Adaptive Deadlock- and Livelock-Free Routing With all Minimal Paths in Torus Networks

Luis Gravano, Gustavo D. Pifarré, Pablo E. Berman, and Jorge L. C. Sanz, *Fellow, IEEE*

Abstract—**This paper consists of two parts. In the first part, two new algorithms for deadlock- and livelock-free wormhole routing in the torus network are presented.**

The first algorithm, called *-*Channels*, is for the n-*dimensional* torus network. This technique is *fully-adaptive minimal*, that is, all paths with a minimal number of hops from source to destination are available for routing, and needs only five virtual channels per bidirectional link, the lowest channel requirement known in the literature for fully-adaptive minimal *worm-hole* routing. In addition, this result also yields the lowest buffer requirement known in the literature for packet-switched fully-adaptive minimal routing. The second algorithm, called 4-*Classes*, is for the *bidimensional* torus network. This technique is fully-adaptive minimal and requires only eight virtual channels per bidirectional link. Also, it allows for a highly parallel implementation of its associated routing node.

In the second part of this paper, four worm-hole routing techniques for the two-dimensional torus are experimentally evaluated using a dynamic message injection model and different traffic patterns and message lengths.

Index Terms— **Adaptivity, deadlock-freedom, torus network, livelock-freedom, parallel communication, parallel computer, performance simulation, worm-hole 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 [11], [43], [42], [46], [34], [47], [28], [38], [22], [5], [1], [33], [27] .

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 150–300 bits). In packet-switching routing, the messages are of constant (and small) size, and they are stored completely in every node they visit. In [16], a survey of some packet routing algorithms has been presented. In [14], simulation results have been shown comparing a number of different oblivious packet routing schemes on the hypercube. In worm-hole routing [11], 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 worm-hole methods, see [37]. In between packet-routing and worm-hole lie some hybrid approaches. In these methods [25], a message is routed by using a worm-hole 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], [41]. In other algorithms, deadlock freedom is guaranteed in a deterministic sense [35], [18]. 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 [11], [43], [42], [26], [3], and [21], [36], among others. This idea results in the 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 alternative paths toward their destinations according to traffic congestion in the nodes of the network. Recent simulations results for packet switching on the two-dimensional mesh [15] have shown that adaptivity improves the performance of dynamic injection routing when compared to oblivious methods. Also, the same conclusion was obtained by [8] for k-ary n-cubes. 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 messages depend on the traffic congestion found in the nodes of the network. 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 [46] 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.

New algorithms for deadlock-free worm-hole routing have been reported in [32], [8], [9], [19], [20], and [13]. Recently,

Manuscript received November 9, 1992; revised March 16, 1994.

L. Gravano is with the Computer Science Department, Stanford University, Stanford, CA 94305–2140 USA; e-mail: gravano@cs.stanford.edu.

G. D. Pifarré is with the Departamento de Computación, Fac. de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina; e-mail: pifarre@buevm2.vnet.ibm.com. He is also with the Advances Solutions and Innovative Technologies Department, IBM Argentina, Ing. E. Butty 275, 1300 Buenos Aires, Argentina.

P. E. Berman is with ESLAI, Escuela Superior Latino Americana de Informática, CC 3193, (1000) Buenos Aires, Argentina.

J. L. C. Sanz is with the Coordinated Science Laboratory and the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA; e-mail: sanz@where.csl.uiuc.edu. He is also with the Advances Solutions and Innovative Technologies Department, IBM Argentina, Ing. E. Butty 275, 1300 Buenos Aires, Argentina.

IEEE Log Number 9405075.

some mathematical analyses have been reported on the performance of worm-hole oblivious algorithms [44]. The algorithms in [32] are for k-ary n-dimensional cubes and n-dimensional mesh-connected networks. These techniques need a number of virtual channels per link that increases exponentially with n: in a k-ary n-cube 2^{n-1} virtual networks are needed, each with $n+1$ copies of the network. In [8], a method for deadlock-free adaptive routing in k-ary n-dimensional cubes is presented, which can be used for worm-hole routing. The new technique is based on a dynamic view of the conditions under which deadlock may arise. Routing of messages is accomplished by enforcing certain priorities on the use of virtual channels potentially intervening in deadlock conditions. In [9], a technique based on the use of 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. Simulations for worm-hole routing on a multistage interconnection network are shown. In [20], fully-adaptive worm-hole 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 worm-hole algorithm for the hypercube and a very similar design methodology were independently presented in [13].

In Section III, a new algorithm for routing on toroidal networks is presented [20], [2]. The new technique, called *-*Channels*, is fully-adaptive, deadlock- and livelock-free, and requires a very moderate amount of resources in the routing nodes. The new method is presented for n-dimensional tori. *-*Channels* works for messages of unknown size, thus rendering new routing techniques for both *packet-switched* and *worm-hole* models. It requires 5 virtual channels per bidirectional link of an n-dimensional torus. This algorithm is the first one with these characteristics, i.e., fully-adaptive minimal and deterministically livelock- and deadlock-free, that requires a number of virtual channels per link that does not depend on n or the size of the network. Actually, the number of virtual channels can be made equal to 3 for one of the dimensions. Thus, the total number of virtual channels per node is $10(n-1)+6$ for an n-dimensional torus. This count compares very favorably with that of the techniques presented in [32], as explained above. An instance of *-*Channels* yields a fully-adaptive minimal deadlock-free worm-hole routing algorithm for the bidimensional torus using 3 virtual channels per bidirectional link associated with the X dimension, and 5 virtual channels per bidirectional link associated with the Y dimension.

In addition, *-*Channels* also yields the smallest number of buffers known in the literature for packet-switched fully-adaptive minimal routing. For example, a total of 16 buffers are needed in the node of a two-dimensional torus, and 26 buffers in a three-dimensional torus. Thus, *-*Channels* offers an appealing approach to practical implementations of packet-switched low-dimensional toroidal networks because of its reduced storage requirement, ensuring freedom from deadlock and livelock *deterministically* without any source of randomization, and yet allowing all minimal paths for routing. Another remarkable characteristic of *-*Channels* for the packet model is that *no central queue is necessary* to guarantee any of its properties. These properties make *-*Channels* a practical alternative to chaotic routing [4], [29] for two-dimensional and three-dimensional adaptive torus networks.

In Section IV, a new algorithm for worm-hole routing on bidimensional torus networks, dubbed 4-*Classes* [15], is presented. This new technique is fully-adaptive minimal, and free of deadlock and livelock in a deterministic sense, and requires only eight virtual channels per bidirectional link for its implementation.

Section V presents a comparison of worm-hole algorithms for the two-dimensional torus. Four algorithms are analyzed and compared: *-*Channels*, 4-*Classes*, *Linder-Harden's*, and *Oblivious*. Although 4-*Classes* uses more virtual channels than *-*Channels*, it allows a node model design with potentially more parallel logic than that in *-*Channels*. *Linder-Harden's* algorithm [32], is fully-adaptive minimal, but uses more virtual channels than *-*Channels* and 4-*Classes*. The *Oblivious* algorithm [11] is *oblivious* and *nonminimal*, that is, the path followed by a message is completely determined by its source and destination, and the number of hops of this path may be greater than that of a minimal path. A node model for each of the algorithms is presented. For a fair comparison of the four algorithms, the number of virtual channels used by the different techniques is equalized.

II. Definitions

Definition 2.1: An n-dimensional k-torus (sometimes referred to as k-ary n-cube) is a network with k^n nodes. The n dimensions of the n-dimensional torus will be referred to as $X_{n-1}, \cdots, X_0$. Each node of the torus will be denoted by a tuple $(x_{n-1}, \cdots, x_0)$, with $0 \leq x_i < k$ for all $0 \leq i < n$, and will be connected to nodes $(x_{n-1}, \cdots, x_{i+1}, (x_i + 1) \bmod k, x_{i-1}, \cdots, x_0)$, and $(x_{n-1}, \cdots, x_{i+1}, (x_i - 1) \bmod k, x_{i-1}, \cdots, x_0)$, for all $0 \leq i < n$.

The link connecting nodes $(x_{n-1}, \cdots, k-1, \cdots, x_0)$ and $(x_{n-1}, \cdots, 0, \cdots, x_0)$ along dimension X_i will be called a wrap-around link along dimension X_i.

The directed link $((x_{n-1}, \cdots, x_i, \cdots, x_0), (x_{n-1}, \cdots, (x_i + 1) \bmod k, \cdots, x_0))$ corresponds to dimension X_i, and will be referred to as having orientation X_i^+.

Analogously, link $((x_{n-1}, \cdots, x_i, \cdots, x_0), (x_{n-1}, \cdots, (x_i - 1) \bmod k, \cdots, x_0))$, corresponding to dimension X_i, will be referred to as having orientation X_i^-.

Dimensions X_1 and X_0 of a two-dimensional torus will often be referred to as X and Y, respectively.

III. Algorithm*-*Channels*

In this section, a fully-adaptive, minimal, deadlock-free, worm-hole routing algorithm for the n-dimensional k-torus will be presented. This routing algorithm requires five virtual channels per bidirectional link for its implementation[1], a number that does not depend on the size or dimension of the

[1] In fact, only three virtual channels are needed for each bidirectional link associated with the most significant of the dimensions of the torus network.

torus, and allows each message to choose *adaptively*, step by step, among *all* the minimal paths that take it to its destination. No minimal path is discarded in order to obtain freedom from deadlock. This algorithm will be referred to as **-Channels*.

The central idea in **-Channels* is simple. There are basically two types of virtual channels, as will be explained below: *star* channels, and *nonstar* channels. Messages will move through the star channels as when doing *dimension-order* oblivious routing [11]. The nonstar channels will be used when taking any of the transitions that would not be allowed by the dimension-order routing algorithm, thus obtaining full adaptivity while preserving freedom from deadlock.

Related routing functions presented in the literature have acyclic channel dependency graphs ($\mathcal{CDG}$). Although this property is sufficient to guarantee deadlock freedom, it is not necessary, and can be substantially relaxed: the channel dependency graph has to be *dynamically* acyclic [36]. Intuitively, the **-Channels* algorithm involves two subnetworks, one composed of star channels and one of nonstar channels. The star channels implement a complete oblivious subnetwork that acts as a "release valve" or "drain" for the subnetwork built from the nonstar channels. The nonstar subnetwork can deadlock at any time but the star-channel subnetwork acts as an "escape" from deadlock.

The key idea behind the new algorithm is adding adaptivity to a routing algorithm based on an acyclic $\mathcal{CDG}$. Starting with the acyclic $\mathcal{CDG}$, some transitions are added under several constraints. The restrictions ensure that a message has always a valid transition in the original $\mathcal{CDG}$ as a valid step toward its destination. This means that if a message can be routed along a new transition, it will still have the possibility of taking a transition in the original $\mathcal{CDG}$. Therefore, at any moment, every message has a path to its destination in the original $\mathcal{CDG}$. In other words, every message will be able to progress towards its target node through the underlying DAG. This technique was used elsewhere for building routing functions over several networks, but for packet routing only. In [40], 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 [40] have been used for the same routing model in n-dimensional torus networks by using three queues per node [6], and this result is optimal for the given model and network [7]. Unfortunately, the methodology of [40] and [6] does not apply to worm-hole routing.

Next, the new algorithm will be described. Consider the directed link:

$$((x_{n-1},\cdots,x_i,\cdots,x_0),(x_{n-1},\cdots,(x_i+1) \bmod k,\cdots,x_0)).$$

This link will have three virtual channels associated with it:

- $c^*_{i,+,0,(x_{n-1},\cdots,(x_i+1) \bmod k,\cdots,x_0)}$,
- $c^*_{i,+,1,(x_{n-1},\cdots,(x_i+1) \bmod k,\cdots,x_0)}$, and
- $c_{i,+,(x_{n-1},\cdots,(x_i+1) \bmod k,\cdots,x_0)}$.

Analogously, link:

$$((x_{n-1},\cdots,x_i,\cdots,x_0),(x_{n-1},\cdots,(x_i-1) \bmod k,\cdots,x_0))$$

will have three virtual channels associated:

- $c^*_{i,-,0,(x_{n-1},\cdots,(x_i-1) \bmod k,\cdots,x_0)}$,
- $c^*_{i,-,1,(x_{n-1},\cdots,(x_i-1) \bmod k,\cdots,x_0)}$, and
- $c_{i,-,(x_{n-1},\cdots,(x_i-1) \bmod k,\cdots,x_0)}$.

Channels:

- $c^*_{i,+,0,(x_{n-1},\cdots,(x_i+1) \bmod k,\cdots,x_0)}$,
- $c^*_{i,+,1,(x_{n-1},\cdots,(x_i+1) \bmod k,\cdots,x_0)}$,
- $c^*_{i,-,0,(x_{n-1},\cdots,(x_i-1) \bmod k,\cdots,x_0)}$, and
- $c^*_{i,-,1,(x_{n-1},\cdots,(x_i-1) \bmod k,\cdots,x_0)}$

will be referred to as *star* channels.

In both cases, the star channels will be used as when doing dimension-order routing [11]: messages will move through star channels with prefix $i,+,0$ while correcting dimension X_i following orientation X_i^+ before taking a wrap-around link along dimension X_i. After taking a wrap-around along dimension X_i following orientation X_i^+, messages will move through star channels with prefix $i,+,1$ when correcting dimension X_i.

Now, the routing function $\mathcal{R}$: *VirtualChannels* $\times$ *Nodes* $\rightarrow$ $\mathcal{P}$ (*VirtualChannels*) will be described, where $\mathcal{P}$ (*VirtualChannels*) is the powerset of the set of *VirtualChannels*. It is supposed that each node p has an *injection channel* i_p where p injects the new messages into the network[2]. Whenever a message arrives at a channel incident with its destination node, it is delivered and taken out of the network.

The routing function.[3]

$$\mathcal{R}(c_x,y)=\begin{cases} c^*_{i,+,0,(x_{n-1},\cdots,x_{i+1},x_i+1,x_{i-1},\cdots,x_0)} \\ \quad \text{if } x_j=y_j \forall j=n-1,\cdots,i+1 \text{ and} \\ \quad x_i\neq y_i \text{ and } x_i\neq k-1 \text{ and} \\ \quad \text{the message has not taken} \\ \quad \text{a wrap-around along} \\ \quad \text{dimension } X_i \text{ yet and} \\ \quad \text{dimension } X_i \text{ should be corrected} \\ \quad \text{following } X_i^+ \\ c^*_{i,+,1,(x_{n-1},\cdots,x_{i+1},0,x_{i-1},\cdots,x_0)} \\ \quad \text{if } x_j=y_j \ \forall j=n-1,\cdots,i+1 \text{ and} \\ \quad x_i\neq y_i \text{ and } x_i=k-1 \text{ and} \\ \quad \text{dimension } X_i \text{ should be corrected} \\ \quad \text{following } X_i^+ \\ c^*_{i,+,1,(x_{n-1},\cdots,x_{i+1},x_i+1,x_{i-1},\cdots,x_0)} \\ \quad \text{if } x_j=y_j \forall j=n-1,\cdots,i+1 \text{ and} \\ \quad x_i\neq y_i \text{ and the message has already} \\ \quad \text{taken a wrap-around} \\ \quad \text{along dimension } X_i \text{ and} \\ \quad \text{dimension } X_i \text{ should be corrected} \\ \quad \text{following } X_i^+ \\ c_{j,+,(x_{n-1},\cdots,x_{j+1},(x_j+1) \bmod k,x_{j-1},\cdots,x_0)} \\ \quad \text{if } x_j\neq y_j \\ \quad \text{and dimension } X_j \text{ should be corrected} \\ \quad \text{following } X_j^+;\ j\in\{n-1,\cdots,0\} \end{cases}$$

[2]This model, called *one-port architecture* [23], is not required for the correctness of the algorithm. Several injection buffers can be used, still preserving deadlock and livelock-freedom. However, in the torus network a single injection buffer is enough to define the theoretically achievable throughput.

[3]In fact, $\mathcal{R}(c_x,y)$ is the set of all the virtual channels above whose corresponding condition on the right hand size is true.

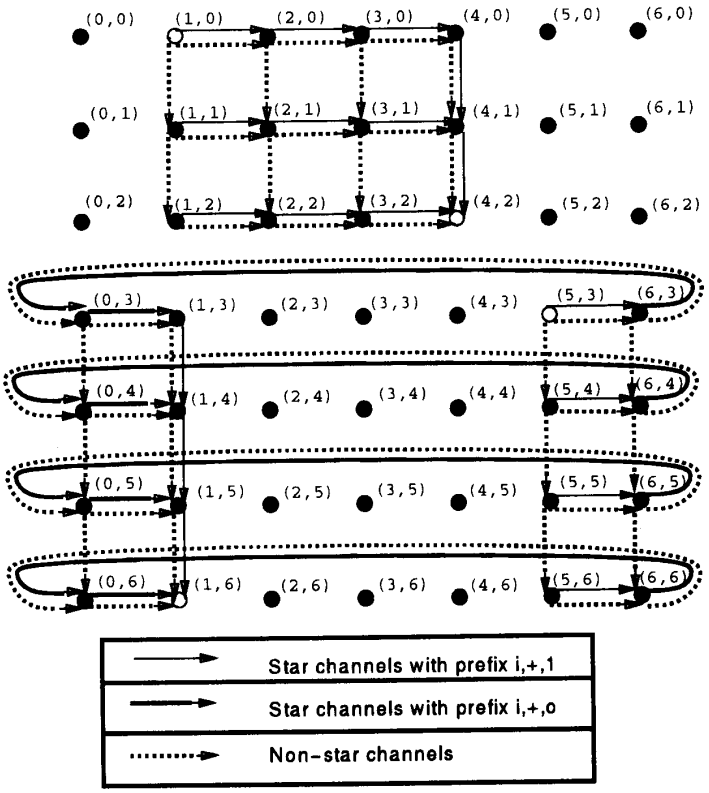


Fig. 1. The paths available in the two-dimensional 7-torus between node (1, 0) and (4, 2) and between node (5, 3) and node (1, 6) with the **-Channels* algorithm.

where c_x is a virtual channel incident to node $x = (x_{n-1}, \cdots, x_0)$ and $y = (y_{n-1}, \cdots, y_0)$ is the destination of the message. In the definition above, only the equations for correcting each dimension following the "+" orientation have been described.

A message will be allowed to correct *any* of the dimensions that need correction through nonstar channels. A message will be allowed to enter a star channel corresponding to dimension X_i only if X_i is the most significant dimension the message needs to correct, and only if the star channel corresponds to the message's having taken a wrap-around along that dimension or not, as explained above.

Consequently, the resulting worm-hole routing algorithm is fully-adaptive minimal. Moreover, assuming that virtual channels are assigned fairly, and that messages are of finite but arbitrary length, this routing algorithm is free of deadlock. Star channels play the most important role in proving this.

Fig. 1 shows two examples for a two-dimensional 7-torus. None of the minimal paths between nodes $(1,0)$ and $(4,2)$ contains a wrap-around link. So, all the star channels potentially used by a message going from node $(1,0)$ to node $(4,2)$ have prefix $i,+,1$, with $i = 1$ or $i = 0$. Note that any nonstar channel in a minimal path between the source node and the destination node can be taken at any moment. However, the star channels along the Y dimension can only be used after the X dimension has been totally corrected. For example, if the message is at node (2, 1), it can take any of the nonstar channels to nodes (3, 1) and (2, 2), namely $c_{1,+,(3,1)}$ or $c_{0,+,(2,2)}$, respectively. However, $c*_{1,+,1,(3,1)}$ is the only star channel the message can take, since it has not finished correcting dimension X yet. The same restrictions apply to the minimal paths between nodes (5, 3) and (1, 6). All of these paths contain a wrap-around link along the X dimension. Consequently, after the message reaches the "edge" of the torus (nodes (6, 3), (6, 4), (6, 5), and (6, 6)), it can only proceed through star channels with prefix $1,+,0$.

The following definition will be used in the remaining of this section. The *star-channel dependency graph* $\mathcal{CDG}^* = (C^*, D^*)$ is defined as follows:

- C^* consists of all the star channels.
- $(c_i^*, c_j^*) \in D^*$ if and only if there exists a message m such that $\mathcal{R}$ builds a path from m's source node's injection channel towards its destination that passes through c_i^* and afterwards through c_j^*.

The main difference between this dependency graph and the well-known $\mathcal{CDG}$ (see [11] and Definition 4.1) is as follows.

1) The nodes of the $\mathcal{CDG}^*$ graph are a subset of the channels of the network.
2) Although the nonstar channels are not nodes of the $\mathcal{CDG}^*$ graph, they participate in paths that create channel dependencies: two star channels that are not connected in the $\mathcal{CDG}$ for the *dimension-order* algorithm may now

be connected in the $\mathcal{CDG}^*$ because of paths containing nonstar channels. For example, consider star channels $c*_{1,+,1,(1,0)}$ and $c*_{1,+,1,(2,1)}$ in Fig. 1, which originate in nodes (0, 0) and (1, 1), respectively. These channels, which correspond to the X dimension and "+" orientation, are not linked in the $\mathcal{CDG}$ for the dimension-order algorithm. However, the nonstar channel $c_{0,+,(1,1)}$ from node (1, 0) to node (1, 1) establishes a dependency between them according to the definition above.

The set of all the virtual channels, star and nonstar, will often be referred to as C.

Lemma 3.1: The star-channel dependency graph $\mathcal{CDG}^*$ associated with $\mathcal{R}$ is acyclic.

Proof: The proof can be found in Appendix A .

Lemma 3.2: The routing algorithm that results from routing function $\mathcal{R}$ is free of deadlock.

Proof: The proof can be found in Appendix A.

From Lemma 3.1, Lemma 3.2, and $\mathcal{R}$, we have Theorem 3.3.

Theorem 3.3: The *-Channels algorithm yields a wormhole routing technique for any n-dimensional torus. The technique is fully-adaptive, minimal, free of deadlock and livelock in a deterministic sense, and can be implemented using 5 virtual channels per bidirectional physical link in all but one of the dimensions where only 3 channels suffice. Thus, the total number of virtual channels per node is $10(n-1)+6$.

In fact, the above presentation yields an algorithm with six virtual channels per physical link. However, only five virtual channels per bidirectional link are necessary. This is so because the star channels $c^*_{i,+,1,(x_{n-1},\cdots,x_i,\cdots,x_0)}$ with $x_i \geq \lfloor k/2 \rfloor$ are never used, because paths are minimal, and so, each message travels at most $\lfloor k/2 \rfloor$ steps along each dimension. Channels $c^*_{i,+,1,(x_{n-1},\cdots,x_i,\cdots,x_0)}$ with $x_i \geq \lfloor k/2 \rfloor$ would be used by messages that had taken a wrap-around along dimension X_i and orientation X_i^+, and moved at least $\lfloor k/2 \rfloor - 1$ steps along X_i^+ after the wrap-around. So, entering such a channel would involve traveling more than $\lfloor k/2 \rfloor$ steps along dimension X_i.

Analogously, channels $c^*_{i,-,1,(x_{n-1},\cdots,x_i,\cdots,x_0)}$ with $x_i < \lfloor k/2 \rfloor$ are never used.

So, only five virtual channel per bidirectional link are needed, because channels $c^*_{i,+,0,(x_{n-1},\cdots,0,\cdots,x_0)}$ and $c^*_{i,-,0,(x_{n-1},\cdots,k-1,\cdots,x_0)}$ are never used. Furthermore, dimension X_{n-1}, the most significant one, does not need nonstar channels: each bidirectional link corresponding to dimension X_{n-1} needs only three virtual channels.

As a result of this, the algorithm above instantiated to a two-dimensional torus needs three virtual channels for each bidirectional link associated with dimension X and five virtual channels for each bidirectional link associated with dimension Y. A simplified routing node model is depicted in Fig. 4 for the two-dimensional torus.

The above theorem was presented in the context of a wormhole model. In this model, messages serviced by the network are of unknown size. Therefore, the design of the routing node cannot be based on the assumption that messages will be completely stored in a node. If a packet implementation is desired, **-Channels* also yields the following important and immediate result:

Theorem 3.4: The **-Channels* algorithm yields a packet routing technique for any n-dimensional torus. The technique is fully-adaptive, minimal, free of deadlock and livelock in a deterministic sense, and can be implemented using 5 buffers, in each node, per bidirectional physical link in all but one of the dimensions where only 3 buffers per link suffice. Thus, the total number of buffers per node is $10(n-1)+6$. Furthermore, the packet routing technique does not require any central queue mechanism for ensuring the above properties.

As no central queue is mandatory, **-Channels* can be used for *combined routing*, i.e., both short fixed-size packets, and long messages of unknown size are handled by the same network and routing resources. While this type of combined routing can be naturally handled in multistage adaptive networks such as the multi-butterflies [30], fully-adaptive combined routing remained as a difficult goal for n-dimensional toroidal networks before **-Channels* was discovered. Combined routing is accomplished in **-Channels* by a "partial" virtual cut-through implementation, i.e. the storage of long messages would span more than one routing node while short messages may be buffered completely in one node. Practical studies are being carried out to experimentally determine the performance of this combined routing, and the potential improvements offered by a central queuing mechanism for enhanced routing of the fixed-size packets.

IV. Algorithm *4-Classes*

This algorithm divides all (source, destination) pairs into four classes and creates a virtual network for each class using all minimal paths in the network. [32] and [24] used similar ideas with a different choice for the classification of (source, destination) pairs. The partition chosen in *4-Classes* renders an algorithm that requires fewer resources and that has more potential parallelism in the node design than the techniques in [32] and [24].

Algorithm *4-Classes* [15] requires 8 virtual channels per bidirectional physical link, while allowing all the minimal paths between any pair of nodes s, d to be of potential use by those messages moving from s to d. Although *4-Classes* requires more virtual channels than **-Channels*, it has some important features: its routing node can be designed with a highly parallel implementation, as switching in a node can be decoupled into a set of eight independent 3×3 adaptive crossbars. This node model will be presented in Section V-B. A more detailed description of the algorithm follows.

If for a given node (x, y) dimension Y is fixed to y, then a cycle of length k is defined by moving along the X dimension. Cycles along dimension Y are defined in an analogous manner. Given a message with source node (x, y) and destination node (x', y'), a minimal path from (x, y) to (x', y') is built in such a way that the message will have to travel through at most $\lfloor k/2 \rfloor$ links along dimension X and through at most $\lfloor k/2 \rfloor$ along dimension Y. To find such minimal paths, the correct orientation along each dimension should be chosen. If k is odd, there is only one possible orientation of each dimension

following which all the minimal paths are built. If k is even, and a message m is $k/2$ steps away from its destination along dimension X, for example, then either orientation along the X dimension can be taken in order to follow minimal paths towards m's destination. This choice is independent for each dimension. Consider dimension X and the set of cycles associated with this dimension. Moving along dimension Y does not change the relative position within the different cycles associated with dimension X that will be visited. Similar considerations can be made for dimension Y. Therefore, the set of minimal paths between any pair of nodes is determined by a correct choice of the direction in which to change each of the dimensions.

Consequently, when a message is injected in the network, it will be classified into one, two, or four out of four classes according to the orientation in which each dimension should be corrected in order to follow a minimal path towards the destination. These four classes will be referred to as X^+Y^+, X^-Y^+, X^+Y^-, and X^-Y^-. This concept of classifying messages according to *classes* is well known [24] . If k is odd, then every message belongs to exactly one of these four classes. However, if k is even, then a message may belong to more than one class. For example, a message that is $k/2$ steps away from its destination along the X dimension will belong to both classes X^+Y and X^-Y, where Y is the orientation(s) following which dimension Y should be corrected.

In the following, the case of a two-dimensional k-torus with k odd will be analyzed.

Given a source node, a destination node, and an orientation for each dimension, a "submesh" is determined. All minimal paths between a source and a destination node are included in the submesh determined by the source, the destination and the orientation of each dimension chosen as explained above.

Now, the virtual network associated with each of the four classes into which messages are classified, viz X^+Y^+, X^-Y^+, X^+Y^-, X^-Y^-, will be presented. Each of these virtual networks will have a particular set of virtual channels assigned. The definition of each of these four virtual networks is almost identical. So, only the case X^+Y^+ will be described here (see Fig. 2).

A message will belong to class X^+Y^+ if the minimal paths from its source towards its destination are built by choosing X^+ and Y^+ as the orientation for correcting dimensions X and Y, respectively. Only this kind of messages will be routed through the virtual network X^+Y^+.[4]

Each physical link will have two virtual channels associated in the virtual network corresponding to class X^+Y^+. These two virtual channels will both use the underlying bidirectional physical link in the same direction. Every virtual channel is identified by a 3-uple: the first component of the 3-tuple is either a 0 or a 1[5]. The second component is the dimension associated with the underlying link, X or Y. Finally, the last component is the node in which a message arrives by taking this virtual channel. For example, there are two virtual channels from node (3, 2) to node (4, 2): $c^{++}_{0,X,(4,2)}$ and $c^{++}_{1,X,(4,2)}$.

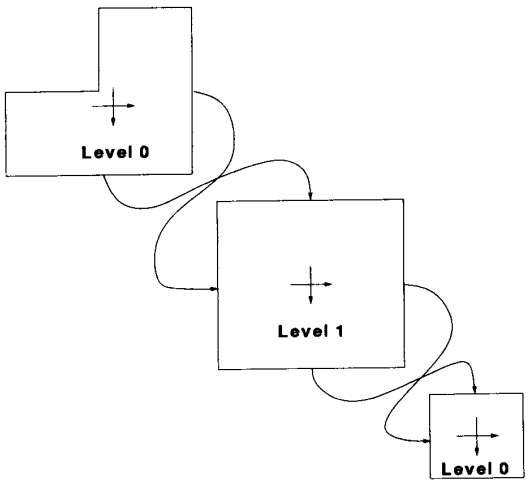


Fig. 2. Virtual Network X^+Y^+ for Algorithm *4-Classes*.

Next, the routing function for routing on the two-dimensional k-torus used for those messages belonging to class X^+Y^+ will be described. The routing function will be defined in terms of the virtual channels just mentioned. Let m be a message with source node (x, y) and destination node (x', y'). A distinction has to be made between those messages that will need to use the wrap-around links and those that will not. If m does not have to go through any wrap-around, then m will visit only channels with prefix 1 until it arrives at its target node. At each step, m will move along any of the dimensions that need correction. If $x > x'$ then m will use a wrap-around corresponding to dimension X and orientation X^+. If $y > y'$ then m will use a wrap-around corresponding to dimension Y and orientation Y^+. Again, the use of wrap-arounds or not along dimension X (resp. Y) depends exclusively on the values of x and x' (resp. y and y'). If either $x > x'$ or $y > y'$, then m will start moving through virtual channels with prefix 0, moving along any of the dimensions that need to be corrected, until it has to go through a wrap-around. It is important to notice that those virtual channels $c^{++}_{0,d,(z,w)}$ with $d = X$ or $d = Y$, and $0 \leq z, w \leq \lfloor k/2 \rfloor$ are not used during this first phase. This is so because if $x > x'$ and $x < \lfloor k/2 \rfloor$ then the path from x to x' following X^+ has length greater than $\lfloor k/2 \rfloor$ and so, the X^- orientation would have been chosen to correct dimension X. An identical argument follows for dimension Y.

After using a wrap-around, m will start moving through virtual channels with prefix 1. m will go on moving through channels with prefix 1 either until it arrives at its destination node, or until it needs to use another wrap-around, corresponding to the dimension that has not gone through a wrap-around yet. At this moment, m will start visiting channels with prefix 0 again until it arrives at its target node. It is important to note that the channels with prefix 0 that m will use in its last phase towards its destination are exactly those channels that have not been used yet, as pointed out above.

[4] In the case k even, messages that belong to more than one class may start visiting channels belonging to class X^+Y^+, as will be explained below.

[5] This first component will often be referred to as the *prefix* of the virtual channel in what follows.

The routing algorithm that has just been described allows each message to choose adaptively among *all* the minimal paths from its source node to its destination node in a two-dimensional k-torus with k odd. Furthermore, the routing algorithm is deadlock-free.

Next, the routing function for the messages in class X^+Y^+ will be described more formally. In the definition of the routing function it is supposed that every node (x, y) has an *injection channel* $i_{(x,y)}$ into which node (x, y) injects its messages.

A. Routing Function

Given a message at a virtual channel, and its destination, the following routing function defines the virtual channels the message can use as its next step to its destination. As soon as a message arrives in a channel that is incident with its destination node, the message is delivered and taken out of the network. The routing function below is for a two-dimensional k-torus with k odd. This routing function is modified later to cover the case k even by substituting extra statements for the $(*)$ placeholders:

$$\mathcal{R}(i_{(x,y)}, (x', y'))$$
$$= \begin{cases} c^{++}_{0,X,((x+1),y)} & \text{if } (x' < x \vee y' < y) \wedge \\ & \quad x \neq x' \wedge x \neq k-1 \quad (1) \\ c^{++}_{1,X,(0,y)} & \text{if } x \neq x' \wedge x = k-1 \quad (2) \\ c^{++}_{0,Y,(x,(y+1))} & \text{if } (x' < x \vee y' < y) \wedge \\ & \quad y \neq y' \wedge y \neq k-1 \quad (3) \\ c^{++}_{1,Y,(x,0)} & \text{if } y \neq y' \wedge y = k-1 \quad (4) \\ c^{++}_{1,X,((x+1),y)} & \text{if } x' > x \wedge y' \geq y \quad (5) \\ c^{++}_{1,Y,(x,(y+1))} & \text{if } x' \geq x \wedge y' > y \quad (6) \\ (*) & \end{cases}$$

$$\mathcal{R}(c^{++}_{0,d,(x,y)}, (x', y'))$$
$$= \begin{cases} c^{++}_{0,X,((x+1),y)} & \text{if } x \neq x' \wedge x \neq k-1 \quad (7) \\ c^{++}_{1,X,(0,y)} & \text{if } x \neq x' \wedge x = k-1 \quad (8) \\ c^{++}_{0,Y,(x,(y+1))} & \text{if } y \neq y' \wedge y \neq k-1 \quad (9) \\ c^{++}_{1,Y,(x,0)} & \text{if } y \neq y' \wedge y = k-1 \quad (10) \\ (*) & \end{cases}$$

$$\mathcal{R}(c^{++}_{1,d,(x,y)}, (x', y'))$$
$$= \begin{cases} c^{++}_{1,X,((x+1),y)} & \text{if } x \neq x' \wedge x \neq k-1 \quad (11) \\ c^{++}_{0,X,(0,y)} & \text{if } x \neq x' \wedge x = k-1 \quad (12) \\ c^{++}_{1,Y,(x,(y+1))} & \text{if } y \neq y' \wedge y \neq k-1 \quad (13) \\ c^{++}_{0,Y,(x,0)} & \text{if } y \neq y' \wedge y = k-1 \quad (14) \\ (*) & \end{cases}$$
$$\text{for } d = X, Y.$$

Next, the case k even will be considered. Suppose that a message m has been injected in node (x, y) and has node (x', y') as destination. Suppose that $|x - x'| = k/2$. Then, there are minimal paths from (x, y) to node (x', y') following both orientations X^+ and X^- for correcting dimension X. As soon as m moves a step following one of these two possible orientations, then only one of the two orientations will include all the minimal paths from the new current node to the destination of the message. Then, m should be allowed to switch to orientation X^+ or X^- while m is moving along dimension Y without moving along dimension X. As soon as m moves along dimension X, the orientation in which to correct dimension X becomes fixed.

To avoid deadlock, the following policy will be followed regarding messages belonging to more than one class. Notice that a message m can belong to 1, 2 or 4 classes according to its final destination.

Suppose that m belongs to 4 classes. Then, m is $k/2$ steps away from its destination node along both dimensions X and Y. So, while m is in its injection channel, it will be able to move along any of the two dimensions following any of the two orientations of these dimensions.

- If m moves along dimension X as its first step, then m will enter the set of virtual channels corresponding to class X^+Y^+ if the first step taken follows orientation $X+$, or to class X^-Y^+ if it follows orientation $X-$.
- If m moves along dimension Y as its first step, then m will enter the set of virtual channels corresponding to class X^+Y^+ if the first step taken follows orientation $Y+$, or to class X^+Y^- if it follows orientation $Y-$.

So, after this first step, all the minimal paths from the new node in which m resides towards m's final destination belong to two classes, as the orientation following which the dimension corresponding to m's first step was corrected becomes fixed after this first step.

Now, after this first movement, m is at a virtual channel incident with a node that is $k/2$ steps away from m's destination along the dimension that has not been touched yet. If this dimension is dimension X, then m is in class X^+Y^o where $o = +$ or $o = -$ and Y^o is the orientation that was taken in the first step (which was a step along dimension Y). Before the first step along dimension X, m will move through virtual channels belonging to class X^+Y^o. If the first step along dimension X is taken by following orientation X^+, then m will move through virtual channels belonging to class X^+Y^o until it gets delivered. Otherwise, m will switch to class X^-Y^o and will remain there. The case in which the first step m takes is along dimension X is analogous.

Now, if m belongs to 2 classes, X^+Y^o and X^-Y^o where $o = +$ or $o = -$, then m is $k/2$ steps away from its destination along dimension X, and less than $k/2$ steps away from its destination along dimension Y following orientation Y^o. Initially, m will start visiting virtual channels belonging to class X^+Y^o. If the first step m takes along dimension X is a step following orientation X^+, then m will belong to class X^+Y^o until it gets delivered. On the other hand, if the first step m takes along dimension X is a step following orientation X^-, then m will pass to class X^-Y^o, and will remain in that class. Dimension Y is treated analogously.

Therefore, a message in a virtual channel of class X^+Y^+ can switch to class X^+Y^-, or to class X^-Y^+ A message in class X^+Y^- can move to class X^-Y^-, and a message in class X^-Y^+ can move to class X^-Y^-. In every case, the policy just outlined must be followed to switch from a class to another.

Now, the routing function defined above will be modified to cope with messages switching class.

A message moving from class X^+Y^+ to class X^-Y^+, for example, will be treated as being injected as a message in class X^-Y^+ in the node where the passage between the classes takes place.

Therefore, the following is added to the definition of the routing function above, for a message initially in virtual network X^+Y^+:

$$(*) = \begin{cases} c^{-+}_{0,X,(x-1,y)} & \text{if } (x' > x \vee y' < y) \wedge \\ & |x - x'| = k/2 \wedge x \neq 0 \\ c^{-+}_{1,X,(k-1,y)} & \text{if } |x - x'| = k/2 \wedge x = 0 \\ c^{-+}_{1,X,((x-1),y)} & \text{if } x' < x \wedge y' \geq y \wedge \\ & |x - x'| = k/2 \wedge x \neq 0 \\ c^{+-}_{0,Y,(x,y-1)} & \text{if } (x' < x \vee y' > y) \wedge \\ & |y - y'| = k/2 \wedge y \neq 0 \\ c^{+-}_{1,Y,(x,k-1)} & \text{if } |y - y'| = k/2 \wedge y = 0 \\ c^{+-}_{1,Y,(x,(y-1))} & \text{if } x' \geq x \wedge y' < y \wedge \\ & |y - y'| = k/2 \wedge y \neq 0 \end{cases}$$

The other cases are handled completely analogously.

B. Correctness of the Routing Algorithm

In this section, it will be proved that the routing algorithm described in Section IV is correct and deadlock-free. When proving the deadlock freedom of the algorithm, the following definition will be used, which is very similar to the one presented in [11].

Definition 4.1: The *channel dependency graph (CDG)* corresponding to a set of virtual channels C and a routing function $\mathcal{R}$ is a directed graph such that its set of vertices is C and there exists an edge from c_i to c_j ($c_i, c_j \in C$) iff there exist an injection channel $s \in C$ and a node d of the underlying network such that $\mathcal{R}$ builds a route from s to d passing through both c_i and c_j, and $c_j \in \mathcal{R}(c_i, d)$.

Theorem 4.1: The *4-Classes* algorithm presented in this section is correct, deadlock-free, and yields a worm-hole fully-adaptive minimal routing technique for the two-dimensional torus using eight virtual channels per bidirectional link.

Proof: The proof can be found in Appendix B.

V. Node Models and Simulations

In this section, an experimental comparison of four different routing algorithms for the two-dimensional torus network will be presented. The four algorithms studied in this section are **-Channels* (see Section III), 4-*Classes* (see Section IV), *Linder-Harden's* [32], and *Oblivious* [11]. The first three algorithms are fully-adaptive minimal, whereas the last one is oblivious and nonminimal. These algorithms require 16, 32, 36, and 8 virtual channels per node, respectively, for their deadlock-free implementation. The comparison is carried out by equalizing the number of virtual channels used by the different routing techniques. This task is accomplished by adding *lanes* to the algorithms originally requiring fewer virtual channels per physical link. In the scope of this paper, a *lane* in a physical link is simply a virtual channel that plays an identical role to one of the original virtual channels associated with the same link [9]. Lanes are known to play a central role in enhancing the throughput of oblivious routing worm-hole algorithms [9]. Thus, more than one message can be simultaneously using the same virtual channel by occupying different lanes associated with the virtual channel. For example, in order to compare *Oblivious* with 4-*Classes*, 6 lanes will be added per physical link, thus matching the 32 virtual channels of 4-*Classes*. Furthermore, lanes are added in a way that crossbars grow with the total number of inputs.

The analysis of algorithms is carried out in two directions. First, routing node models are proposed for each technique. Second, simulations on their performance are shown for a network of 31×31 nodes. This machine size is feasible according to current VLSI and packaging technology. Moreover, it has a sufficient number of processors to be considered a massively parallel computer while still keeping the time required by the simulations manageable. The number of nodes in each dimension is odd with the purpose of having a slightly simpler routing function.

A. Oblivious and Linder-Harden's Algorithms

The oblivious, non minimal routing technique presented in [11] for routing in n-dimensional tori will be briefly described for $n = 2$. This algorithm has been designed with a unidirectional torus in mind, and will be referred to as *Oblivious*.

Messages are routed to their destination in two phases. In the first phase, they correct dimension X, and in the second phase they correct dimension Y. Number the nodes along each dimension from 0 to $k - 1$. In order to avoid deadlock, the physical channels are split in two virtual channels, a *high* and a *low* one. Restricting to each dimension, messages at a node numbered less than their destination node in that particular dimension are routed through the high channels, and messages at a node numbered higher than their destination node are routed through the low channels. If a message is injected at a node numbered less than its destination, it starts using high channels until it reaches node 0 in that dimension, and then it starts using low channels.

This oblivious algorithm requires two virtual channels per physical link to guarantee freedom from deadlock. Each link is only used in one of the directions.

In what follows, the fully-adaptive minimal routing algorithm for the n-dimensional torus presented in [32] will be briefly described for $n = 2$. This algorithm will be referred to as *Linder–Harden's* algorithm.

For this algorithm, the torus network is split into two virtual networks, dubbed Y^+ and Y^-. Each virtual network will have virtual channels in only one of the two possible directions in dimension Y, and in both directions in dimension X. For each of the two virtual networks, three "copies" of the network are required, because a message can use at most two wrap-around edges, and it needs to enter a new "copy" of the virtual network whenever it takes one wrap-around edge. Every message will be injected in one of the two virtual networks depending on its destination.

This technique requires 12 virtual channels per bidirectional link in dimension X and 6 virtual channels per bidirectional link in dimension Y for its deadlock-free implementation

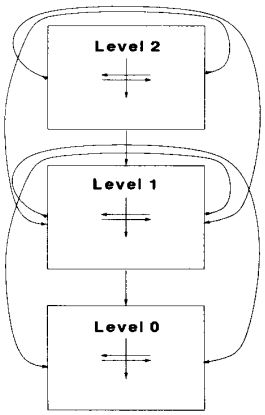


Fig. 3. Virtual network Y^+ for *Linder–Harden's* algorithm.

[32]. Fig. 3 shows a generic virtual interconnection network corresponding to Y^+.

B. Node Models

This section gives high-level descriptions of the routing nodes for each technique, using buffers and crossbars as building blocks. The exact time-delay introduced in each node is not calculated. For related work, see [10], [12], [45], [29], and [4].

Each virtual channel is implemented through an input and an output buffer. In addition, each node has an injection and a delivery buffer. The size of all of these buffers depends on whether the algorithms are intended to work for worm-hole or for packets of fixed and small size. In the experiments, worm-hole routing was tried. Therefore, the buffer size was 1 flit.

For some of the algorithms being compared, a partition of all of the buffers can be found such that the connections that the routing function performs are only between buffers that belong to the same set. Then, each set requires a crossbar that is independent of the others (see, for example, the node model of 4-*Classes* and that of *Linder-Harden's* algorithm, in Figs. 5 and 6, respectively). It is assumed that each crossbar can set one new connection from its inputs to its outputs per cycle [29], and this is executed independently for each of the other crossbars in the same node. Therefore, the more independent crossbars each node has, the more parallelism within each switch is achieved. *Oblivious* is allowed to make all possible connections (depending on availability of output buffer space) in its crossbar in each routing cycle, because this crossbar is oblivious, and therefore simpler than all of the others, which are adaptive. These adaptive crossbars are a mechanism to connect input frames to output frames adaptively which is functionally similar to a crossbar for oblivious routing, except that more than one useful output buffer, if available, may be used for potential progressing of a message toward its destination. An analogous input-to-output frame connectivity is used in other routers such as the Chaos [4]. The inputs to the crossbars and physical channel arbitration are handled with fairness to avoid starvation.

*Algorithm *-Channels:* In Fig. 4, a node model for **-Channels* is presented. Only 8 input and 8 output buffers per node are needed. These buffers, together with the injection and delivery ones, make a total of 9 the number of inputs and outputs of the adaptive crossbar, because adaptive connectivity between all of the inputs and all of the outputs of a node is required. Lanes are added to the virtual channels, as explained above. Therefore, each node will have an adaptive 17×17 crossbar.

Algorithm 4-Classes: This algorithm requires four virtual networks, each of which is split into two levels. Therefore, there are eight levels. Virtual channels corresponding to one level are independent of those of other levels. Thus, 8 adaptive 3×3-crossbars are required. Fig. 5 shows the node model for this algorithm. 2 input buffers, 2 output buffers and a 3×3-crossbar for each level, add up to 16 input buffers, 16 output buffers and 8 crossbars per node. Multiplexers connect the injection buffer to the 8 crossbars and the 8 crossbars to the delivery buffer.

Linder–Harden's Algorithm: Fig. 6 shows the node model for this algorithm. Each node has 36 buffers and 6 adaptive 4×4 crossbars, assigned as follows: 1 crossbar, 3 input buffers and 3 output buffers per independent level. Two multiplexers connect the injection buffer to the crossbars and the crossbars to the delivery buffer.

Algorithm Oblivious: For this routing algorithm, the set of buffers cannot be split into independent classes. Therefore, the node model has a single crossbar (see Fig. 7). Then, eight buffers and one 5×5-crossbar [10] are needed. 12 input and 12 output buffers are added as lanes to the virtual channels, as mentioned above. Adding buffers to implement lanes gives this algorithm some degree of adaptivity, since messages can take different lanes depending on traffic congestion.

C. Network Activity

This section describes the activity of the network assumed in the experiments that were performed. As latency will be measured in terms of *routing cycles*, a definition of the amount of work involved in one such cycle is needed. The routing cycle consists of two cycles: the *node cycle* and the *link cycle*, which take place simultaneously.

To keep the node cycle as short as the link cycle, only one new connection per independent crossbar will be set during a single node cycle [29].[6] Allowing more connections in one cycle would require some mechanism to handle arbitration, as two or more worm heads might want to enter the same output buffer. The overhead resulting from this arbitration would probably make a node cycle much longer than a link cycle. Moreover, if long worms are present in the network, making more than one connection per cycle might not be used frequently enough to offset the cost introduced by this function.

- *Node cycle*: During a single node cycle, each node performs the following sequence of tasks:
  `Connect;`

[6] Algorithm *Oblivious* is an exception to this, as explained above.

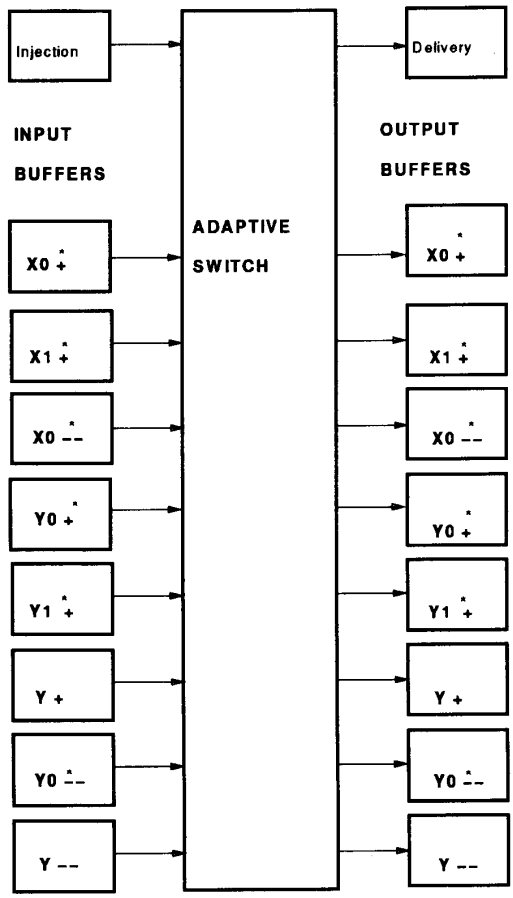


Fig. 4. Node model for algorithm **-Channels*.

`FreeBuffers;`
`Inject;`

— `Connect`: Each crossbar scans its inputs in a round-robin fashion and looks for a worm header. If a worm header is in some input and there is one idle output buffer which is a feasible connection for the worm header, the connection is performed. During a cycle, at most one head from the input buffers is allowed to set a connection to an output buffer, for each crossbar. A fair policy to prevent starvation is used. The injection buffer has the same priority as the rest of the buffers. The header of a message that has arrived at an input buffer of its destination node tries to set a connection with the delivery buffer of that node and then, it is eventually consumed.

— `FreeBuffers`: For all the connections already set in each crossbar, one flit is transmitted through the crossbar if possible, i.e., if the receiving output buffer is free. If the flit transmitted is the last flit of a worm, the connection is released in that routing cycle. Thus, once a buffer accepts the first flit of a message, it must accept the remainder of the message before accepting any flit of any other message.

— `Inject`: Each node injects a new flit if it needs to do so and its injection buffer is empty.

All the algorithms considered in this paper, except for **-Channels*, have acyclic channel dependency graphs associated with them. For these algorithms, to preserve freedom from deadlock, it suffices that the header of each worm check that the tail of a worm is occupying an output buffer for that header to set a connection with that output buffer. On the other hand, as the channel dependency graph associated with the algorithm presented in Section III is not acyclic, it is required that the header of a worm check that both the output buffer and its associated input buffer are free for the header to set the connection with the output buffer in the corresponding crossbar. Note that this extra condition to be tested requires that there be two empty buffers separating every pair of "consecutive" worms.

- *Link cycle:* During the link cycle, the flits of the messages move from the output buffers to the corresponding input buffers. For those algorithms with more than one output buffer per directed link, only one flit is transferred per cycle. These buffers are served in a round-robin fashion. The first output buffer, according to this round-robin policy, that can effectively transmit a flit is chosen to

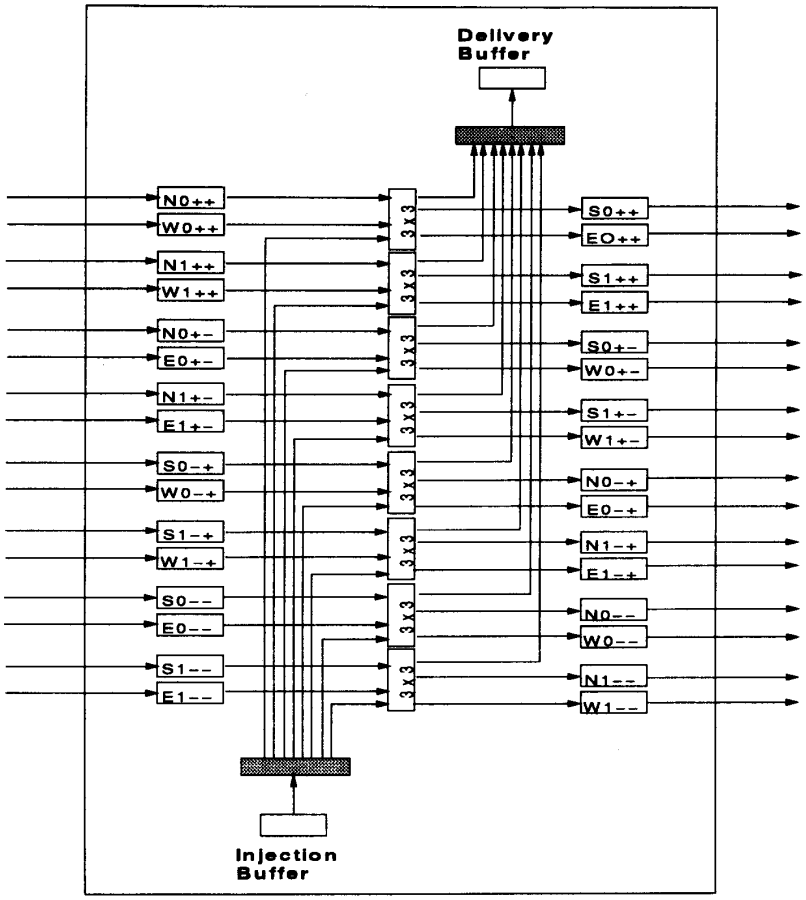


Fig. 5. Node model for Algorithm *4-Classes*.

do so. A fair policy is used so as to prevent starvation.

Oblivious uses each physical link in only one direction. To compare this algorithm to the others, the set of output buffers associated with each link was split into two equally sized subsets. It was assumed that each of these subsets had a "dedicated" link. Therefore, up to two flits can be transmitted from one node to its neighboring node along one of the dimensions, as long as the two flits come from output buffers belonging to different subsets.

From these definitions, some implications can be deduced. 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 the flit reaches an output buffer, and only in the next routing cycle is it transferred to the neighboring node. Second, under optimal conditions, a worm of length b needs $2b$ routing cycles to complete its injection process (i.e., the number of cycles from the cycle when the header gets to the injection buffer and the cycle when the tail leaves the injection buffer). Finally, as a buffer accepts a new flit only if it is empty, it takes at least $2b - 1$ cycles to route a message through a link.

In the simulations reported in this paper, both the node and link cycles are synchronized throughout the network, in the sense that all of the cycles take place at the same time. However, the model presented can be made asynchronous.

Injection Model: There are two injection models: the *static* injection model, in which every node has a fixed number of messages to inject, and the *continuous* or *dynamic* injection model, in which each node attempts to inject new messages 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 an infinite process that has to be stopped at some point.

In this paper, simulation results for *continuous injection* are reported. It is widespread knowledge that this type of injection models more accurately the network activity of MIMD machines. A node decides to inject a new message into the network at some rate λ. If a node decides to inject a message, and the corresponding injection buffer is empty, the new message is effectively injected. Otherwise, the message is discarded, and it is considered as a failure. If λ is too large, the network can become *saturated*, i.e., the latency of the messages grows without bound or the network rejects a lot of messages.

The number of messages injected per cycle in each node, τ, is bounded by the characteristics of traffic patterns, the

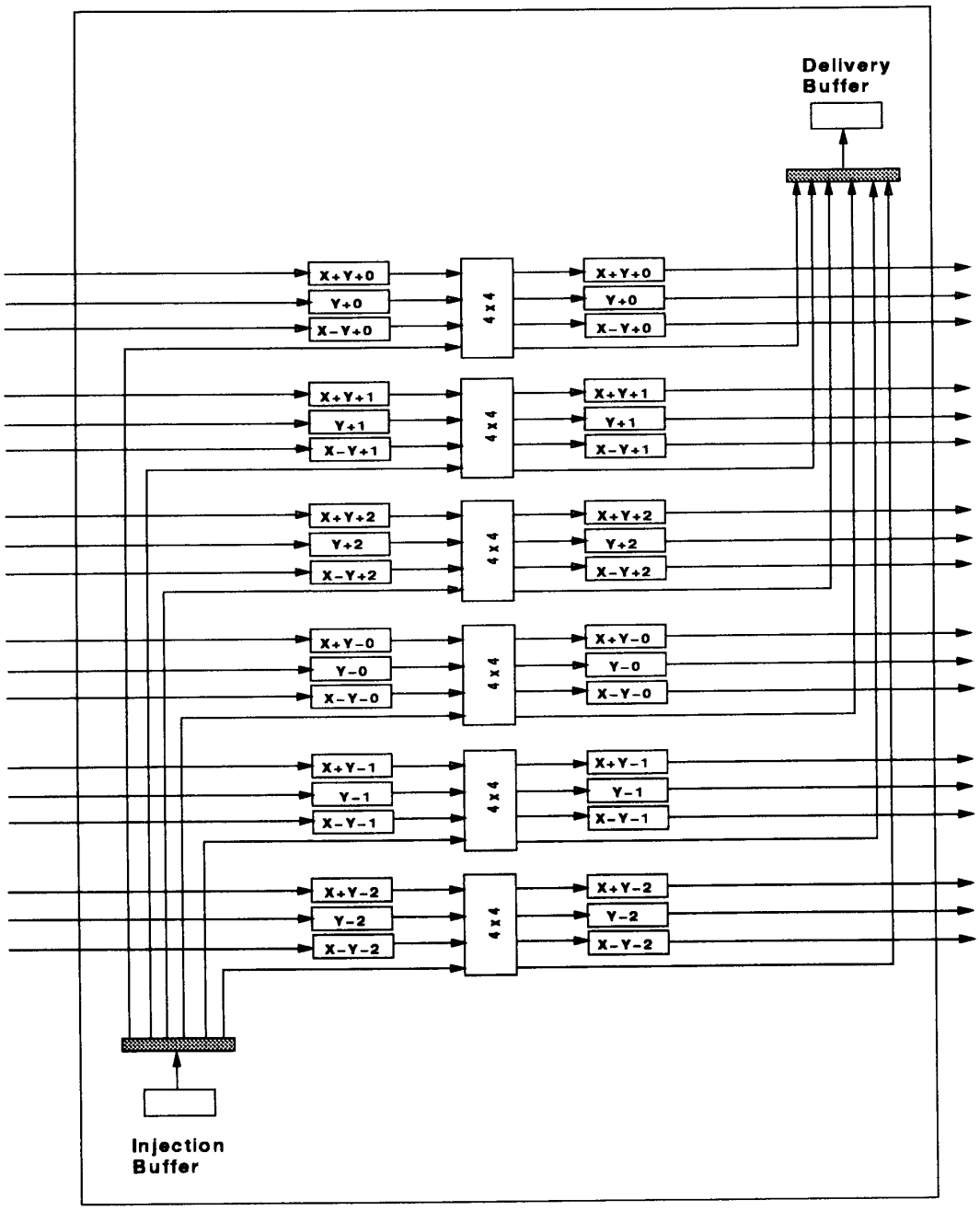


Fig. 6. Node model for *Linder–Harden's* Algorithm.

network, and the routing algorithm. Some bounds are know for τ [15], [29]. If N is the number of nodes in the network, B the bisection of the network [31] [39], c the proportion of messages that cross the bisection and T_b the minimum time required for a message to traverse a link, then:

$$\frac{N\tau c T_b}{2} < B. \tag{1}$$

For the two-dimensional k-torus, $B = 2k$ and $N = k^2$. In this simulations, $T_b = 2b - 1$, where b is the number of flits in each message. Thus,

$$\tau < \frac{4}{kc(2b-1)} = \tau_{\max}. \tag{2}$$

Traffic Characteristics: The performance of the routing algorithms was measured for different communication patterns.

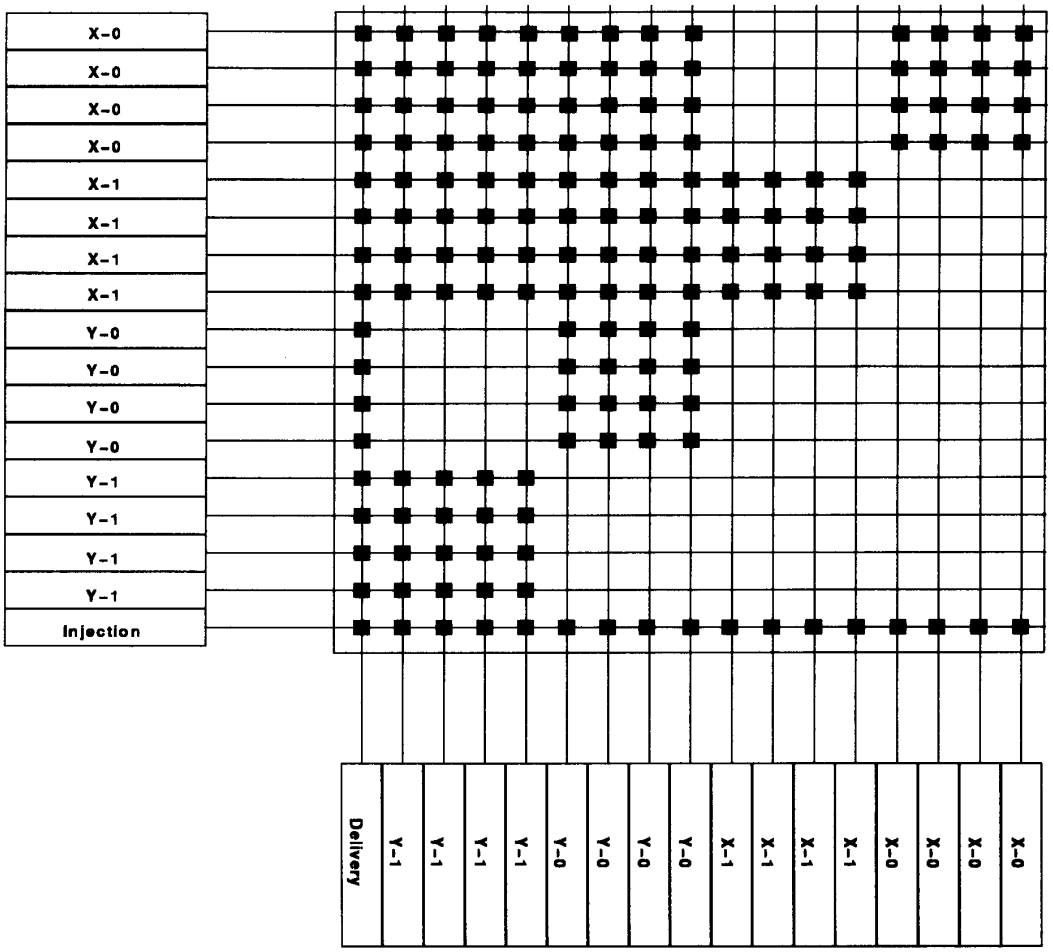


Fig. 7. Node model for Algorithm *Oblivious*.

- *Random Routing:* The destinations of the messages had a uniform distribution over the set of nodes. This models the unstructured communication pattern that is present in many applications. In this case, $c = \frac{1}{2}$.
- *Fixed Permutations:* In this case, a permutation σ of the processors' indices was fixed in advance. Node p injected all of its messages with destination $\sigma(p)$. In particular, the following permutation was simulated:
  - *Bit Reversal:* Node $((x_p \cdots x_0), (y_p \cdots y_0))$ sends all of its messages to node $((y_0 \cdots y_p), (x_0 \cdots x_p))$, where $p = \lceil \log_2(k) \rceil$ [8]. In this case, $c = 1/2$.

D. Simulation Results

Each routing algorithm was simulated according to the high-level node models sketched above. Several continuous injection simulations were tried. These experiments involved two different message sizes, namely worms of 15 and 31 flits, and two different traffic patterns: random traffic and bit-reversal traffic, on a two-dimensional torus with 961 nodes.

Figs. 8–10 show the mean latency as a function of the achieved throughput, for the four routing algorithms and the different traffic patterns and worm lengths. The results corresponding to bit-reversal traffic for 31-flit worms are similar to those for 15-flit worms.

From the experimental performance study, several conclusions can be drawn. First, **-Channels* showed better performance than *Oblivious* for all of the communication patterns and at all of the applied loads tried. This comparison is fair because *Oblivious* was allowed to make all possible connections in its oblivious crossbar in each routing cycle, while in the adaptive crossbar of **-Channels* only one connection per cycle was allowed. Furthermore, the number of inputs and outputs to their crossbars was equalized.

It is important to remark that **-Channels* outperformed *Oblivious* even for random traffic, and did so with a large performance gap. This gap is not commonly encountered when comparing adaptive and oblivious packet-switching routers [4], [17], [29]. This observation holds true in spite of increasing the number of extra lanes in the oblivious router to yield a total of 32 buffers per node.

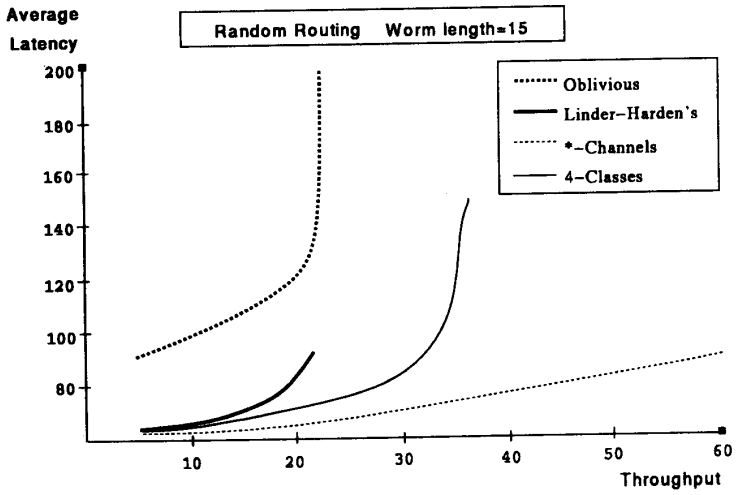


Fig. 8. Results for *Random Routing* with worms of length 15.

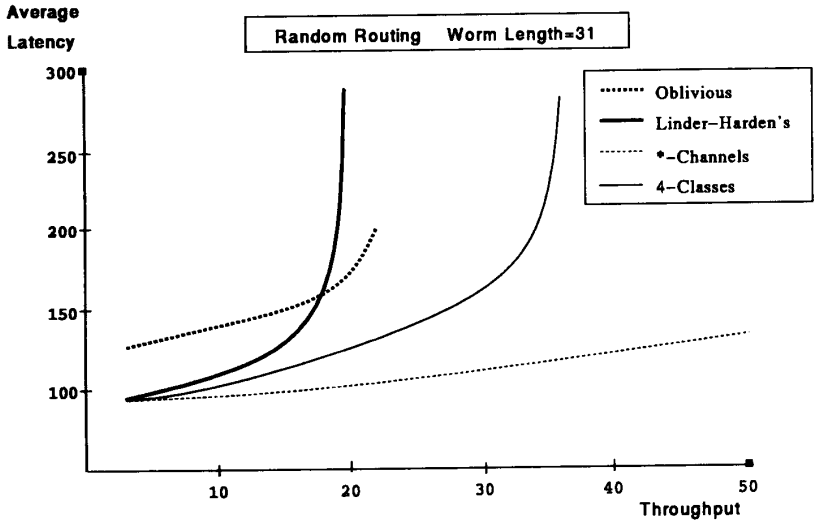


Fig. 9. Results for *Random Routing* with worms of length 31.

It should be noted that the oblivious routing used in the above comparison is nonminimal. The nonminimal algorithm was chosen because the simulations were conducted with the purpose of comparing the new algorithms presented in this paper with those known in the literature for worm-hole routing in the torus. Also, there is a large number of experimental results in the literature documenting the better performance of adaptive routing when compared with oblivious minimal routing. Therefore, it is strongly believed that no minimal oblivious routing will outperform the **-Channels* algorithm.

While the experimental results show that **-Channels* also outperformed the other two routers, this comparison would be unfair. The reason is that even though all routers have been equalized with respect to the number of virtual channels, the resulting adaptive crossbars have *different* sizes. Thus, *Linder-Harden's* algorithm is based on a 4×4 adaptive crossbar, and *4-Classes* on a 3×3 crossbar, while **-Channels* uses an adaptive crossbar connecting all inputs to all outputs.

On the other hand, *4-Classes* outperforms *Linder-Harden's*, and the comparison is fair because the size of the adaptive crossbar in the former is smaller than that in the latter. The result of this comparison is consistent for all message sizes tried, all patterns of communication, and all applied loads. To explain this, notice that *4-Classes* has more independent crossbars than *Linder–Harden's*, and so, it can perform more new connections per cycle. Also, this independent switching capability promotes a different flow control in the use of available buffers.

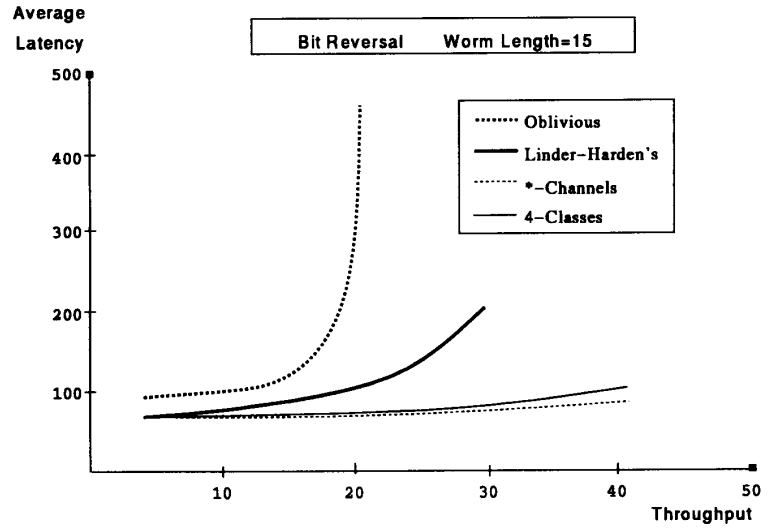


Fig. 10. Results for *Bit-Reversal Routing* with worms of length 15.

VI. Conclusion

Two new algorithms for deadlock- and livelock-free worm-hole routing in the torus network were presented.

The first algorithm, **-Channels*, is for the *n-dimensional* torus network, and requires the smallest number of channels known in the literature for fully-adaptive minimal worm-hole routing. In addition, these results also yield the smallest number of buffers known in the literature for packet-switched fully-adaptive minimal routing.

The second algorithm, *4-Classes*, is for the *bidimensional* torus network. This technique is fully-adaptive minimal and requires only eight virtual channels per bidirectional link for its implementation. Due to the characteristics of this algorithm, its routing node can be designed with a highly parallel implementation, since switching in a node can be decoupled into a set of eight independent 3 × 3 adaptive crossbars.

Both of the new algorithms work for messages of unknown size, thus yielding new routing techniques for both packet-switched and *worm-hole* models of message transmission. In addition, as no central queue is mandatory, these techniques can be used for *combined routing*, i.e., both short fixed-size packets, and long messages of unknown size are handled by the same network and routing resources.

The second part of this paper compared four worm-hole techniques for the two-dimensional torus. Simulation results on the performance of the four worm-hole algorithms were reported for a dynamic injection model and different traffic patterns and message lengths. **-Channels* displayed better performance than the other routing algorithms tried in all of the experiments. Regarding the algorithms that require smaller crossbars, *4-Classes* outperformed *Linder-Harden's* for every traffic pattern and message length tried.

Appendix A

Lemma 3.1: The star-channel dependency graph $\mathcal{CDG}^*$ associated with $\mathcal{R}$ is acyclic.

Proof: Consider the dependencies existing between channels belonging to orientation X_i^+ with prefix $i,+,0$. Suppose the head of a message m enters channel $c^*_{i,+,0,(x_{n-1},\cdots,x_i,\cdots,x_0)}$. Then, m has to correct dimension i following the X_i^+ orientation, and m has not taken a wrap-around along dimension X_i yet. If m's head enters channel $c^*_{i,+,0,(y_{n-1},\cdots,y_i,\cdots,y_0)}$ afterwards, then it follows that m has still not taken any wrap-around along dimension X_i yet. So, if $y_i \leq x_i$, then m's head would have had to move following orientation X_i^-, contradicting the minimality of $\mathcal{R}$. So, $y_i > x_i$. Consequently, no cycles can exist in $\mathcal{CDG}^*$ involving channels belonging to orientation X_i^+ with prefix $i,+,0$ exclusively. Analogously, no cycles can exist in $\mathcal{CDG}^*$ involving exclusively channels with prefix $i,+,1$. In addition, if a message m enters a star channel with prefix $i,+,1$, then m has already taken a wrap-around link along dimension X_i. Therefore, m will never enter a channel with prefix $i,+,0$ again, from the definition of $\mathcal{R}$. Consequently, for a given dimension X_i, there cannot be any dependency from a channel with prefix $i,+,1$ to a channel with prefix $i,+,0$ in $\mathcal{CDG}^*$. Analogous conclusions can be drawn for orientation X_i^-, $i = n-1,\cdots,0$.

Furthermore, as the routing function $\mathcal{R}$ is minimal, never can a message enter a star channel corresponding to orientation X_i^+ after entering a star channel corresponding to orientation X_i^-, or vice versa. Therefore, there are no dependencies between channels corresponding to a dimension X_i that have different orientations, $i = n-1,\cdots,0$.

So, if there is a cycle in $\mathcal{CDG}^*$, it must involve star channels corresponding to different dimensions.

Consider a dependency from a channel $c^*_{i,o,w,x}$ to a channel $c^*_{j,o',w',x'}$, with $i \neq j$. If the head of a message m enters $c^*_{i,o,w,x}$, it follows from $\mathcal{R}$ that at that moment X_i was the most significant dimension m's head had to correct. As the routing algorithm is minimal, and as X_j was the most significant dimension that needed correction when m's head

entered $c^*_{j,o',w',x'}$, then $i > j$. Therefore, there cannot be any cycle involving virtual channels corresponding to different dimensions. Consequently, $\mathcal{CDG}^*$ is acyclic.

Lemma 3.2: The routing algorithm that results from routing function $\mathcal{R}$ is free of deadlock.

Proof: First, suppose that no channel in C^* ever participates in a deadlock cycle. Therefore, as from the definition of $\mathcal{R}$ the head of a message that has not arrived at its destination always has a channel in C^* in its waiting set of virtual channels, no deadlock can ever arise, as virtual channels are assigned fairly, and messages are of finite length.

So, it suffices to show that no channel in C^* ever participates in a deadlock cycle.

As from Lemma 3.1, $\mathcal{CDG}^*$ is acyclic, each channel in C^* can be assigned a *level* in $\mathcal{CDG}^*$: Let $c^* \in C^*$. Then, *level* (c^*) is the length of the longest path between any *root* r^* of $\mathcal{CDG}^*$ such that there is a path from r^* to c^* in $\mathcal{CDG}^*$, and c^*. A virtual channel $r^* \in C^*$ is a root of $\mathcal{CDG}^*$ if r^* has indegree 0 in $\mathcal{CDG}^*$. As $\mathcal{CDG}^*$ is acyclic, *level*(c^*) is finite $\forall c^* \in C^*$.

It will be shown that no channel in C^* can ever participate in a deadlock cycle by using induction on the *level* of the star channels.

- *Basis:* Let $l^* \in C^*$ be such that it has outdegree 0 in $\mathcal{CDG}^*$. The basis is correct since every $c^* \in C^*$ has a channel $l^* \in C^*$ with outdegree 0 reachable from c^*, because $\mathcal{CDG}^*$ is acyclic. Suppose that l^* participates in a deadlock cycle, i.e., l^* holds a flit of a message m that is deadlocked. As m is deadlocked, m's head has not arrived at its destination node yet. Therefore, when m's head entered l^*, from the definition of $\mathcal{R}$, m's head had at least one possible next step within C^*. But this is not possible, as l^* has outdegree 0 in $\mathcal{CDG}^*$. Then, m's head arrived at its destination by entering l^*. So, m is not deadlocked. Consequently, l^* can never participate in a deadlock cycle.
- *Inductive Step:* Consider virtual channel $c^* \in \mathcal{CDG}^*$. It is supposed that c'^* can never participate in a deadlock cycle, $\forall c'^* \in C^* : level\ (c'^*) > level(c^*)$. Suppose that at a certain moment, there exists a message m with destination d that is deadlocked such that c^* contains a flit of m's. So, m's head has not arrived at its destination yet (as otherwise, m would not be deadlocked). Then, m's head is at a virtual channel $c \in C$ (not necessarily in C^*) such that $\exists c'^* \in C^* : c'^* \in \mathcal{R}(c, d)$, from the definition of $\mathcal{R}$. Then, as $c'^* \in C^*$, and by definition of $\mathcal{CDG}^*$, it follows that $(c^*, c'^*) \in D^*$. Then, *level* $(c'^*) >$ *level* (c^*), as $\mathcal{CDG}^*$ is acyclic. So, by inductive hypothesis, c'^* can never participate in a deadlock cycle. Then, as messages are finite and virtual channels are assigned fairly, m's head will eventually have c'^* available. Consequently, m is not deadlocked.

Therefore, no channel in C^* can ever be involved in a deadlock cycle.

APPENDIX B

Theorem 4.1: The *4-Classes* algorithm is correct, deadlock-free, and yields a worm-hole fully-adaptive minimal routing technique for the two-dimensional torus using eight virtual channels per bidirectional link.

Proof: Only class X^+Y^+ will be analyzed here. The other three classes can be treated analogously.

- *Correctness:* First consider a two-dimensional k-torus with k odd. Then, a message in class X^+Y^+ will remain in this class until it is delivered. If only the virtual channels with prefix 0 are considered (or equivalently, only those with prefix 1), then a directed network is obtained that allows to route from any node (x, y) to any other node (x', y') if and only if $x \leq x'$ and $y \leq y'$. In addition, never can a message use wrap-around links twice along the same dimension, as all the routes built by the routing algorithm are minimal.

Given a message m with a source node (x, y) and a destination node (x', y') such that m belongs to class X^+Y^+, there are four possible cases:

— $x \leq x'$ and $y \leq y'$: In this case, m moves through channels with prefix 1 ((5) and (6)), and keeps visiting channels with prefix 1 until it arrives at its destination node, using any of the minimal paths ((11) and (13)).

— $x > x'$ and $y \leq y'$: In this case, m starts moving through channels with prefix 0 ((1) and (3)), and moves adaptively using any of the minimal paths ((7) and (9)) until it uses a wrap-around link corresponding to dimension X. Then, m starts moving through the channels with prefix 1 ((8)). Note the special case when the wrap-around channel is taken as the first routing step ((2)).

After using the wrap-around channel, m will enter a virtual channel $c_{1,X,(0,y'')}$ such that $y'' \leq y'$. Therefore, m will be able to reach node (x', y') using adaptively all the minimal paths through channels with prefix 1 ((11) and (13)).

— $x \leq x'$ and $y > y'$: Analogous to the previous case, using (10) instead of (8), and (4) instead of (2).

— $x > x'$ and $y > y'$: m starts moving through channels with prefix 0 ((1) and (3)), and will move adaptively using all the minimal paths ((7) and (9)) until it takes a wrap-around link ((8) or (10)). Note the special case when m takes a wrap-around link as its first routing step ((2) and (4)). After taking a wrap-around link, m starts moving adaptively through channels with prefix 1 ((11) and (13)) until it takes the second wrap-around link ((12) or (14)). After this, m has arrived at a virtual channel $c_{0,d,(x'',y'')}$ such that $x'' \leq x'$ and $y'' \leq y'$. Therefore, m will move towards node (x', y') through channels with prefix 0 using (7) and (9).

As k is odd, a message will belong to only one class, as all the minimal paths from a message's source node to its destination node belong to only one class.

If k is even, then a message m may belong to one, two, or four classes. If m belongs to 4 classes, then it is $k/2$ steps away from its destination along both dimensions X and Y. m will be injected in the injection channel at its source node. As m takes its first step, it will fix one orientation in which to correct the dimension along which this first step is taken. So, after this first step, m will belong to two classes.

Now, the case of a message m belonging to two classes will be analyzed. Suppose that m belongs to classes X^+Y^+

and X^-Y^+. Then, if m has been injected in node (x, y), and has as destination node (x', y'), then $|x - x'| = k/2$. As explained above, before moving along dimension X, m will travel through virtual channels corresponding to class X^+Y^+. When m takes its first step along dimension X, there are two possible cases:

— The first step along dimension X is taken following orientation X^+. Therefore, from the routing function, m will remain in class X^+Y^+ until it arrives at its destination, as the distance from the nodes m will visit and m's destination node will never be $k/2$ again. The reason for this is that m will not be able to abandon class X^+Y^+ unless it is $k/2$ steps away from its destination along some dimension. But m is less than $k/2$ steps away from (x', y') along dimension X and dimension Y, and it can only move closer to (x', y') by following the routing function as far as class X^+Y^+ is concerned. So, m will not switch to another class, and the argument for the case k odd applies.

— The first step along dimension X is taken following orientation X^-. Therefore, from the routing function, m will move to class X^-Y^+. Furthermore, suppose that m entered node p by taking this first step in dimension X. As m was $k/2$ steps away from node (x', y') before switching class, then either X^+ or X^- could be chosen as orientations to correct dimension X following minimal paths. So, class X^-Y^+ contains *all* the minimal paths from node p to node (x', y'), as p is $k/2 - 1$ steps away from node (x', y') along dimension X following orientation X^-. Furthermore, as explained in the previous case, m will never leave class X^-Y^+. In addition, when changing class, m is treated as being injected in node p for the definition of the channels through which m will move in class X^-Y^+. Consequently, the same arguments as in the case k odd apply to show that m will be able to reach its destination.

The case of m belonging to other pairs of classes can be treated analogously.

- *Freedom from Deadlock:* It will be proved that the routing algorithm is deadlock-free by showing that the channel dependency graph (CDG) associated with $\mathcal{R}$ is acyclic. Again, the case k odd will be analyzed first.

First, notice that the injection channels have indegree 0 in the CDG, and so, they cannot participate in any cycle in the CDG. Regarding the other virtual channels, they can be divided into disjoint sets according to the class (X^+Y^+, X^+Y^-, X^-Y^+, or X^-Y^-) to which they belong. Never will a message pass from a virtual channel in one of these classes to a virtual channel in another class if k is odd. Consequently, the analysis of the existence of cycles can be performed independently for each of these four classes of virtual channels. X^+Y^+ will be analyzed below. The other three cases can be handled analogously.

As pointed out above, no injection channel can participate in a cycle in the CDG. So, (1) through (6) do not generate any cycle in the CDG. A cycle in the CDG cannot consist solely of channels with prefix 0. This is so because if the equations that define movements within this set of channels, i.e., (7) and (9), allow transitions from a channel $c_{0,d,(x,y)}$ to a channel $c_{0,d,(x',y')}$, then $x + y < x' + y'$. An analogous consideration can be made regarding a cycle consisting solely of channels with prefix 1, but using (11) and (13). So, only the case in which a cycle is formed by both channels with prefix 0 and prefix 1 has to be analyzed. Furthermore, a cycle must have both a link from a channel with prefix 0 to a channel with prefix 1, and a link from a channel with prefix 1 to a channel with prefix 0.

Consider a link from a channel with prefix 1 to a channel with prefix 0. Such a dependency can only be built by (12) and (14). The link will be either $(c_{1,d,(k-1,y)}, c_{0,X,(0,y)})$ or $(c_{1,d,(x,k-1)}, c_{0,Y,(x,0)})$, for some $d \in \{X, Y\}$. Consider the first case. A message m using that dependency is going through a wrap-around link corresponding to dimension X. Therefore, m started moving through the network using virtual channels with prefix 0, and then it switched to the channels with prefix 1 by taking a wrap-around corresponding to dimension Y. As a result of this, and the fact that the paths taken by messages are shortest, $y < \lfloor k/2 \rfloor$, as a message travels at most $\lfloor k/2 \rfloor$ steps along any dimension. Summing up, there can only be a dependency in the CDG from $c_{1,d,(k-1,y)}$ to $c_{0,X,(0,y)}$ if $y < \lfloor k/2 \rfloor$. Analogously, there can only be a dependency from $c_{1,d,(x,k-1)}$ to $c_{0,Y,(x,0)}$ if $x < \lfloor k/2 \rfloor$. Consequently,

$$A_1 = \{c_{0,d,(x,y)} : d = X, Y; (x = 0 \text{ or } y = 0) \text{ and } x, y < \lfloor k/2 \rfloor\}$$

is the set of all the virtual channels with prefix 0 that may have a dependency from a channel with prefix 1 in the CDG.

For a cycle to exist, it must have a link from a channel with prefix 0 to a channel with prefix 1.

$$A_2 = \{c_{0,d,(x,y)} : d = X, Y; (x = k - 1 \text{ or } y = k - 1)\}$$

is the set of channels that may have a dependency to a channel with prefix 1. To reach a channel in A_2 from a channel in A_1 following a path in the CDG, there must exist links in the CDG from a channel in

$$A_3 = \{c_{0,d,(x,y)} : d = X, Y; (x \leq \lfloor k/2 \rfloor \text{ and } y = \lfloor k/2 \rfloor) \text{ or } (y \leq \lfloor k/2 \rfloor \text{ and } x = \lfloor k/2 \rfloor)\}$$

to a channel in

$$A_4 = \{c_{0,d,(x,y)} : d = X, Y; (x \leq \lfloor k/2 \rfloor \text{ and } y = \lfloor k/2 \rfloor + 1) \text{ or } (y \leq \lfloor k/2 \rfloor \text{ and } x = \lfloor k/2 \rfloor + 1)\}$$

It will be proved that there are no links from A_3 to A_4 in the CDG. Therefore, the CDG is acyclic.

To see this, notice that a message m arriving at a channel in A_3 has already taken two wrap-around links necessarily. m could not have taken only one wrap-around, as in that case, it would be in a channel with prefix 1. Finally, if m had not gone through any wrap-around yet, as it is in a channel with prefix 0, it would have to take a wrap-around link. Otherwise, it would have started moving through channels with prefix 1, according to $\mathcal{R}$. So, m would have to travel at least $\lfloor k/2 \rfloor$

steps more along one dimension, to be able to wrap. But this is not possible. So, m has already taken two wrap-around links, one in each dimension. So, if m would move from a channel in A_3 to a channel in A_4, this would involve traveling more than $\lfloor k/2 \rfloor$ steps along one dimension. Then, no message at a channel in A_3 ever passes to a channel in A_4. Therefore, the CDG is acyclic for the case k odd. Now, the case k even will be analyzed. As explained above, messages can change from one class to another according to their final destination. Messages can switch from class X^+Y^+ to either class X^+Y^- or class X^-Y^+, and from class X^+Y^- and class X^-Y^+ to class X^-Y^-. Never can a message switch from class X^-Y^- to class X^+Y^-, for example. Without considering this switching between the classes, the CDG is acyclic (case k odd). So, if there is a cycle in the CDG, it must involve at least one dependency generated by this switching between the classes.

Suppose that a dependency from a virtual channel belonging to virtual network X^+Y^+ to a virtual channel belonging to X^-Y^+ participates in a cycle in the CDG. Therefore, there must be a dependency from some virtual network *towards* X^+Y^+. But this is not possible, because no message can switch from a virtual network other than X^+Y^+ to virtual network X^+Y^+. The other possible cases are analyzed analogously. Consequently, the CDG is acyclic for the case k even as well.

ACKNOWLEDGMENT

The authors are indebted to four reviewers for their very useful comments and suggestions.

REFERENCES

[1] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, *et al.*, "iWarp: An integrated solution to high-speed parallel computing," in *Proc. Supercomputing 88*, ACM, 1988.

[2] 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), June 1992.

[3] Y. Birk, P. B. Gibbons, D. Soroker, and J. L. C. Sanz, "A simple mechanism for efficient barrier synchronization in MIMD machines," *RJ 7078 (67141) Comput. Sci.*, IBM Almaden Res. Ctr., Oct. 1989.

[4] K. Bolding and L. Snyder, "Mesh and torus chaotic routing," in *MIT/Brown Advanced Res. in VLSI and Parallel Syst. Conf.*," Mar. 1992.

[5] F. Chong, E. Egozy, A. DeHon, and T. Knight, "Multipath fault tolerance in multistage interconnection networks," *Transit note # 48*, MIT, June 1991.

[6] R. Cypher and L. Gravano, "Adaptive deadlock-free packet routing in torus networks with minimal storage," in *Proc. ICPP 92*, 1992.

[7] R. Cypher and L. Gravano, "Requirements for deadlock-free, adaptive packet routing," in *Proc. PODC 92*, 1992.

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

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

[10] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distrib. Computing*, pp. 187–196, 1986.

[11] ——, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput.*, vol. 36, pp. 547–553, May 1987.

[12] W. J. Dally and P Song, "Design of a self-timed VLSI multicomputer communication controller," in *Proc. Int. Conf. Comput. Design*, 1987, pp. 230–234.

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

[14] 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*, 1991.

[15] S. A. Felperin, L. Gravano, G. D. Pifarré, and J. L. C. Sanz, "Fully-adaptive routing: Packet switching performance and worm-hole algorithms," in *Supercomputing*, 1991.

[16] S. A. Felperin, L. Gravano, G. D. Pifarré, and J. L. C. Sanz, " Routing Techniques for Massively Parallel Communication," *Proc. IEEE, Special Issue on Massively Parallel Computers*, vol. 79, no. 4, Apr. 1991.

[17] S. A. Felperin, H. Laffitte, G. Buranits, and J. L. C. Sanz, "Deadlock-free minimal packet routing in the torus network," Tech. Rep. 91–22, IBM Argentina, CRAAG, 1991.

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

[19] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proc. 19th Annu. Int. Symp. Comput. Architecture*, May 1992, pp. 278–287.

[20] L. Gravano, G. D. Pifarré, S. A. Felperin, and J. L. C. Sanz, "Adaptive deadlock-free worm-hole routing with all minimal paths," Tech. Rep. 91–21, IBM Argentina, CRAAG, 1991.

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

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

[23] S. L. Johnsson and C. T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Comput.*, 38, no. 9, pp. 1249–1268, Sept. 1989.

[24] C. R. Jesshope, P. R. Miller, and J. T. Yantchev, "High performance communication in processor networks," in *Proc. 16th Annu. Int. Symp. Comput. Architecture*, pp. 150–157, May 1989.

[25] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Comput. Netw.*, 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. Annu. ACM SPAA*, 1990, pp. 21–30.

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

[30] S. Konstantinidou and E. Upfal, "Experimental comparison of multistage interconnection networks," RJ:8451 (76459), IBM Almaden Res. Ctr., Nov. 1991.

[31] T. Leighton, "Average case analysis of greedy routing algorithms on arrays," in *SPAA*, 1990.

[32] 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, no. 1, pp. 2–12, Jan. 1991.

[33] 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.

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

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

[36] P. M. Merlin and P. J. Schweitzer, "Deadlock avoidance in store-and-forward networks, 1: Store-and-forward deadlock," *IEEE Trans. Commun.*, vol. 28, no. 3, pp. 345–354, Mar. 1980.

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

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

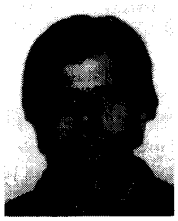
[39] ——, "Adaptive routing in multicomputers," in *Opportunities and Constraints of Parallel Computing*, J. L. C. Sanz, Ed. New York: Springer-Verlag, 1989.

[40] 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 Annu. ACM Symp. Parallel Algorithms and Architectures*, 1991.

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

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

[43] A. G. Ranade, S. N. Bhat, and S. L. Johnson, "The Fluent abstract machine," in *Fifth MIT Conference on Advanced Research in VLSI*, J. Allen and F. T. Leighton, editors. Cambridge, MA: The MIT Press, Mar. 1988, pp. 71–93.
[44] P. Raghavan and E. Upfal, "A theory of wormhole routing in parallel computers," Tech. Rep., IBM Res., Dec. 1991.
[45] P. Y. Song, "Design of a network for concurrent message passing systems," Master's thesis, Massachusetts Inst. of Technol., Dep. Comput. Sci., May 1988.
[46] E. Upfal, "An **O** (log N) deterministic packet routing scheme," in *21st Annu. ACM SIGACT Symp. Theory of Computing*, May 1989.
[47] L. G. Valiant, "General purpose parallel architectures," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Amsterdam, The Netherlands: North-Holland, 1988.



Luis 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.

Gustavo 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.

Pablo E. Berman received the B.S. degree in computer science from the Escuela Superior Latino-Americana de Informatica (ESLAI) in 1992.

During 1991 and 1992, he was a Researcher at the Computer Research and Advanced Applications Group in IBM Argentina, working on parallel computers. He is now an independent software consultant. His current areas of research interest include software engineering, operating systems, connectivity, real-time programming, and distributed systems.

Jorge 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.