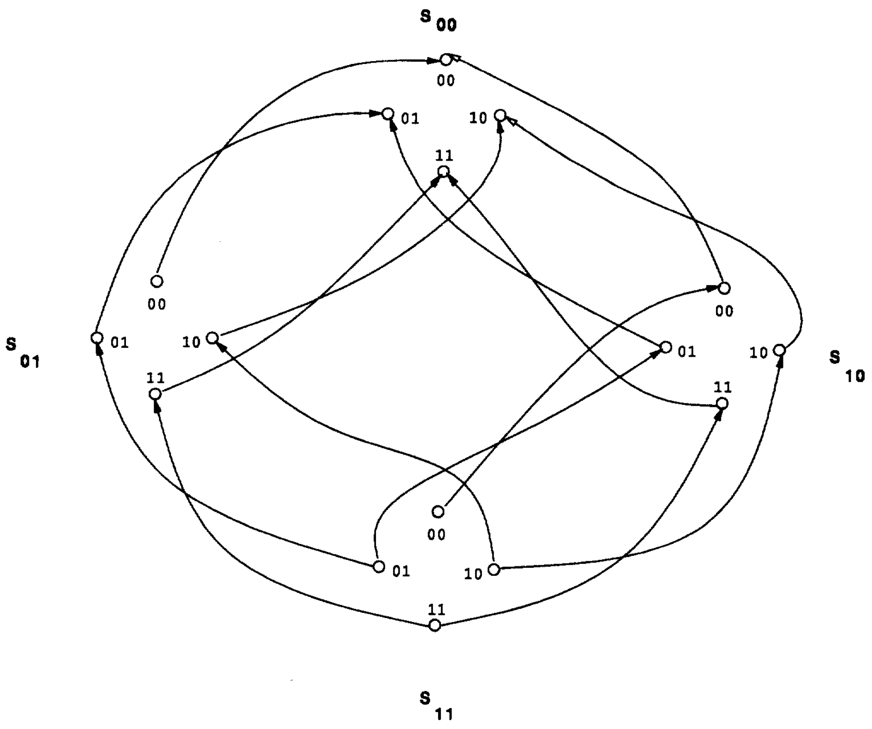


Fig. 6. The second phase of the *Basic Subcubes* algorithm on a 4-hypercube.

TABLE I
THE NUMBER OF VIRTUAL CHANNELS PER BIDIRECTIONAL LINK REQUIRED BY EACH ALGORITHM

Algorithm	Number of virtual channels
Basic Subcube	2
Hanging	2
Hanging-Order	2
E-Cube	2
Zenith	3
Fully Adaptive	4
Nonminimal $d = 3$	8

of adaptivity. Two virtual channels per bidirectional link are used. This algorithm has been described in [14].

B. Hanging Algorithm

For any message m with source s and destination d, let node $(s \vee d)$ be the *Zenith* of m [26]. Each message will be routed in two phases. During the first phase, the message is routed with the objective of decreasing its Hamming distance to its Zenith node, and thus only $0 \longrightarrow 1$ transitions are allowed. Dimensions are corrected following no particular order; i.e., the algorithm is adaptive. During the second phase, having reached its Zenith node, the message is routed (adaptively), decreasing its Hamming distance to its destination d. Therefore, the $1 \longrightarrow 0$ transitions take place. There are two virtual channels per bidirectional physical link, one for each phase. This algorithm has been described in [5] and [26].

C. Hanging-Order Algorithm

This algorithm can be seen as a combination of the *Hanging* and *Dimension-Order* algorithms. A message is allowed to make any $1 \longrightarrow 0$ transition at any moment (as in the first phase of the *Hanging* algorithm), and also a $0 \longrightarrow 1$ if this transition occurs in the highest dimension still to be corrected. Note that there is more adaptivity here than in *Hanging*, because it is not required that messages make all of the $1 \longrightarrow 0$ transitions before taking $0 \longrightarrow 1$ transitions. This a minimal algorithm that can be implemented using just two virtual channels per bidirectional link. This algorithm has been presented in [6].

D. Zenith Algorithm

For any message m with destination node d, currently in some node p, let node $(p \wedge d)$ be the *Nadir(p, d)* of m. Messages can be divided into two classes. Class 1 messages are routed (adaptively), decreasing their Hamming distance to their Zenith first (Ascend Phase). When the Zenith has been reached, Class 1 messages are routed (adaptively), decreasing their Hamming distance to their destination d (Descend Phase).

On the other hand, Class 2 messages are routed (adaptively), decreasing first their Hamming distance to their Nadir node (Descend Phase). After the Nadir has been reached, they are routed (adaptively) to decrease their Hamming distance to their destination (Ascend Phase). Every message m with source s

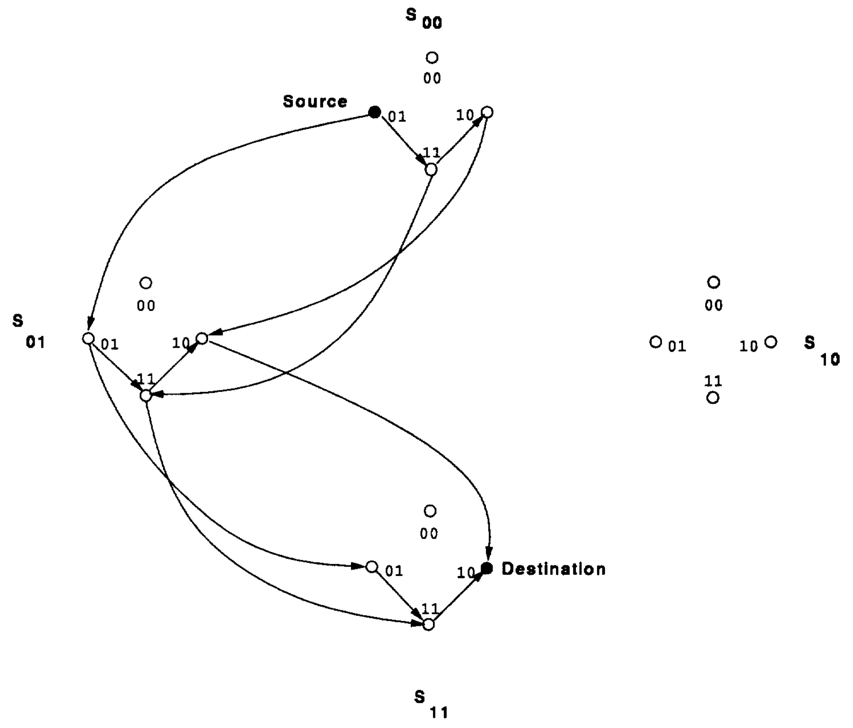


Fig. 7. The available paths from node (0, 0, 0, 1) to node (1, 1, 1, 0) using the *Basic Subcube* algorithm.

and destination d is injected into the network as a Class 1 message. It is routed first following the Ascend Phase. After reaching its Zenith, it proceeds with the Descend Phase. If a Class 1 message m in its Ascend Phase is in some node p other than its Zenith, and if all the output buffers that would lead m to its Zenith are full, then the message switches from Class 1 to Class 2. Because the message is now a Class 2 message, it will be routed first following a Descend Phase starting from node p to node Nadir(p, d), where d is m's destination. When the message has reached Nadir(p, d), the Ascend Phase begins, and proceeds until m reaches its destination d.

There are three virtual channels per physical link, called 1, 2, and 3. Channel 1 holds only Class 1 messages in their Ascend Phase, Channel 2 holds both Class 1 and Class 2 messages in their Descend Phase, and Channel 3 is for Class 2 messages during their Ascend Phase. This algorithm has been presented in [26].

E. Fully Adaptive Algorithm

Each directed link has two virtual channels. One of these channels is referred to as a *star-channel* [21]. Star channels will be used judiciously, observing certain rules to guarantee freedom from deadlock, whereas the rest of the channels will be used freely. The head of a message m may enter the star channel corresponding to the dimension i *only* if i is the most significant of the dimensions that m has to correct to arrive at its target node.

On the other hand, m can correct any of the dimensions as far as the corresponding nonstar channels are used for accomplishing the transitions. This technique does not discard any minimal path for potential routing and requires only four virtual channels per bidirectional link, and deterministic freedom from livelock and deadlock are still ensured. This result was independently developed by [15] and [21].

VI. Simulations

In this section, the algorithms described in Section V for wormhole routing on the hypercube are evaluated and compared with the techniques of Sections III and IV. The comparison is carried out from an algorithmic point of view. Therefore, a simple node model was designed and adapted to all the algorithms. For empirical performance evaluation, several simulations with continuous routing are tried. These simulations involve three message sizes and four traffic patterns on a hypercube of 2^{10} nodes.

A. The Model for the Routing Node

A high-level functional description of the routing node for implementing the algorithms is discussed in this section. A simple node model was designed for all the algorithms. Each directed link has four virtual channels associated with it, because this is the number of virtual channels required by the *Nonminimal* algorithm. Because all of the other algorithms use fewer virtual channels, several lanes [12] are used for one logical virtual channel. Thus, different messages can take the same virtual channel by using different lanes associated with the channel. Each lane of a virtual channel is implemented by an independent input and an independent output buffer; i.e., each input buffer receives messages from only one output buffer. Note that by normalizing the number of virtual channels per link for all of the algorithms, not only the storage but also the routing capability are equalized. This allows for a fair comparison of the performance of the different algorithms.

Thus, each node consists of a central crossbar, input and output buffers, and an injection and a delivery buffer. All of the input buffers and the injection buffer are connected to the crossbar's inputs. The output buffers and the delivery buffer are connected to the crossbar's outputs. It should be noted that neither the injection nor the delivery buffers differ from the rest of the buffers regarding their access to the crossbar. In principle, all connections in the central crossbar are possible, with the only restrictions being those imposed by the particular routing function of each algorithm. All buffers have the capacity for one flit. Nodes are able to consume one flit per routing cycle. In the simulations, it was considered that the header of a worm consisted of only one flit.

All of the algorithms considered in this paper, except for *Fully Adaptive*, are based on routing functions that form channel dependency graphs [14] that are acyclic. These algorithms need to find only the tail of a worm in an output buffer to be able to establish a connection for a header of a worm to that output buffer. *Fully Adaptive*, because it uses a different approach with dynamic transitions between the resources, requires that it be checked that the input and output buffers implementing a given virtual channel are both free before establishing a new connection for that virtual channel. Otherwise, deadlock may arise.

Although all the compared algorithms were simulated with the same number of input and output buffers per physical link, some consideration must be given to the complexity of the node models. In those algorithms with a small number of required virtual channels per physical link, a simpler node model can be used. Each algorithm requires a crossbar switch that performs the routing function. Regardless of the lanes added to the algorithm to equalize the storage and the routing capabilities of all of the algorithms, the fewer virtual channels an algorithm needs, the fewer connections the associated crossbar will perform, and so, the simpler the resulting switch will be. When the lanes are added to the virtual channels, only a second step of arbitration between the different lanes that belong to the same virtual channel is needed. Consequently, minimizing the number of virtual channels is a key issue in the design of routing algorithms. Moreover, an algorithm with only one virtual channel per physical link, such as the subcube algorithm presented here, does not require multiplexing over physical links, so the routing node is simpler and faster.

Another important consideration is the switching required from the input to the output buffers. With the exception of algorithm *Nonminimal*, every algorithm requires some input buffers to be connected to at least n output buffers. Furthermore, in some cases, as in algorithms *Zenith* and *Hanging*, there are different patterns of connections inside the different nodes of the network, so a full crossbar connection is required to obtain a uniform node design.

In the case of algorithm *Nonminimal*, a limited buffer connectivity is required inside a node. Only four connections are needed for a derouting input buffer (see Figs. 2 and 3), and thus the corresponding routing decision is very simple. If the current dimension bit has to be corrected, there will be only one output buffer to be used, and the message must wait until this buffer becomes free. If the current dimension does not need to be changed, then any of the three connected derouting output buffers could be used, and thus only an arbitration step is required among them. On the other hand, from a routing input buffer, there is no routing decision at all (see Fig. 2). A routing input buffer is connected to three output buffers, and a message in that input buffer can take any of these outputs buffers. So, only a simple arbitration among these competing input buffers is needed. The above features suggest a design of a simple and fast routing node, thus making algorithm *Nonminimal* competitive in spite of its relatively large number of virtual channels.

B. Network Activity

This subsection describes the activity of the network assumed for the experiments. Because latency will be measured in terms of *routing cycles*, a definition of the amount of work involved in a cycle is needed. The routing cycle consists of two phases: the *node cycle* and *link cycle*, which occur simultaneously.

Some decisions were made in order to keep the node cycle as short as the link cycle. Only one crossbar connection can be established during a single node cycle, because allowing for more would require some mechanism to handle collisions when two or more worm headers might want to enter the same output buffer. The overhead resulting from this arbitration would make the node cycle much longer than the link cycle.

- **Node Cycle**: As explained in Section VI-A, each node has one crossbar. A connection is said to be established if the header of a worm has previously set it in the crossbar. During the node cycle, each node passes the flits from the input buffers to the output buffers for all previously set connections, provided that the output buffers are empty. After the last flit of a worm has passed through the crossbar, it releases the crossbar connection in that routing cycle. During this cycle, at most one head from the input buffers is allowed to establish a connection to an output buffer. A fair policy to prevent starvation is used. The injection buffer has the same priority as the rest of the buffers. The head of a message that has arrived at an input buffer of its destination node tries to establish a connection with the delivery buffer of that node.
- **Link Cycle**: During the link cycle, each directed link with a flit of a message in its associated output buffer sends it to the corresponding input buffer, provided that this buffer is empty. For those algorithms with more than one output buffer per directed link, only one flit is transferred. These buffers are served in a round-robin fashion. The first buffer, according to this round-robin policy, that can effectively transmit a flit is chosen to do so. A fair policy is used to prevent starvation.

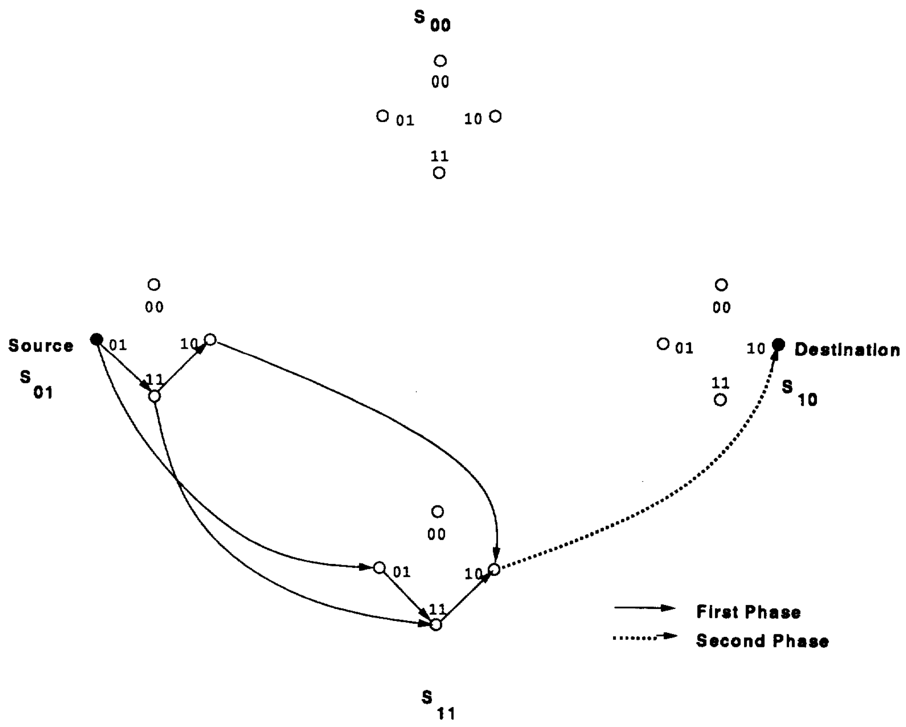


Fig. 8. The available paths from node (0, 1, 0, 1) to node (1, 0, 1, 0) using the *Basic Subcube* algorithm.

It is important to point out some implications of the previous definitions. First, it takes at least two cycles for a flit under no congestion to pass from an input buffer to another input buffer. In one cycle, it reaches an output buffer, and only in the next routing cycle is it transferred to the neighboring node. (Note that the flit cannot be transferred in the same cycle, because the node and the link cycles are assumed to occur simultaneously.) Second, under optimal conditions, a worm of length b needs $2b$ routing cycles to complete its injection process (i.e., the time between its head being in the injection buffer and the tail leaving it).

In the simulations, both node and link cycles are synchronized throughout the network in the sense that all of the nodes are involved in the node cycle, and afterward all of the links perform the link cycle. However, the algorithms can be implemented in an asynchronous model if desired.

C. Injection Model

Network simulations for parallel computers usually involve two kinds of models: the *static* injection model, in which every node has a fixed number of messages to inject, and the *dynamic* injection model, in which each node attempts to inject at arbitrary moments following some probability distribution. In the first case, the routing begins when each node injects its first message, and ends when the last message arrives at its destination. In the latter, the simulations are samples of an infinite process that must be stopped at some point.

In this paper, simulations for dynamic injection are reported. This model mimics the network activity of multiple-instruction, multiple-data (MIMD) computers; hence, it is more interesting than the static case. A node decides to inject a new message into the network with some probability λ. If the corresponding buffer is empty, the message is effectively injected. Otherwise, the message is discarded, and it is considered a failure. If λ is too large, the network may *saturate*; i.e., the behavior of the network becomes unstable, with the maximum latency of the messages growing without bound and the network rejecting a lot of messages.

The number of messages injected per cycle in each node, τ, is bounded by the characteristics of traffic patterns, the network, and the routing algorithm. Some bounds are known for τ [17], [29]. In this work, $\tau_{\max} = 1/(2b)$, where b is the worm length.[5] Although this optimal bound is far away from the achieved throughput, it is used only as a normalization factor common to all compared algorithms, thus expressing the throughput of the network in terms of its maximum estimated capacity.

Two parameters have been considered: **throughput** (*the number of messages delivered per cycle*) and **maximum latency** (*the largest latency experienced by a worm*).

D. Communication Patterns and Traffic Characteristics

The performance of the routing algorithms should be measured for different communication patterns and traffic characteristics. Two communication patterns will be used:

[5]In the routing model used for all the algorithms, the injection of a message into the network takes $2b$ cycles in the best case. If buffer capacity is increased to two flits or more, then pipelining in message transmission will be possible, and the injection of a message will take b cycles.

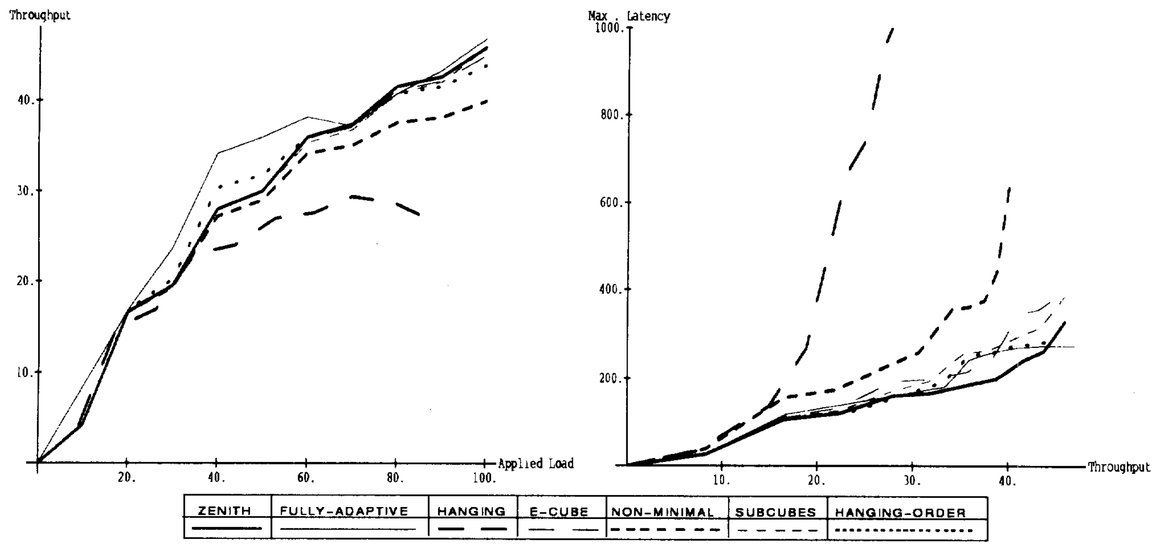


Fig. 9. Results for *Random Routing* with worms of length 10.

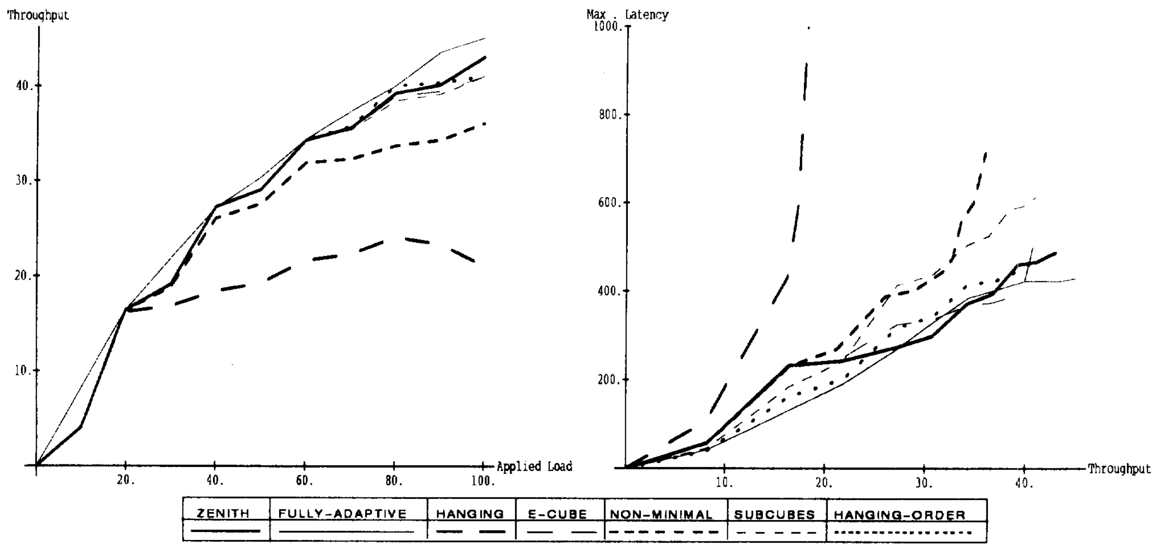


Fig. 10. Results for *Random Routing* with worms of length 20.

- **Random Routing:** Messages have random destinations. Several distributions are possible, from which two were chosen.
  a) *Uniform:* Destinations have a uniform distribution over the set of nodes. This distribution models the unstructured communication pattern that is present in many applications.
  b) *Leveled:* Each node p sends each message to a node that is randomly chosen, with uniform distribution, among all of the nodes with the same Hamming weight as p.
- **Fixed Permutations:** In this case, a permutation σ of the processors' indices is fixed in advance. A node p injects messages with destination $\sigma(p)$. In particular, the following permutations are simulated.
  a) *Complement:* All the bits in the binary address of the node are complemented.
  b) *Transpose:* The binary address of the node is split into halves, and these halves are swapped (if the dimension of the hypercube is odd, then the middle bit remains unchanged).

E. Simulation Results

All of the algorithms presented in Section V are simulated on a hypercube of 2^{10} nodes with the model shown in the previous sections. The patterns explained above are tried with several worm sizes: 5, 10, and 20.

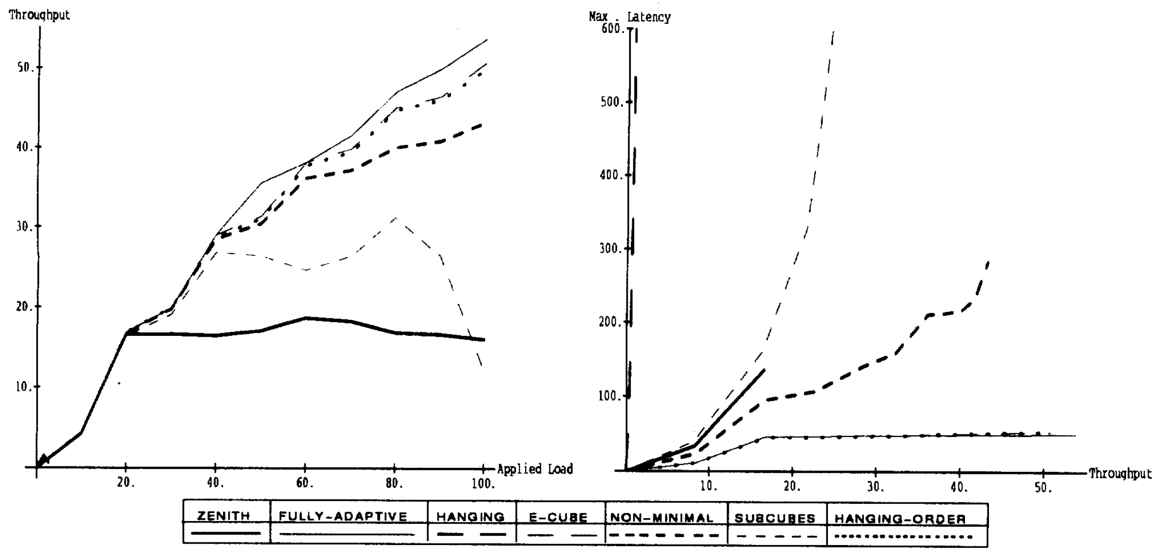


Fig. 11. Results for *Complement* with worms of length 10.

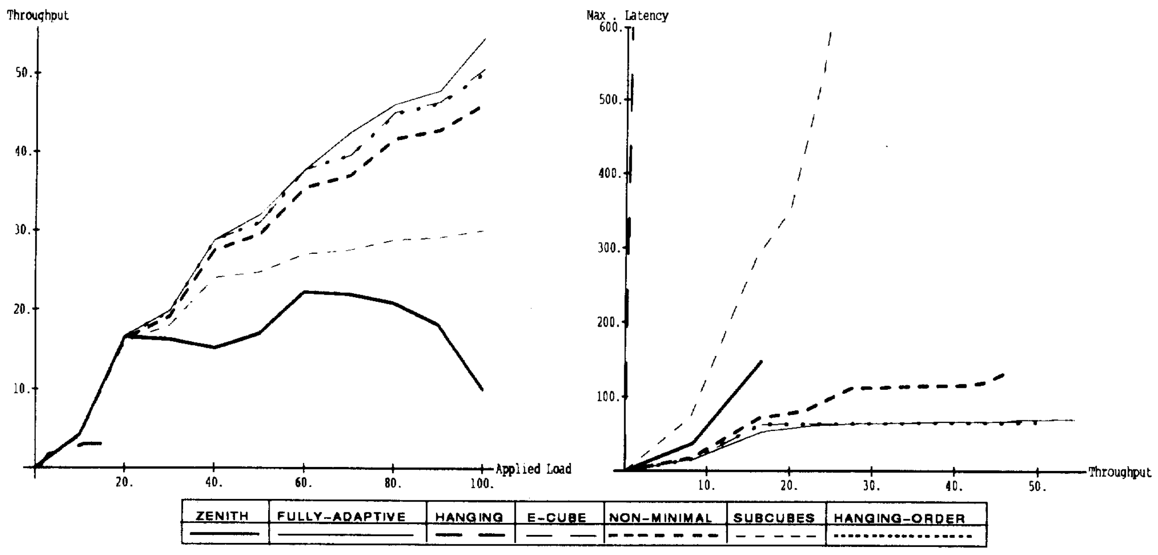


Fig. 12. Results for *Complement* with worms of length 20.

The results for the uniform random, complement, and transpose communication patterns are shown in Figs. 9 through 14. Two figures are shown for each communication pattern and for worms of length 10 and 20 flits. One plot shows the maximum latency reached when the network achieves a given throughput. The other plot shows the achieved throughput as a function of the applied load. Recall that throughput and injection rate are measured as a percentage of the theoretical bound, as explained in Section VI-C.

Both *Nonminimal* and *Fully Adaptive* show good results for all the communication patterns, but *Fully Adaptive* is better. *Nonminimal* presents always higher latency than the best for each case, because of the nonminimality of the technique. On the other hand, *Hanging* is the worst in all the cases.

It was observed that some of the algorithms displayed bad behavior under a certain traffic pattern. For example, *Dimension-Order*, and also *Hanging-Order*, behave very badly for the transpose permutation, but show a good performance for the complement permutation. On the other hand, the algorithms based on hanging the hypercube, namely, *Hanging, Zenith,* and *Basic Subcubes*, have their worst performance for the complement pattern. Note that in the case of *Hanging*, all of the messages pass through node $1 \cdots 1$ for this pattern. *Basic Subcubes* and *Zenith* also suffer, but *Basic Subcubes is*

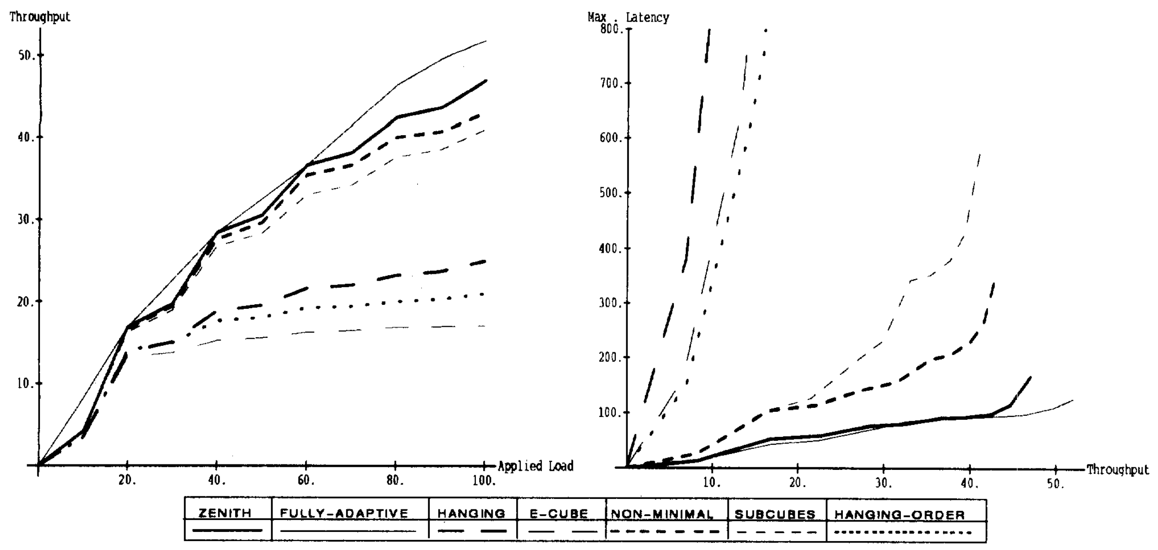


Fig. 13. Results for *Transpose* with worms of length 10.

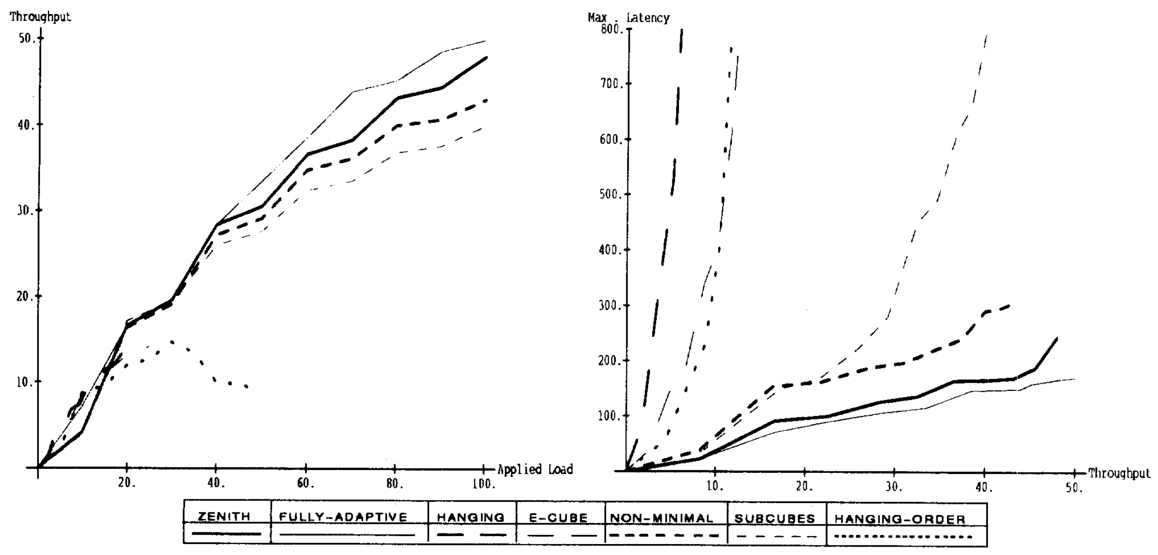


Fig. 14. Results for *Transpose* with worms of length 20.

slightly better. Furthermore, *Zenith*, and, to a lesser extent, *Basic Subcubes*, exhibit very good behavior for the transpose permutation.

The simulations for random routing, both uniform and leveled, show a similar behavior for all the algorithms, except for *Hanging* and *Nonminimal. Hanging* is the worst in latency and can sustain very small throughput relative to the others. *Nonminimal* sustains higher throughput than *Hanging*, but has higher latency than the other algorithms.

VII. Conclusion

The first part of this paper presented two new algorithms for wormhole routing in the hypercube network. Algorithm *Nonminimal* (see Section III) is adaptive and nonminimal. This algorithm uses eight virtual channels per physical link and requires small connectivity among the virtual channels on the same network node. Algorithm *Basic Subcubes* (see Section IV) is adaptive and minimal, and uses just two virtual channels per bidirectional physical link.

The second part of the paper reported an experimental comparison of the new techniques with other known algorithms from the literature. This wide range of experiments covers the state of the art for wormhole routing in the hypercube. Oblivious and adaptive, minimal and nonminimal, partially adaptive and fully adaptive algorithms were tried. A simple node model for all the algorithms was used. This model allows to compare the different strategies from an algorithmic standpoint.

Consider first those algorithms requiring fewer resources per node, and thus allowing for a simpler node design, as discussed in Section VI-A.[6] Among these algorithms, *Basic Subcubes* is the only one that can sustain at least 20% of the $\tau_{\max}$ in throughput for all of the patterns tried.

If more complex node models are allowed, *Zenith*, with three virtual channels per bidirectional link, is a good candidate for routing. It performs closer to the optimal in all the communication patterns except the complement where it can tolerate only up to 20% of $\tau_{\max}$. In that case, even *Basic Subcubes* is slightly better when the size of the worm increases.

Nonminimal performs well for all of the communication patterns, because of its potential ability of breaking the structure of the regular communication patterns. However, *Nonminimal* is slightly worse than the best algorithm in each case, because it is a nonminimal routing technique. Although the number of virtual channels per link is very large compared to the other algorithms, thus increasing the node complexity, one feature of the algorithm can result in a simple node design: A moderate connectivity is required between the virtual channels of the same routing node. More precisely, an input channel needs to reach only four output virtual channels.

Finally, *Fully Adaptive* appeared to be the best technique for all of the communication patterns and worm lengths tried. It requires fewer virtual channels per link than *Nonminimal*, which is the only other algorithm that did not show a bad-case communication pattern. *Fully Adaptive* makes a better use of the bandwidth of the hypercube network than the others, as a result of its being fully adaptive, and this comparison is fair, because all of the algorithms tested were assigned the same number of virtual channels per physical link.

APPENDIX

Theorem 1: The routing algorithm described in Section III is correct and deadlock-free.

Proof:

- *Correctness:* It suffices to show that at the end of phase i, dimensions $n-1$ through i have already been corrected. The proof is straightforward, using induction on i.
- *Deadlock-Freedom:* It suffices to show that each message goes through virtual channels following a strictly descending lexicographic ordering with respect to the virtual channels' subscripts. Initially, each virtual channel is injected in a virtual channel with prefix n. Then messages go through phases $n-1$ to 0, and during phase i, messages move through channels with prefix i. At most two channels are visited during each phase, and, if so, the two channels are visited following a strictly descending lexicographic ordering. Therefore, messages visit virtual channels in strictly descending lexicographic ordering of the subscripts, and so the routing algorithm is deadlock-free.

[6] All of the techniques requiring just one virtual channel per directed link can be used for circuit-switching models of message transmission.

Theorem 2: Injection channel $c_{n,0,p}$ has outdegree d ($d < n$), whereas channels $c_{i,1,j,p}$ and $c_{i,0,i,p}$ have outdegree $|dims(p, i-1)|+1$ and $|dims(p,i-1)|$, respectively, if $j < i$.

Proof:

- A message m at channel $c_{n,0,p}$ is starting its phase $n-1$. So, according to the routing algorithm, m will move to any of the channels in $\{c_{n-1,1,i_1,p_{(i_1)}}, \cdots, c_{n-1,1,i_d,p_{(i_d)}}\}$, where $\mathrm{dims}(p, n-1) = \{i_1, \cdots, i_d\}$.$\mathrm{dims}(p, n-1)$ is the same for every message. Consequently, $c_{n,0,p}$ has outdegree d.
- When a message m enters $c_{i,1,j,p}, j < i$, it is in the ith routing phase, performing a potentially derouting step. So, there are two cases.
  a) m does not have to correct the ith dimension. In this case, m has already finished phase i once it has entered channel $c_{i,1,j,p}$. So, m has to start phase $i-1$, and this means entering any of $|\mathrm{dims}(p,i-1)|$ virtual channels, namely, $c_{i-1,1,i_1,p^{(i_1)}}, \cdots, c_{i-1,1,i_k,p^{(i_k)}}$, where $\mathrm{dims}(p,i-1) = \{i_1, i_2, \cdots, i_k\}$, which is fixed for every message entering $c_{i,1,j,p}$.
  b) m has to correct the ith dimension. In this case, m has to enter channel $c_{i,0,i,p_{(i)}}$.

Therefore, it follows that the outdegree of channel $c_{i,1,j,p}$ is $|\mathrm{dims}(p,i-1)|+1$.

- When a message m enters $c_{i,0,i,p}$, it is in the ith routing phase, correcting dimension i. So, after entering this channel, m has to enter phase $i-1$. So, it has to go to any of the channels in $\{c_{i-1,1,i_1,p^{(i_1)}}, \cdots, c_{i-1,1,i_k,p^{(i_k)}}\}$, where $\mathrm{dims}(p,i-1) = \{i_1, \cdots, i_k\}$. So, $c_{i,0,i,p}$ has outdegree $|\mathrm{dims}(p,i-1)|$.

The virtual channels corresponding to phases $d-1$ through 0 are a special case, because, in general, they have less outdegree. So, every virtual channel has outdegree less than or equal to $d+1$.

ACKNOWLEDGMENT

The authors are indebted to the very useful comments and suggestions provided by three reviewers.

REFERENCES

[1] M. Arango, H. Badr, and D. Gelenter, "Staged circuit switching," *IEEE Trans. Comput.*, vol. C-34, pp. 174–180, Feb. 1985.
[2] S. Borkar *et al.*, "iWarp: An integrated solution to high-speed parallel computing," in *Proc. Supercomputing* 1988.
[3] H. Badr, D. Gelenter, and S. Podar, "An adaptive communications protocol for network computer," *Performance Evaluation*, vol. 6, pp. 35–51, Mar. 1986.
[4] P. Berman, L. Gravano, G. D. Pifarré, and J. L. C. Sanz, "Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks," in *Proc. 4th Symp. Parallel Algorithms and Architectures (SPAA)*, 1992.
[5] Y. Birk, P. B. Gibbons, D. Soroker, and J. L. C. Sanz, "A simple mechanism for efficient barrier synchronization in MIMD machines," Tech. Rep. RJ-7078-(67141), Dept. of Comput. Sci., IBM Almaden Research Center, CA, USA, Oct. 1989.

[6] G.-M. Chiu, S. Chalasani, and C.S. Raghavendra, "Flexible, fault-tolerant routing criteria for circuit-switched hypercubes," in *11th Int. Conf. Distrib. Computing Syst.*, 1991.

[7] F. Chong, E. Egozy, A. DeHon, and T. Knight, "Multipath fault tolerance in multistage interconnection networks," Transit Note 48, Massachusetts Inst. of Technol., Cambridge, MA, USA, June 1991.

[8] R. Cypher and L. Gravano, "Adaptive deadlock-free packet routing in torus networks with minimal storage," in *Proc. ICPP 92*, vol. III, pp. 204–211, 1992.

[9] ——, "Requirements for deadlock-free, adaptive packet routing," in *Proc. PODC 92*, 1992.

[10] Thinking Machine Corp., "Connection machine model CM-2," Tech. Summary, May 1989.

[11] W.J. Dally and H. Aoki, "Adaptive routing using virtual channels," Tech. Rep., MIT, Cambridge, MA, USA, 1990.

[12] W.J. Dally, "Virtual-channel flow control," in *17th Ann. Int. Symp. Comput. Architecture* , May 1990.

[13] S.P. Dandamundi, "A performance comparison of routing algorithms for hierarchical hypercube multicomputer networks," in *Proc. 1990 Int. Conf. Parallel Processing*, vol. 1, pp. 281–285, Aug. 1990.

[14] W.J. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. C-36, pp. 547–553, May 1987.

[15] J. Duato, "Deadlock-free adaptive routing algorithms for multicomputers: Evaluation of a new algorithm," in *Proc. 3rd IEEE Symp. Parallel Distrib. Processing*, 1991.

[16] M.L. Fulgham, R. Cypher, and J.L.C. Sanz, "A comparison of SIMD hypercube routing strategies," in *Proc. ICPP'91 Int. Conf. Parallel Processing* , vol. III, pp. 236–243, 1991.

[17] S.A. Felperin, L. Gravano, G.D. Pifarré, and J.L.C. Sanz, "Fully-adaptive routing: Packet switching performance and wormhole algorithms," in *Supercomputing*, pp. 654–663, 1991.

[18] ——, "Routing techniques for massively parallel communication," *Proc. IEEE* (special issue on massively parallel computers), vol. 79, pp. 488–503, Apr. 1991.

[19] D. Gelernter, "A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks," *IEEE Trans. Comput.*, vol. C-30, pp. 709–715, Oct. 1981.

[20] L. Gravano, G.D. Pifarré, G. Denicolay, and J.L.C. Sanz, "Adaptive deadlock-free wormhole routing in hypercubes," in *Proc. 6th Int. Parallel Processing Symp.*, 1992.

[21] L. Gravano, G.D. Pifarré, S.A. Felperin, and J.L.C. Sanz, "Adaptive deadlock-free wormhole routing with all minimal paths," Tech. Rep. TR:91-21, CRAAG, IBM Argentina, Buenos Aires, Argentina, 1991.

[22] L. Gravano, G.D. Pifarré, and J.L.C. Sanz, "Adaptive wormhole routing in tori and hypercubes," Tech. Rep. TR-91-10, CRAAG, IBM Argentina, Buenos Aires, Argentina, Mar. 1991.

[23] K.D. Gunther, "Prevention of deadlocks in packet-switched data transport system," *IEEE Trans. Commun.*, vol. COM-29, no. 4, Apr. 1981.

[24] D. Hillis, *The Connection Machine.* Cambridge, MA: MIT, 1985.

[25] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Comput. Networks*, vol. 3, pp. 267–286, 1979.

[26] S. Konstantinidou, "Adaptive, minimal routing in hypercubes," in *6th MIT Conf. Advanced Res. VLSI*, 1990, pp. 139–153.

[27] C.P. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Trans. Comput.* , vol. C-32, pp. 1091–1098, Dec. 1983.

[28] S. Konstantinidou and L. Snyder, "The Chaos router: A practical application of randomization in network routing," in *2nd Ann. ACM SPAA*, 1990, pp. 21–30.

[29] ——, "Chaos router: Architecture and performance," in *18th Int. Symp. Comput. Architecture*, 1991, pp. 212–221.

[30] D.H. Linder and J.C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Trans. Comput.*, vol. 40, pp. 2–12, Jan. 1991.

[31] D. Lenoski, J. Landon, K. Gharachorloo, W. Weber, A. Goopta, and J. Hennessy, "Overview and status of the Stanford Dash multiprocessor," in *Int. Symp. Shared Memory Multiprocessing* , Tokyo, Japan, Apr. 1991.

[32] T. Leighton and B. Maggs, "Expanders might be practical: Fast algorithms for routing around faults on multibutterflies," in *30th Ann. Symp. Foundations of Comput. Sci.*, 1989, pp. 384–389.

[33] T. Leighton, B. Maggs, and S. Rao, "Universal packet routing algorithms," *Proc. 29th IEEE Symp. Foundations Comput. Sci.*, 1988, pp. 256–269.

[34] P.M. Merlin and P.J. Schweitzer, "Deadlock avoidance in store-and-foreward networks, part I: Store-and-forward deadlock," *IEEE Trans. Commun.*, vol. 28, no. 3, pp. 345–354, Mar. 1980.

[35] L.M. Ni and P.K. McKinley, "A survey of routing techniques in wormhole networks," Tech. Rep. MSU-CPS-ACS-46, Dept. of Comput. Sci., Michigan State Univ., USA, Oct. 1991.

[36] J.Y. Ngai and C.L. Seitz, "A framework for adaptive routing," Tech. Rep. 5246:TR:87, Comput. Sci. Dept., California Inst. of Technol., USA, 1987.

[37] G.D. Pifarré, L. Gravano, S.A. Felperin, and J.L.C. Sanz, "Fully adaptive minimal deadlock-free packet routing in hypercubes, meshes, and other networks," in *Proc. 3rd Ann. ACM Symp. Parallel Algorithms and Architectures*, 1991.

[38] N. Pippenger, "Parallel communication with limited buffers," in *Foundations of Comput. Sci.*, pp. 127–136, 1984.

[39] A.G. Ranade, "How to emulate shared memory," in *Foundations of Comput. Sci.*, pp. 185–194, 1985.

[40] A.G. Ranade, S.N. Bhat, and S.L. Johnson, "The Fluent abstract machine," in J. Allen and F.T. Leighton, Eds., *5th MIT Conf. Advanced Res. in VLSI*, 1988, pp. 71–93.

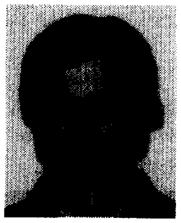
[41] E. Upfal, "An O(log N) deterministic packet routing scheme," in *21st Ann. ACM-SIGACT Symp. Theory of Computing*, May 1989.

[42] L.G. Valiant, "General purpose parallel architectures," in J. van Leeuwen, Ed., *Handbook of Theoretical Computer Science.* Amsterdam, Netherlands: North-Holland, 1988.

G.D. Pifarré received the B.S. degree in computer science from the Escuela Superior Latino-Americana de Informatica (ESLAI) in 1991.

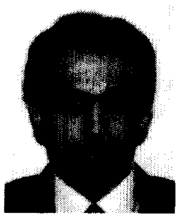
In 1990, he joined the Computer Research and Advanced Applications Group at IBM Argentina as a Researcher. He has been a Student Visitor to the IBM Almaden Research Center, CA, USA, three times. He is now a Professor and doctoral student at the Department of Computer Science, Universidad de Buenos Aires, Argentina. His current areas of research interest include multicomputers, multiprocessors, routing algorithms, parallel processing, and computer architectures.

Mr. Pifarré is the recipient of a research scholarship from the Universidad de Buenos Aires, as well as a scholarship from ESLAI.



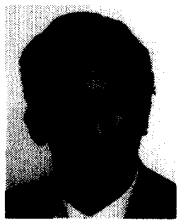
L. Gravano received the B.S. degree in computer science from the Escuela Superior Latino-Americana de Informatica (ESLAI) in 1991, and the M.S. degree in computer science from Stanford University, Stanford, CA, USA, in 1994.

From 1990 until 1992, he was a Researcher at the Computer Research and Advanced Applications Group at IBM Argentina. He has been a Student Visitor to the IBM Almaden Research Center, CA, USA, three times. He is now a doctoral student at the Computer Science Department, Stanford University, Stanford, CA, USA. His current areas of research interest include databases and parallel computers.



G. Denicolay received the B.S. degree in computer science from the Escuela Superior Latino Americano de Informática (ESLAI) in 1992.

During 1991, he was a Researcher at the Computer Research and Advanced Applications Group at IBM Argentina. He is now working on financial business. His current areas of research interest include software engineering, compiler design, and operating systems.



J. L. C. Sanz (M'82–SM'86–F'91) received the M.S. degree in computer science, the M.S. degree in mathematics, and the Ph.D. degree in applied mathematics from the Universidad de Buenos Aires, Argentina, in 1977, 1978, and 1981, respectively.

Since 1993, he has been a Professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign, USA. He is also affiliated with the Coordinated Science Laboratory there, and is also Manager of the Advanced Applications and Innovative Technologies Group at IBM Argentina, Buenos Aires, Argentina. He was an Instructor with the Department of Mathematics, Universidad de Buenos Aires, Argentina, during 1978 through 1980, and conducted research as a scholar member of the National Council of Scientific and Technical Research of Argentina for four years. .He was a recipient of many scholarships and was a member of the Argentinian Institute of Mathematics. He was a Visiting Scientist at the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, USA, during 1981–82. He was a Visiting Assistant Professor in the Department of Electrical Engineering and the Coordinated Science Laboratory in 1983. During 1983, he was a member of the summer visiting faculty at the Department of Computer Science at the IBM Research Laboratory in San Jose, CA, USA. From 1984 to 1993, he was with the Department of Computer Science, IBM Research Laboratory, San Jose, CA, USA, as a Research Staff Member. He conducted work on industrial machine vision, parallel computing, and multidimensional signal processing. He was the Technical Manager of the machine vision group during 1985–86. He has also been an Adjunct Associate Professor at the University of California at Davis, USA, where he conducted research as the Associate Director of the Computer Vision Research Laboratory until 1988. He has served as a consultant for several companies in the USA.

Dr. Sanz is a member of ACM. In 1986, he received the IEEE Acoustics, Speech, and Signal Processing Society's Paper Award for a publication in IEEE TRANSACTIONS ON ACOUSTICS, SPEECH,, AND SIGNAL PROCESSING. He is a Committee Member of the Multidimensional Signal Processing Group of the IEEE Acoustics, Speech, and Signal Processing Society. He was an Associate Editor of IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING from August 1987 until August 1989. He was the Editor of IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE special issues on industrial machine vision and computer vision technology in 1988. Also, he is an Editor-in-Chief of *Machine Vision and Applications: An International Journal*, and is author of the book *Radon and Projection Transform-Based Computer Vision* (Springer-Verlag 1988). He is the Editor of the book *Advances in Machine Vision* (Springer-Verlag). He is also a coauthor of the book *Massively Parallel Computing: Theory, Algorithms, Applications, and Technology* (Springer-Verlag 1991). He has been Chair and Organizer of the 1988 IEEE Workshop on Machine Vision, held in Ann Arbor, MI, USA. He has been the Chair and Organizer of the IBM Almaden/National Science Foundation Workshop on Opportunities and Constraints of Parallel Computing. He was the Program Committee Chair of the VI IEEE Acoustics, Speech, and Signal Processing Workshop on Multidimensional Signal Processing. He was the Program Committee Chair of the Computer Architecture Chapter at the 1990 International Conference on Pattern Recognition. He is Chair of the Industrial Machine Vision Chapter of the International Association of Pattern Recognition, and is a member of the Architecture Chapter of that organization.