Principles and Practices of Interconnection Network

Article · January 2004

CITATIONS
2,070

READS
2,493

2 authors:

William J. Dally
NVIDIA
383 PUBLICATIONS **31,439** CITATIONS
SEE PROFILE

Brian Towles
D. E. Shaw Research
51 PUBLICATIONS **8,505** CITATIONS
SEE PROFILE

Deadlock-Free Message Routing in Multiprocessor Interconnection Networks

WILLIAM J. DALLY, MEMBER, IEEE, AND CHARLES L. SEITZ, MEMBER, IEEE

Abstract—**A deadlock-free routing algorithm can be generated for arbitrary interconnection networks using the concept of virtual channels. A necessary and sufficient condition for deadlock-free routing is the absence of cycles in a channel dependency graph. Given an arbitrary network and a routing function, the cycles of the channel dependency graph can be removed by splitting physical channels into groups of virtual channels. This method is used to develop deadlock-free routing algorithms for k-ary n-cubes, for cube-connected cycles, and for shuffle-exchange networks.**

Index Terms—**Communication networks, concurrent computation, graph model, interconnection networks, message passing multiprocessors, parallel processing.**

I. Introduction

MESSAGE passing concurrent computers [13] such as the Cosmic Cube [11] consist of many processing nodes that interact by sending messages over communication channels between the nodes. Deadlock in the interconnection network of a concurrent computer occurs when no message can advance toward its destination because the queues of the message system are full [7]. Consider the example shown in Fig. 1. The queue of each node in the 4-cycle is filled with messages destined for the opposite node. No message can advance toward its destination; thus, the cycle is deadlocked. In this locked state, no communication can occur over the deadlocked channels until exceptional action is taken to break the deadlock.

In this paper we consider networks that use *wormhole* [12] rather than *store-and-forward* [16] routing. Instead of storing a packet completely in a node and then transmitting it to the next node, wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels. Only a few flow control digits (flits) are buffered at each node. A *flit* is the smallest unit of information that a queue or channel can accept or refuse.

As soon as a node examines the header flit(s) of a message, it selects the next channel on the route and begins forwarding flits down that channel. As flits are forwarded, the message

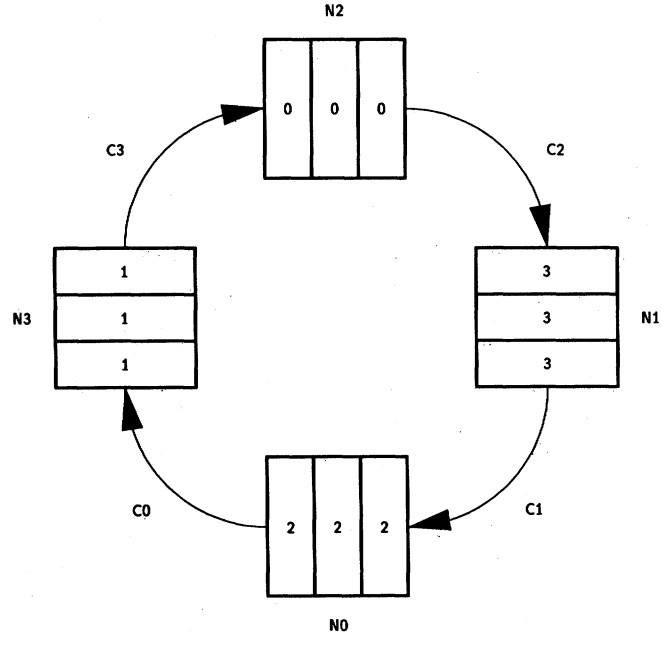


Fig. 1. Deadlock in a 4-cycle.

becomes spread out across the channels between the source and destination. It is possible for the first flit of a message to arrive at the destination node before the last flit of the message has left the source. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with the flits of other messages. When the header flit of a message is blocked, all of the flits of a message stop advancing and block the progress of any other message requiring the channels they occupy.

A method similar to wormhole routing, called *virtual cut-through*, is described in [6]. Virtual cut-through differs from wormhole routing in that it buffers messages when they block, removing them from the network. The deadlock properties of cut-through routing are accordingly identical to those of store-and-forward routing. With wormhole routing, blocked messages remain in the network.

Many deadlock-free routing algorithms have been developed for store-and-forward computer communications networks [5], [9], [17], [18], [4]. These algorithms are based on the concept of a *structured buffer pool*. The message buffers in each node of the network are partitioned into classes, and the assignment of buffers to messages is restricted to define a partial order on buffer classes. The structured buffer pool

Manuscript received May 15, 1985; revised July 2, 1986. This work was supported in part by the Defense Advanced Research Projects Agency, ARPA Order 3771, and monitored by the Office of Naval Research under Contract N00014-79-0587, in part by a grant from Intel Corporation, and in part by an AT&T Ph.D. fellowship.

W. J. Dally is with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139.

C. L. Seitz is with the Department of Computer Science, California Institute of Technology, Pasadena, CA 91125.

IEEE Log Number 8713946.

method has in common with the virtual channel method that both prevent deadlock by assigning a partial order to resources. The two methods differ in that the structured buffer pool approach restricts the assignment of buffers to packets while the virtual channel approach restricts the routing of messages. Either method can be applied to store-and-forward networks, but the structured buffer pool approach is not directly applicable to wormhole networks, since the flits of a message cannot be interleaved the way that packets can. Once one flit of a message is accepted and assigned a buffer, the remaining flits must be accepted before the flits of any other message can be accepted by the channel. With wormhole routing, we cannot restrict buffer allocation to break deadlock. Instead, we must restrict routing.

Reliable concurrent computation requires a routing algorithm that is provably free of deadlock. In this paper we present a general method for constructing deadlock-free routing algorithms for arbitrary networks. In Section II we state our assumptions and develop the necessary and sufficient conditions on a *channel dependency graph* for a routing to be deadlock free. These conditions are used in Section III to develop a method of constructing deadlock-free algorithms for arbitrary networks. This method is based on the concept of virtual channels: groups of channels that share a physical channel but in which each virtual channel has its own queue. Using virtual channels, we develop deadlock-free routing algorithms for arbitrary k-ary n-cubes in Section IV, for cube-connected cycles in Section V, and for shuffle-exchange networks in Section VI. In each of these examples, physical channels belonging to cycles are split into a group of virtual channels. The virtual channels are ordered; routing is restricted to visit channels in decreasing order to eliminate cycles in the channel dependency graph.

II. Deadlock-Free Routing

We assume the following.

1) A message arriving at its destination node is eventually consumed.

2) A node can generate messages destined for any other node.

3) The route taken by a message is determined only by its destination, and not by other traffic in the network (*deterministic or nonadaptive* routing).

4) A node can generate messages of arbitrary length. Packets will generally be longer than a single flit.

5) Once a queue accepts the first flit of a message, it must accept the remainder of the message before accepting any flits from another message.

6) An available queue may arbitrate between messages that request that queue space, but may not choose among waiting messages.

7) Nodes can produce messages at any rate subject to the constraint of available queue space (source queued).

The following definitions develop a notation for describing networks, routing functions, and configurations. A summary of notation is given below.

Definition 1: An *interconnection network* I is a strongly connected *directed* graph, $I = G(N, C)$. The vertices of the graph N represent the set of processing nodes. The edges of the graph C represent the set of communication channels. Associated with each channel, c_i, is a queue with capacity cap (c_i). The source node of channel c_i is denoted s_i and the destination node d_i.

Definition 2: A *routing function* $\boldsymbol{R}: C \times N \mapsto C$ maps the current channel c_c and destination node n_d to the next channel c_n on the route from c_c to n_d, $\boldsymbol{R}(c_c, n_d) = c_n$. A channel is not allowed to route to itself, $c_c \neq c_n$. Note that this definition restricts the routing to be memoryless in the sense that a message arriving in channel c_c has no memory of the route that brought it to c_c. However, this formulation of routing as a function from $C \times N$ to C has more memory than the conventional definition of routing as a function from $N \times N$ to C. Making routing dependent on the current channel rather than the current node allows us to develop the idea of channel dependence.

Definition 3: A *channel dependency graph* D for a given interconnection network I and routing function $\boldsymbol{R}$, is a directed graph, $D = G(C, E)$. The vertices of D are the channels of I. The edges of D are the pairs of channels connected by $\boldsymbol{R}$:

$$E = \{(c_i, c_j) \mid \boldsymbol{R}(c_i, n) = c_j \text{ for some } n \in N\}. \quad (1)$$

Since channels are not allowed to route to themselves, there are no 1-cycles in D.

Definition 4: A *configuration* is an assignment of a list of nodes to each queue. The number of flits in the queue for channel c_i will be denoted size (c_i). If the first flit in the queue for channel c_i is destined for node n_d, then head $(c_i) = n_d$. A configuration is legal if

$$\forall c_i \in C, \text{ size } (c_i) \leq \text{cap } (c_i) \quad (2)$$

Definition 5: A *deadlocked configuration* for a routing function $\boldsymbol{R}$ is a nonempty legal configuration of channel queues $\ni$

$$\forall c_i \in C, (\text{head } (c_i) \neq d_i \text{ and } c_j = \boldsymbol{R}(c_i, n) \Rightarrow \text{size } (c_j) = \text{cap } (c_j)). \quad (3)$$

In this configuration no flit is one step from its destination and no flit can advance because the queue for the next channel is full. A routing function $\boldsymbol{R}$ is *deadlock free* on an interconnection network I if no deadlock configuration exists for that function on that network.

Summary of Notation

I	interconnection network, a directed graph $I = G(N, C)$,
N	the set of nodes,
n_i	a node,
C	the set of channels,
c_i	a channel,
s_i	the source node of channel c_i,
d_i	the destination node of channel c_i,
cap (c_i)	the capacity of the queue of channel c_i,
size (c_i)	the number of flits enqueued for channel c_i,
head (c_i)	the destination of the head flit enqueued for channel c_i,

R a routing function $R:C \times N \mapsto C$,
D the channel dependency graph.

Theorem 1: A routing function R for an interconnection network I is deadlock free iff there are no cycles in the channel dependency graph D.

Proof: $\Rightarrow$ Suppose a network has a cycle in D. Since there are no 1-cycles in D, this cycle must be of length two or more. Thus, one can construct a deadlocked configuration by filling the queues of each channel in the cycle with flits destined for a node two channels away, where the first channel of the route is along the cycle.

$\Leftarrow$ Suppose a network has no cycles in D. Since D is acyclic, one can assign a total order to the channels of C so that if $(c_i, c_j) \in E$ then $c_i > c_j$. Consider the least channel in this order with a full queue c_l. Every channel c_n that c_l feeds is less than c_l and thus does not have a full queue. Thus, no flit in the queue for c_l is blocked, and one does not have deadlock. ■

III. Constructing Deadlock-Free Routing Algorithms

Now that we have established this if-and-only-if relationship between deadlock and the cycles in the channel dependency graph, we can approach the problem of making a network deadlock free by breaking the cycles. We can break such cycles by splitting each physical channel along a cycle into a group of *virtual channels*. Each group of virtual channels shares a physical communication channel; however, each virtual channel requires its own queue.

Consider for example the case of a unidirectional four-cycle as shown in Fig. 2(a), $N = \{n_0, \cdots, n_3\}$, $C = \{c_0, \cdots, c_3\}$. The interconnection graph I is shown on the left and the dependency graph D is shown on the right. We pick channel c_0 to be the dividing channel of the cycle and split each channel into high virtual channels, $c_{10}, \cdots, c_{13}$, and low virtual channels, $c_{00}, \cdots, c_{03}$, as shown in Fig. 2(b).

Messages at a node numbered less than their destination node are routed on the high channels, and messages at a node numbered greater than their destination node are routed on the low channels. Channel c_{00} is not used. We now have a total ordering of the virtual channels according to their subscripts: $c_{13} > c_{12} > c_{11} > c_{10} > c_{03} > c_{02} > c_{01}$. Thus, there is no cycle in D and the routing function is deadlock free. In the next three sections we apply this technique to three practical communications networks. In each case we add virtual channels and restrict the routing to route messages in order of decreasing channel subscripts.

IV. k-ary n-cubes

The e-cube routing algorithm [15], [8] guarantees deadlock-free routing in binary n cubes. In a cube of dimension n, we denote a node as n_k where k is an n-digit binary number. Node n_k has n output channels, one for each dimension, labeled $c_{0k}, \cdots, c_{(n-1)k}$. The e-cube algorithm routes in decreasing order of dimension. A message arriving at node n_k destined for node n_l is routed on channel c_{ik} where i is the position of the most significant bit in which k and l differ. Since messages are routed in order of decreasing dimension, and hence decreasing channel subscript, there are no cycles in the channel dependency graph and e-cube routing is deadlock free.

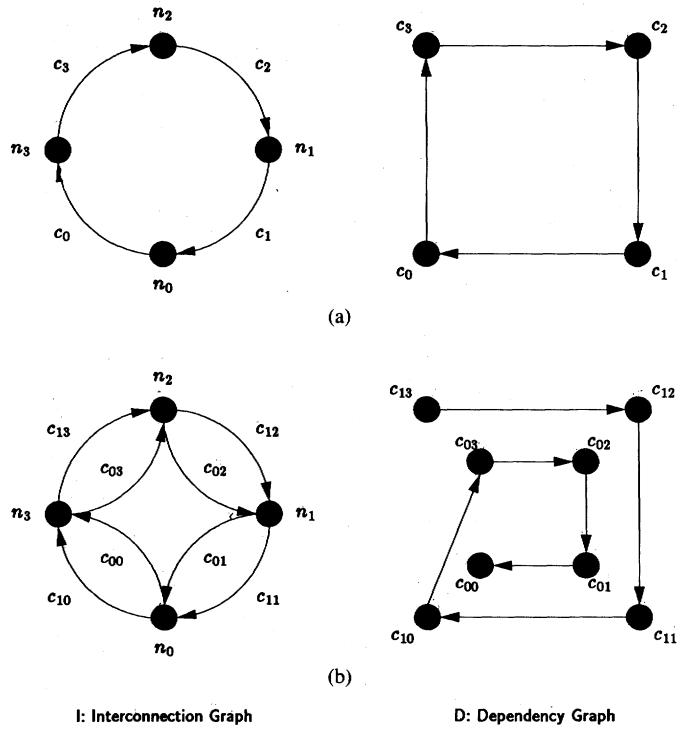


Fig. 2. Breaking deadlock by adding virtual edges.

Using the technique of virtual channels, this routing algorithm can be extended to handle all k-ary n-cubes: cubes with dimension n and k nodes in each dimension. Rings and toroidal meshes are included in this class of networks. This algorithm can also handle mixed radix cubes. Each node of a k-ary n-cube is identified by an n-digit radix k number. The ith digit of the number represents the node's position in the ith dimension. For example, the center node in the 3-ary 2-cube of Fig. 3 is n_{11}. The channels are identified by the number of their source node and their dimension. For example, the dimension 0 (horizontal) channel from n_{11} to n_{10} is c_{011}. To break cycles we divide each channel into an upper and lower virtual channel. The upper virtual channel of c_{011} will be labeled c_{0111}, and the lower virtual channel will be labeled c_{0011}. Virtual channel subscripts are of the form dvx where d is the dimension, v selects the virtual channel, and x identifies the source node of the channel. To assure that the routing is deadlock free, we restrict it to route through channels in order of descending subscripts.

As in the e-cube algorithm we route in order of dimension, most significant dimension first. In each dimension i, a message is routed in that dimension until it reaches a node whose subscript matches the destination address in the ith position. The message is routed on the high channel if the ith digit of the destination address is greater than the ith digit of the present node's address. Otherwise, the message is routed on the low channel. It is easy to see that this routing algorithm routes in order of descending subscripts, and is thus deadlock free.

Formally, we define the routing function.

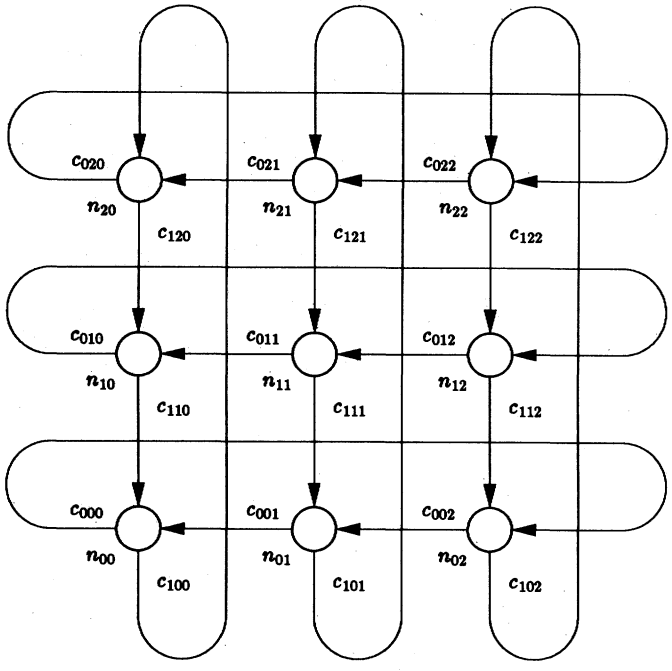


Fig. 3. 3-ary 2-cube.

$$R_{KNC}(c_{dvx}, n_j) = \begin{cases} c_{d1(x-k^d)} \\ \quad \text{if } (\text{dig } (x, d) < \text{dig } (j, d)) \wedge (\text{dig } (x, d) \neq 0), \\ c_{d0(x-k^d)} \\ \quad \text{if } (\text{dig } (x, d) > \text{dig } (j, d)) \vee (\text{dig } (x, d) = 0), \\ c_{i1(x-k^d)} \\ \quad \text{if } (\forall k > i, \text{dig } (x, k) = \text{dig } (j, k)) \\ \qquad \wedge (\text{dig } (n, i) \neq \text{dig } (j, i)). \end{cases} \tag{4}$$

Where dig (x, d) extracts the dth digit of x, and k is the radix of the cube. The subtraction, $x - k^d$, decrements the dth digit of x modulo k.

Assertion 1: The routing function R_{KNC} correctly routes messages from any node to any other node in a k-ary n-cube.

Proof: By induction on dimension d. For $d = 1$, a message, destined for n_j, enters the system at n_i. If $i < j$, the message is forwarded on channels, $c_{01i}, \cdots, c_{010}, c_{00k-1}, \cdots, c_{00(j+1)}$ to node n_j. If $i > j$, the path taken is $c_{00i}, \cdots, c_{00(j+1)}$. In both cases the route reaches node n_j.

Assume that the routing works for dimensions $\leq d$. Then for dimension $d + 1$ there are two cases. If dig $(i, d) \neq$ dig (j, d), then the message is routed around the most significant cycle to a node $n_x \ni$ dig $(x, d) =$ dig (j, d), as in the $d = 1$ case above. If dig $(i, d) =$ dig (j, d), then the routing need only be performed in dimensions d and lower. In each of these cases, once the message reaches a node, $n_x, \ni$ dig $(x, d) =$ dig (j, d), the third routing rule is used to route the message to a lower dimensional channel. The problem has then been reduced to one of dimension $\leq d$ and by induction the routing reaches the correct node. ■

Assertion 2: The routing function R_{KNC} on a k-ary n-cube interconnection network I is deadlock free.

Proof: Since routing is performed in decreasing order of channel subscripts, $\forall c_i, c_j, n_c \ni R(c_i, n_c) = c_j, i > j$, the channel dependency graph, D is acyclic. Thus, by Theorem 1 the route is deadlock free. ■

V. Cube-Connected Cycles

The cube-connected cycle (CCC) [10] is an interconnection network based on the binary n-cube. In the CCC, each node of a binary n-cube is replaced with an n-cycle, and the cube connection in the nth dimension is attached to the nth node in the cycle. A CCC of dimension 3 is shown in Fig. 4.

Each node in the CCC is labeled with the position of its cycle (an n-bit binary number), and its position within the cycle. For example, in Fig. 4, node 2 in cycle 111 is labeled n_{2111}. There are two channels out of each node: an in-cycle channel and an out-of-cycle channel. The in-cycle channel is split into three virtual channels. One set of virtual channels is used to rotate a message around the cycle to get to the most significant node in the cycle. These channels are labeled c_{2d0c} where d is the dimension of the node (its position in the cycle), and c is the cycle address. The next set of virtual channels is used to decrement the dimension during the e-cube routing of the message between cycles. These channels are labeled c_{1d0c}. The out-of-cycle channels, labeled c_{1d1c}, are used to toggle the bit of the current cycle address corresponding to dimension d. Note that the channels c_{1d1c} are actually physical channels. These connections are not shared with any other channels. The third set of virtual channels is used to rotate the message around the cycle to its destination once it is in the proper cycle. These channels are labeled c_{0d0c}. As above, we will restrict our routing to route through channels in order of descending subscripts.

The routing algorithm proceeds in three phases. During phase 1, messages are routed around the cycle using the first set of virtual channels until they reach a node with dimension greater than or equal to the position of the most significant bit in which the destination cycle address differs from the current cycle address. During phase 2 we route the message to the proper cycle using a variant of the e-cube algorithm. At each step of phase 2, we find the most significant dimension i in which the current cycle and destination cycle addresses differ. The message is routed around the current cycle until it reaches the node with dimension i and is then routed out of the cycle. When the message arrives in the destination cycle, it is routed around the cycle using the third set of virtual channels to reach its destination node. It is easy to see that routing is always performed in order of decreasing channel numbers, and thus the routing is guaranteed to be deadlock free.

While most cube-connected cycles are binary, this routing algorithm can be extended for k-ary cube-connected cycles, that is, cubes with k cycles in each dimension. The only modification required is to split each out-of-cycle channel into two virtual channels. For simplicity, however, we will analyze

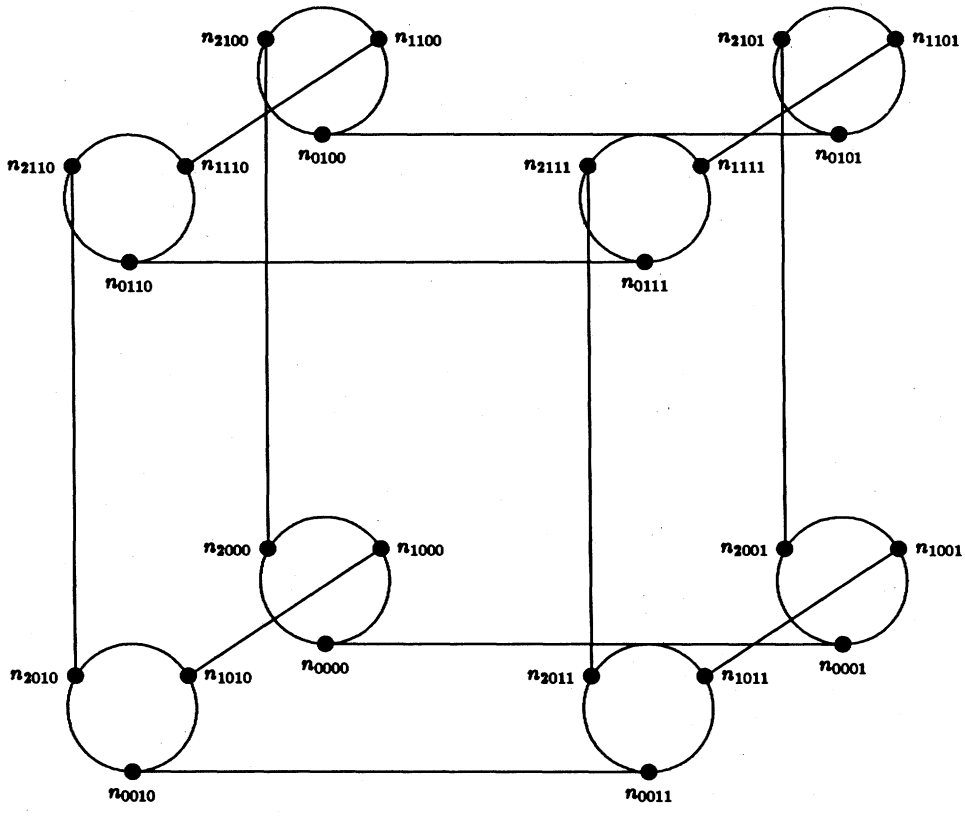


Fig. 4. Cube connected cycle of dimension 3.

the routing for the binary case only. Formally, we define the routing function.

$$R_{CCC}(c_{vdxc}, n_{d'c'}) = \begin{cases} c_{2(d-1)0c} & \text{if } (v \geq 2) \wedge (d > 0) \\ & \quad \wedge (\exists i > d \quad \text{dig } (i, c) \neq \text{dig } (i, c')), \\ c_{1(d-1)1c} & \text{if } (v \geq 1) \wedge (x = 0) \\ & \quad \wedge (\forall i > d \text{ dig } (i, c) = \text{dig } (i, c')) \\ & \quad \wedge (\text{dig } (d, c) \neq \text{dig } (d, c')), \\ c_{1(d-1)0c} & \text{if } (v \geq 1) \wedge (x = 0) \\ & \quad \wedge (\forall i \geq d \text{ dig } (i, c) = \text{dig } (i, c')), \\ c_{1d0(c-k^d)} & \text{if } (v = 1) \wedge (x = 1) \\ & \quad \wedge (\forall i \geq d \text{ dig } (i, c) = \text{dig } (i, c')), \\ c_{1d1(c-k^d)} & \text{if } (v = 1) \wedge (x = 1) \\ & \quad \wedge (\forall i > d \text{ dig } (i, c) = \text{dig } (i, c')) \\ & \quad \wedge (\text{dig } (d, c) \neq \text{dig } (d, c')), \\ c_{0(d-1)0c} & \text{if } (x = 0) \wedge (c = c') \wedge (d' < d), \\ c_{0(d)0c} & \text{if } (x = 1) \wedge (c = c') \wedge (d' < d). \end{cases} \tag{5}$$

Assertion 3: $(v < 2) \Rightarrow \text{dig } (i, c) = \text{dig } (i, c') \; \forall i > d$.

Proof: The only way v can be decreased to less than 2 is for the right side of the assertion to hold. Initially $v = 3$. Then as long as $\exists i > d \ni \text{dig } (i, c) \neq \text{dig } (i, c')$, the first routing rule forwards the message along channels for which $v = 2$. When the arriving channel has $d = 0$, the current node has $d = n - 1$ (where n is the dimension of the CCC). In this case the first routing rule forwards the message along a channel for which $v = 1$. The assertion holds since there are only n digits in c and c', so there is no $i \geq n$ for which the ith digit of these two cycle addresses differ.

Once $v < 2$, the right side of the assertion continues to hold because of routing rules 2, 3, 4, and 5. We prove this by induction on d. For $d = n - 1$, the assertion holds as stated above. If we assume the assertion holds for $d = i$, then for $d = i - 1$ it also holds since routing rules 2 and 5 route the messages out-of-cycle to toggle the dth digit of n if the addresses disagree in that digit. ■

Assertion 4: $(v = 0) \Rightarrow (c' = c) \wedge (d' < d)$.

Proof: The only way to decrease v to zero is by routing rules 6 and 7 which require the right side of the assertion. By Assertion 3, when $v < 2$ and $d = 0$, after one or two more traversals, the right side of the assertion will be met. Once $v = 0$, no out-of-cycle channels will be used so c does not change. ■

Assertion 5: The routing function R_{CCC} correctly routes messages from any node to any other node in a k-ary cube-connected cycle.

Proof: Since there are only a finite number of channels, and R_{CCC} routes in order of decreasing channel subscripts, v will eventually be decreased to 0. Then by Assertion 4, $c' = c$

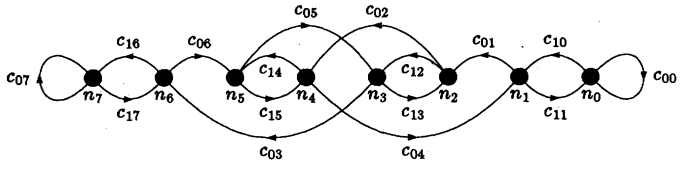


Fig. 5. Shuffle-exchange network, $N = 8$.

and $d' < d$, so the message will be rotated about the cycle until it reaches its destination. ■

Assertion 6: The routing function R_{CCC} on a k-ary cube-connected cycle interconnection network I is deadlock free.

Proof: As in the case of k-ary n-cubes, since routing is performed in decreasing order of channel subscripts, $\forall c_i, c_j, n_c \ni R(c_i, n_c) = c_j, i > j$, the channel dependency graph, D is acyclic. Thus, by Theorem 1 the route is deadlock free. ■

VI. Shuffle-Exchange Networks

The shuffle-exchange network [14], shown in Fig. 5, provides two channels out of each node: a shuffle channel and an exchange channel. The shuffle channel from node n_i has as its destination the node n_j where the binary representation of j is the left rotation of the binary representation of i, denoted here $j = \text{rol}(i)$. The exchange channel from n_i routes messages to n_k where the binary representations of k and i differ in the least significant bit.

The exchange channel out of n_i is labeled c_{1i}. The shuffle channel is labeled c_{0i}. For the shuffle-exchange network we split each channel into n virtual channels where $N = 2^n$. That is, we have one virtual channel for each bit of node address. Readers understanding the relationship between the binary n-cube and the shuffle [14] will find this assignment of virtual channels unsurprising since the shuffle is a uniform single stage of an n-stage indirect binary n-cube. The virtual channels are labeled c_{dxi}, where $0 \le d \le n-1, x \in \{0, 1\}$, and $0 \le i \le N$.

The routing algorithm, like the e-cube algorithm, routes a message toward its destination one bit at a time beginning with the most significant bit. At the ith step of the route, the $n-i$th bit of the destination address is compared to the least significant bit of the current node address. If the two bits agree, the message is forwarded over the shuffle channel to rotate the node address around to the next bit. Otherwise, the message is forwarded over the exchange channel to bring the two bits into agreement and then over the shuffle channel to rotate the address. At the ith step messages are forwarded over channels with $d = n - i$. Since d is always decreasing and, during a single step, the exchange channel is used before the shuffle channel, messages are routed in order of decreasing virtual channel subscripts.

Formally, we define the routing function.

$$R_{SEN}(c_{dxi}, n_j) = \begin{cases} c_{(d-1)0\,\text{rol}(i)} & \text{if } (x=0) \wedge (\text{dig}(d-1, i) = \text{dig}(0, j)), \\ c_{(d-1)1\,\text{rol}(i)} & \text{if } (x=0) \wedge (\text{dig}(d-1, i) \neq \text{dig}(0, j)), \\ c_{d0(i \oplus 1)} & \text{if } (x=1). \end{cases} \quad (6)$$

Assertion 7: If a message is routed on channel c_{d0i} destined for node n_j, then $\forall m \ge d$, $\text{dig}(m, j) = \text{dig}(m, k)$, where k is i rotated left d bits.

Proof: By induction on dimension d. For $d = n - 1$, a message is only routed on the shuffle connection, $x = 0$, of dimension $n - 1$ if $\text{dig}(n-1, i) = \text{dig}(0, j) \Rightarrow \text{dig}(n-1, i) = \text{dig}(n-1, k)$ by the definition of k. Since there is only one possible value for m, the assertion is satisfied.

If the assertion is true for dimension d, then after routing on connection $c_{(d-1)0i}$, the assertion also holds for $d - 1$ by the same argument: a message is only routed on the shuffle connection, $x = 0$, of dimension $d - 1$ if $\text{dig}(d-1, i) = \text{dig}(0, j) \Rightarrow \text{dig}(d-1, i) = \text{dig}(d-1, k)$. ■

Assertion 8: The routing function R_{SEN} correctly routes messages from any node to any other node in a shuffle-exchange network.

Proof: From Assertion 7, after routing on channel c_{00i}, the message will be at its destination. It may reach its destination before this. Since the function routes in order of decreasing channel subscripts and there are a finite number of channels, messages will reach their destinations. ■

Assertion 9: The routing function R_{SEN} on a shuffle-exchange network I is deadlock free.

Proof: Since routing is performed in order of decreasing channel numbers, D is acyclic and the routing is deadlock free. ■

VII. Conclusion

We have shown how a deadlock-free routing algorithm can be constructed for an arbitrary communications network by introducing virtual channels. This technique has been applied to construct deadlock-free routing algorithms for k-ary n-cubes, for cube-connected cycles, and for shuffle-exchange networks.

The use of virtual channels to construct deadlock-free routing functions is motivated by the definition of a routing function that maps $C \times N$ to C, rather than the conventional definition of a routing function that maps $N \times N$ to C. By including C in the domain of the routing function, we explicitly define the dependencies between channels. These dependencies are represented by a channel dependency graph D. A necessary and sufficient condition for deadlock-free routing is that D be acyclic.

To generate a deadlock-free routing algorithm, we restrict the routing by removing edges from D to make D acyclic. If it is not possible to make D acyclic without disconnecting the network, we add edges to D by splitting physical channels into a group of virtual channels. Each group of virtual channels shares a single physical channel; however, each virtual channel requires its own queue. The additional edges provided by the virtual channels make it possible to make D acyclic while keeping I strongly connected.

To develop deadlock-free routing algorithms for specific networks we assign a subscript to each virtual channel using a mixed radix notation. Routing is performed in order of decreasing subscripts. Since the subscripts define a total order on the channels, there are no cyclic dependencies and the routing is deadlock free.

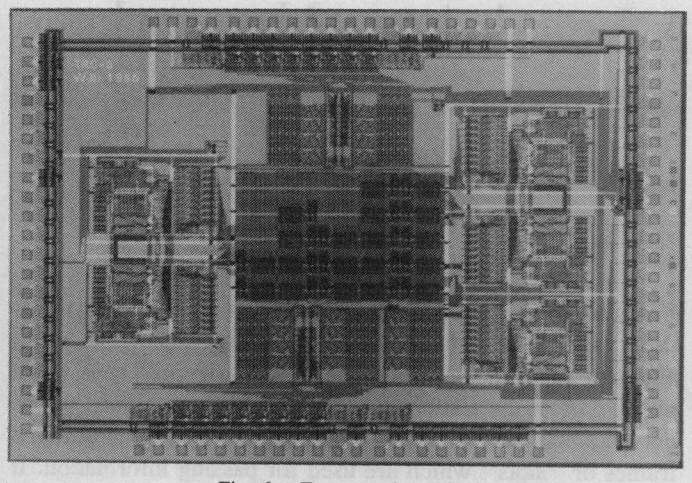


Fig. 6. Torus routing chip.

The cost of implementing virtual channels is small. Each virtual channel requires its own queue, but the queue size can be as small as a single flit.

To demonstrate the concept of virtual channels we have designed the Torus Routing Chip [3], [1], shown in Fig. 6. This chip implements the k-ary n-cube routing function R_{KNC} (4), in hardware. This self-timed VLSI circuit uses wormhole routing, virtual channels, and low-dimension network topology to achieve latencies several orders of magnitude lower than in previous systems.

The availability of deadlock-free routing algorithms encourages the investigation of different interconnection topologies. While $O(\log N)$ diameter networks such as the binary n-cube and the shuffle are attractive because of their richness of interconnection, these networks are almost always embedded in a grid for physical implementation. In keeping with the VLSI imperative of making form fit function, high bandwidth grid interconnections often turn out to be more attractive. Under constant wire bisection, for example, low-dimensional k-ary n-cube networks outperform binary n-cubes [2], [1].

Acknowledgment

The authors thank C. Steele for pointing out problems in an early version of this paper and the referees for their helpful suggestions.

References

[1] W. J. Dally, "A VLSI architecture for concurrent data structures," Ph.D. dissertation, Dep. Comput. Sci., California Instit. Technol., Tech. Rep. 5209:TR:86, 1986.

[2] ——, "On the performance of k-ary n-cube interconnection networks," Dep. Comput. Sci., California Instit. Technol., Tech. Rep. 5228:TR:86, 1986.

[3] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distributed Comput.*, vol. 1, no. 3, 1986.

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

[5] K. D. Gunther, "Prevention of deadlocks in packet-switched data transport systems," *IEEE Trans. Commun.*, vol. COM-29, pp. 512–524, Apr. 1981.

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

[7] L. Kleinrock, *Queueing Systems, Vol. 2.* New York: Wiley, 1976, pp. 438–440.

[8] C. R. Lang, "The extension of object-oriented languages to a homogeneous concurrent architecture," Ph.D. dissertation, California Instit. Technol., 5014:TR:82, pp. 118–124, 1982.

[9] P. M. Merlin and P. J. Schweitzer, "Deadlock avoidance in store-and-forward networks—I: Store-and-forward deadlock," *IEEE Trans. Commun.*, vol. COM-28, pp. 345–354, Mar. 1980.

[10] F. P. Preparata and J. E. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," in *Proc. 20th IEEE Symp. Foundations Comput. Sci.*, pp. 140–147.

[11] C. L. Seitz, "The cosmic cube," *CACM*, vol. 28, pp. 22–33, Jan. 1985.

[12] C. Seitz *et al.*, *Wormhole Chip Project Report*, Winter 1985.

[13] C. L. Seitz, W. C. Athas, W. J. Dally, R. Faucette, A. J. Martin, S. Mattisson, C. S. Steele, and W.-K. Su, *Message-Passing Concurrent Computers: Their Architecture and Programming.* Reading, MA: Addison-Wesley, 1986.

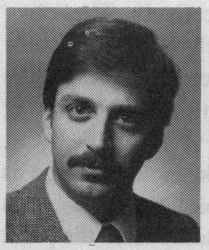
[14] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.

[15] H. Sullivan and T. R. Brashkow, "A large scale homogeneous machine," in *Proc. 4th Annu. Symp. Comput. Architecture*, 1977, pp. 105–124.

[16] A. S. Tanenbaum, *Computer Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1981.

[17] S. Toueg and J. D. Ullman, "Deadlock-free packet switching networks," in *Proc. 11th ACM Symp. Theory Comput.*, 1979, pp. 89–98.

[18] S. Toueg, "Deadlock- and livelock-free packet switching networks," in *Proc. 12th ACM Symp. Theory Comput.*, 1980, pp. 94–99.



William J. Dally (S'78–M'80) received the B.S. degree in electrical engineering from Virginia Polytechnic Institute, Blacksburg, in 1980, the M.S. degree in electrical engineering from Stanford University, Stanford, CA, in 1981, and the Ph.D degree in computer science from California Institute of Technology, Pasadena, in 1986.

From 1980 to 1982 he worked at Bell Telephone Laboratories where he contributed to the design of the BELLMAC-32 microprocessor. From 1982 to 1983 he worked as a consultant in the area of digital systems design. From 1983 to 1986 he was a Research Assistant and then a Research Fellow at Caltech. He is currently an Assistant Professor of Computer Science at the Massachusetts Institute of Technology, Cambridge. His research interests include concurrent computing, computer architecture, computer-aided design, and VLSI design.

Charles L. Seitz (S'68–M'69–M'82) received the B.S., M.S., and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge.

He is now a Professor of Computer Science at the California Institute of Technology, Pasadena, where his research and teaching activities are in the areas of VLSI architecture and design, concurrent computation, and self-timed systems. Prior to joining the Caltech faculty, he worked as a Consultant to Burroughs Corporation from 1972 to 1977, was an Assistant Professor of Computer Science at the University of Utah from 1970 to 1972, and was a member of the Technical Staff of the Evans and Sutherland Computer Corporation from 1969 to 1971. While at the Massachusetts Institute of Technology, he was an Instructor in Electrical Engineering.

Dr. Seitz is a member of the Association for Computing Machinery.