A Necessary and Sufficient Condition for Deadlock-Free Wormhole Routing

Loren Schwiebert and D. N. Jayasimha[1]
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210–1277
{loren,jayasim}@cis.ohio-state.edu

April 29, 1994
Revised: December 10, 1994

[1]Part of this work was done when this author was on leave at NASA Lewis Research Center, Cleveland, OH 44142.

Abstract

An important open problem in wormhole routing has been to find a necessary and sufficient condition for deadlock-free adaptive routing. Recently, Duato has solved this problem for a restricted class of adaptive routing algorithms. In this paper, a necessary and sufficient condition is proposed that can be used for any adaptive or nonadaptive routing algorithm for wormhole routing, as long as only local information is required for routing. The underlying proof technique introduces a new type of dependency graph, the *channel waiting graph*, which omits most channel dependencies that cannot be used to create a deadlock configuration. The necessary and sufficient condition can be applied in a straightforward manner to most routing algorithms. This is illustrated by proving deadlock freedom for a partially adaptive nonminimal mesh routing algorithm that does not require virtual channels and a fully adaptive minimal hypercube routing algorithm with two virtual channels per physical channel. Both routing algorithms are more adaptive than any previously proposed routing algorithm with similar virtual channel requirements.

Keywords: wormhole routing, routing algorithms, deadlock freedom, channel waiting graph, necessary and sufficient condition, mesh architectures, hypercube architectures.

1 Introduction

Wormhole routing [8] has become the switching technique of choice in modern distributed-memory multiprocessors such as the Intel Paragon, the Cray T3D, the MIT J-machine, the Caltech MOSAIC, and the nCUBE-2/3. Implementations of wormhole routing typically divide each message into packets, which are then divided into flits. The header flit of a packet contains the routing information and the data flits of the packet follow the header flit through the network. Since the network treats each packet as a separate message, the term message is used for a single packet. When the header arrives at an intermediate router, the router immediately forwards the message header to a neighboring router if an output channel the message can use is available. The data flits then follow the header flit in a pipelined fashion. If the header is unable to proceed because no appropriate output channel is free, the router buffers only a few flits, rather than the entire message. Since the data flits contain no routing information, messages cannot share channels. Hence, each channel in the path is reserved from the time the header flit acquires the channel until the last flit of the message has traversed the channel. Since the flits of a message are forwarded as soon as possible, the message latency is largely insensitive to the distance between the source and destination. This is especially important for multiprocessors with a high diameter, such as meshes. On the other hand, packet switching buffers the entire message at every intermediate node before forwarding any part of the message. Hence, wormhole routing has lower message latency when there is little or no channel contention. In addition, wormhole routing requires only enough storage at a router to buffer a few flits, rather than the entire packet. These two properties account for the popularity of wormhole routing in distributed-memory multiprocessors. See Ni and McKinley [25] for an in-depth discussion of wormhole routing.

The primary drawbacks to wormhole routing are the message latency and contention that can occur with even moderate network traffic. Whenever a message is unable to proceed due to contention, the header and data flits are not removed from the network. Instead, the message holds all the channels it currently occupies. Since all the channels in the path from the source to the destination are held from the time they are acquired until the entire message has traversed the channel (which is after the entire path has been established except for relatively short messages that fit in the intermediate channel buffers), performance degradation due to contention can be severe and message latency can be unacceptably high. A message that requires several channels can block many messages while being transmitted. These blocked messages can in turn block other messages, which further increases the message latency. Providing additional *physical* channels between nodes in the network can reduce both latency and contention. This is an expensive solution, however, and can actually increase message latency if the routers are pin-out constrained. A more cost-effective method of reducing message latency, proposed by Dally [6], is to allow multiple *virtual* channels to share the same physical channel. Each virtual channel has a separate buffer, with multiple messages multiplexed over the same physical channel. Both latency and contention can be further reduced by using the multiple paths that exist in the network between the source and destination nodes. However, Dally and Seitz [8] have shown that since a message holds channels until the entire message has been transmitted, a routing algorithm with no restrictions on the use of virtual or physical channels can result in deadlock.

The simplest routing algorithms are *nonadaptive* and define a single path between the source and destination. Adaptive routing algorithms, on the other hand, support multiple paths between the source and destination. Adaptive routing algorithms are either minimal or nonminimal. Minimal routing algorithms allow only shortest paths to be chosen, while nonminimal routing algorithms do not require messages to use only shortest paths. Minimal routing algorithms provide higher throughput for high message traffic and are generally simpler to implement. Nonminimal routing algorithms are useful for fault tolerance. Gaughan and Yalamanchili [15] present a good overview of adaptive routing protocols. Whether minimal or nonminimal, adaptive routing algorithms can be further differentiated by the fraction of shortest paths they allow. *Partially adaptive* routing algorithms do not allow *all* messages to use *any* shortest path. *Fully adaptive* routing algorithms *do* allow all messages to use any shortest path. Although all fully adaptive routing algorithms allow a message to use any *physical* channel that is part of a shortest path, different restrictions may be placed on the choice of *virtual* channels on that physical channel. Hence, not all fully adaptive routing algorithms are equivalent. Some fully adaptive routing algorithms allow more adaptiveness than others by placing fewer restrictions on the choice of *virtual* channels.

Since each virtual channel needs a separate buffer and the virtual channels are multiplexed over the physical channel, the number of virtual channels required by an adaptive routing algorithm gives a good approximation of the hardware cost of the router. Routing algorithms that require more virtual channels need additional router control logic and are usually more complex. Multiplexing and scheduling virtual channels on a physical channel is more complicated with additional virtual channels. Router latency and cycle time also increase with the number of virtual channels [3], so fewer virtual channels are generally better. Reducing the number of virtual channels needed for a given degree of adaptiveness is accomplished by using a less restrictive routing algorithm. Conversely, when the same number of virtual channels is used, a less restrictive routing algorithm has better performance than a more restrictive routing algorithm [24].

Recent research on adaptive routing algorithms has partially addressed both of these issues by reducing the virtual channel requirements and imposing fewer restrictions on the virtual channels. A natural question that arises is: Exactly how restrictive must the routing algorithm be to guarantee deadlock freedom? In other words, what is a necessary and sufficient condition for deadlock-free routing? In this paper, we present a theoretical result for minimizing the restrictions imposed for deadlock-free wormhole routing. Besides providing a necessary and sufficient condition for deadlock freedom, the routing algorithms developed using this proof technique are substantially less restrictive than previous routing algorithms. The only restriction we impose on the routing algorithms is that only local information available at the router is used to make the routing decision. In general, routing is done based solely on local information, because of the overhead of transmitting non-local information and the additional router complexity that is required to utilize this information.

2 Previous Work

Designing deadlock-free routing algorithms for wormhole routing was simplified by Dally and Seitz with a proof that an acyclic channel dependency graph guarantees deadlock freedom [8]. Each vertex of the channel dependency graph is a virtual channel. There is a directed edge from one virtual channel to another if a message is permitted to use the second virtual channel immediately after the first. Since the graph is acyclic, deadlock freedom can be shown by assigning a numbering to the edges of the graph, ensuring that all channels are used in strictly increasing or strictly decreasing order.

Dally and Seitz proposed their proof technique for nonadaptive routing algorithms. Nonadaptive routing algorithms can be characterized by functions of the form $R : C \times N \times N \to C$, where the input channel, belonging to the set of channels C, and the current and destination node, belonging to the set of nodes N, define an output channel on which to route the message. An acyclic channel dependency graph has also been used as a basis for developing adaptive routing algorithms defined by relations of the form $R : C \times N \times N \to C^p$, where a set of output channels, rather than a single channel, is defined on which to route the message [2, 4, 5, 9, 16, 17, 18, 20, 23, 28].

Glass and Ni [18] and Boura and Das [2] have proposed methodologies for generating deadlock-free algorithms, but both proof techniques require an acyclic channel dependency graph. Glass and Ni propose a method of analyzing routing algorithms based on the permitted and prohibited dependencies from one channel to another. These dependencies are characterized as turns, with the set of possible turns defined by the topology. For meshes, the turns can be 90° turns (when switching from a channel in one dimension to a channel in a different dimension), 0° turns (when switching from one channel to another channel in the same direction), and 180° turns (when switching from one channel to a channel in the opposite direction of the same dimension). The *turn model* groups the turns into cycles and breaks all cycles by prohibiting some turns. This is equivalent to removing edges from the channel dependency graph. It is then necessary to show that cycles cannot be created from the remaining turns, based on the assumption that an acyclic channel dependency graph is required for deadlock freedom. Boura and Das propose a method of proving deadlock freedom by partitioning the channels into two sets and requiring messages to route completely in the first set before using channels in the second set.

Duato [10, 11] proved that requiring an acyclic channel dependency graph is too restrictive for routing algorithms defined by relations of the form $R : N \times N \to C^p$, where the current node and the destination node, independent of the input channel, define a set of output channels on which to route the message. Cycles are permitted in the channel dependency graph if some subset of channels defines a connected routing subfunction with an acyclic *extended* channel dependency graph. An extended channel dependency graph contains both the direct and the indirect dependencies. Each edge in the channel dependency subgraph defines a *direct* dependency. An *indirect* dependency is a dependency between two channels in the subgraph that exists only because of the intermediate use of one or more channels not in the subgraph. This methodology has been used by Schwiebert and Jayasimha [26] to propose a fully adaptive minimal routing algorithm for meshes. Berman, *et al.* [1] propose a torus routing algorithm with a routing relation of the form $R : C \times N \times N \to C^p$ that allows cyclic dependencies between the channels.

Dally and Aoki [7] prove deadlock freedom for a routing algorithm with cyclic dependencies by guaranteeing an acyclic *packet wait-for graph*. A packet wait-for graph is defined dynamically by the packets in the network and contains an edge from packet p_i to packet p_j if packet p_i is waiting for a channel held by packet p_j.

All these proof techniques provide only a sufficient condition for deadlock-free adaptive routing. Although Dally and Seitz proved that an acyclic channel dependency graph is a necessary and sufficient condition for nonadaptive routing algorithms [8], determining what constitutes a necessary and sufficient condition for *adaptive* routing algorithms has remained an open problem.

Lin, McKinley, and Ni [22] propose a proof technique based on the fact that a routing algorithm is deadlock-free if none of the channels in the network can be held forever. If every message that uses a given channel is guaranteed to reach its destination, no matter which path (of those allowed by the routing algorithm) the message takes, then a deadlock configuration cannot arise from the use of this channel. Since sink channels cannot be part of a deadlock configuration, the proof starts with the sink channels and works backward through the network. If it is possible to show that no channel can be held forever by a message, regardless of the destination and path taken, then the routing algorithm is deadlock-free. This proof technique was proposed as a necessary and sufficient condition, although Duato points out that only sufficiency is proved [12].

Duato [12] has recently proposed a necessary and sufficient condition for proving deadlock freedom for a restricted class of adaptive routing algorithms. This proof technique applies only to routing relations of the form $R : N \times N \rightarrow C^p$. The proof requires the identification of a subset of channels, $C_1 \subseteq C$, which has an acyclic extended channel dependency graph. Unlike the sufficient condition, the set of channels in C_1 can differ for different source-destination pairs. Hence, Duato introduces the notion of cross dependencies. A cross dependency is a dependency from $c_i \in C_1$ to $c_j \in C_1$, where c_i and c_j are both in C_1, but for different source-destination pairs. As with regular dependencies, there are direct cross dependencies and indirect cross dependencies. A routing algorithm is deadlock-free if and only if some connected C_1 exists with no cycles arising from direct, indirect, direct cross, and indirect cross dependencies.

The technique proposed in this paper applies to routing relations of the form $R : C \times N \times N \rightarrow C^p$, while Duato's proof technique is more restrictive and applies only to routing relations of the form $R : N \times N \rightarrow C^p$. The latter routing relations can always be converted to routing relations of the former type by providing the same set of output channels for every input channel that the message could have used to reach that node (including input from the source when the source is the current node). However, in general, routing relations of the form $R : C \times N \times N \rightarrow C^p$ cannot be converted to routing relations of the form $R : N \times N \rightarrow C^p$, since the set of output channels can differ for the same destination when the message arrives on different input channels. In addition to requiring the routing relation to be of the form $R : N \times N \rightarrow C^p$, Duato also imposes two further restrictions on the routing algorithms for which the proof technique is a necessary and sufficient condition, neither of which is imposed by our proof technique. First, the routing algorithm *must* provide a minimal path between every pair of nodes, even for nonminimal routing algorithms. Second, the routing algorithm must be *coherent*. A routing algorithm is coherent if it permits every partial path from any source to any destination to be used by the same source to reach an intermediate node on the path or by an intermediate node on the path to reach the same destination. In other words, a coherent routing algorithm that allows a message from processor n_i to n_j to route through processor n_k, must allow a message to use the partial path between n_i and n_j when routing from processor n_i to processor n_k or from processor n_k to processor n_j. Although requiring coherence appears to be a modest restriction, in section 8 we propose two simple routing algorithms that are not coherent.

3 Assumptions and Definitions

Several assumptions and definitions are introduced to facilitate the presentation of the necessary and sufficient condition. These are standard assumptions made when proving deadlock freedom for wormhole-routed routing algorithms and have also appeared in [8, 11].

1. A node can generate messages of arbitrary length destined for any other node at any rate.

2. A message arriving at its destination is eventually consumed.

3. Since wormhole routing is used, once a channel queue accepts the header flit of a message, it must accept all the flits of the message before accepting any flits from another message.

4. A channel queue cannot contain flits belonging to more than one message at a time. The channel must transmit the tail flit of the current message before the channel queue accepts the header flit of the next message.

5. An available channel queue arbitrates among messages simultaneously requesting the channel. However, messages already waiting on a channel queue are chosen in an order that prevents starvation.

Definition 1 An *interconnection network* I is a strongly connected directed multigraph, $I = G(N, C)$, where the vertices, $n_i \in N$, are the processors and the arcs, $c_i \in C$, are channels that connect neighboring processors. Each arc, c_i, can forward messages from one processor, denoted s_i, to a neighboring processor, denoted d_i.

Definition 2 A *routing algorithm* R_A on interconnection network I is represented by $R_A(n_i, n_j)$ and for each source-destination pair determines the set of possible paths taken by the message. The routing algorithm is generated by repeated application of a routing relation and a selection function at each router between the source and destination of the message. The routing algorithm may be adaptive or nonadaptive; minimal or nonminimal; fault-tolerant or not fault-tolerant.

Definition 3 Routing algorithm R_A is *prefix-closed* if a path that R_A permits from node n_i to node n_j that routes through node n_k implies that R_A also permits the partial path from n_i to n_k when n_k is the destination.

Definition 4 Routing algorithm R_A is *suffix-closed* if a path that R_A permits from node n_i to node n_j that routes through node n_k implies that R_A also permits the partial path from n_k to n_j when n_k is the source. Every routing algorithm with a routing relation of the form $R : N \times N \rightarrow C^p$ is suffix-closed.

Definition 5 Routing algorithm R_A is *coherent* if R_A is prefix-closed and suffix-closed.

Definition 6 A *routing relation* has the form $R : C \times N \times N \rightarrow C^p$ and specifies a set of output channels based on the input channel, the current node, and the destination of the message.

Definition 7 A *selection function* has the form $S : C \times C^p \times F^p \rightarrow C$ and chooses a single output channel from a set of output channels based on the input channel and the status of the output channels. F represents the possible states of an output channel. For fault-tolerant routing, $F = \{\text{free, busy, faulty}\}$; otherwise, $F = \{\text{free, busy}\}$.

Definition 8 A *waiting channel* is a channel at the source or an intermediate node for which the message waits when the message is unable to proceed because every channel the message can use is unavailable. A message may have multiple waiting channels at the source or an intermediate node.

Definition 9 The *channel waiting graph* (CWG) for a given routing algorithm R_A and interconnection network I is a directed graph, $CWG = G(C, E)$. The vertices of CWG are the channels of I. The arcs of CWG are pairs of channels that satisfy the following relationship:

$$E = \{(c_i, c_j) | c_j \text{ is a waiting channel for some message that has used } c_i\}$$

Note: There is no requirement that the message waits for c_j *immediately* after using c_i, only that the message is long enough to fill the channel queues from c_i to c_j. Otherwise, the message cannot hold channel c_i while waiting for c_j. Hence, when analyzing edges in the channel waiting graph, we restrict the examination to messages of sufficient length.

Definition 10 Consider a path from a source to a destination. Routing algorithm R_A is *wait-connected* if for every input channel on a path, there exists a waiting channel through which the message can be routed. In other words, regardless of which input channel the message uses (including the injection channel when the message is at the source), there is always an output channel for which the message can wait. More formally,

$\forall n_i, n_j \in N, \forall c_a \in C$ such that $c_a \in R_A(n_i, n_j)$ and $d_a \neq n_j,$ and for n_i's injection channel (c_a)
$\exists c_b \in C$ such that the message from n_i to n_j can wait for c_b immediately after using c_a

Note: By definition, wormhole routing does not remove a message from the network when the message is unable to proceed. Whenever a message is blocked, it must be possible for the message to wait for at least one output channel. Otherwise, a message is never delivered if it reaches an intermediate node where all the output channels are busy. Therefore, wormhole routing requires a wait-connected routing algorithm.

Definition 11 A *configuration* is an assignment of messages to channels. The header and data flits of each message are stored in the channel queues and each channel queue holds flits from at most one message. The leading channel is the channel the message has most recently acquired and contains the message header. Any other channels occupied by this message contain only data flits. A configuration is *legal* if each message in the configuration occupies one or more consecutive channels and the message header is stored at the head of the leading channel queue that the message occupies; each message occupies only channels the routing algorithm permits the message to use; and the storage capacity of each occupied channel has not been exceeded.

Definition 12 A *deadlock configuration* for routing algorithm R_A on interconnection network I is a non-empty legal configuration consisting of a set of messages, $m_1, m_2, \ldots, m_n, n > 0$, where each message, m_i, in the set has acquired at least one channel. The header flit of m_i has not reached its destination and is unable to proceed because every output channel for m_i is unavailable. Moreover, every waiting channel for m_i is occupied by either data flits of m_i or the header or data flits of another message in the set. The data flits at the head of any other channel queue held by m_i are unable to proceed because the queue of the next channel occupied by m_i is full. Thus, each message is blocked and must wait for an unavailable waiting channel held by another message in the set. Alternatively, when $n = 1$, m_1 waits for a channel already held by itself. The set of messages can be ordered such that:

m_i waits for a channel used by $m_{i+1} \forall i < n$ and

m_n waits for a channel used by m_1

4 Livelock Freedom

Livelock freedom is another issue that arises in nonminimal routing algorithms. Livelock occurs when a message is continually routed from one channel to another, yet never reaches its destination. Since minimal routing always moves the message closer to the destination, livelock is only possible with nonminimal routing. Livelock could occur with a long message that is always routed away from the destination, however, this eventually results in deadlock. If the message length exceeds the storage capacity of *all* the channel queues in the network, the message finally arrives at an intermediate node where the message already occupies every output channel the message is permitted to use. Livelock could also occur when a short message is routed in a cycle of n channels and the message can fit in $n - 1$ channel queues. The message could continually route to the same channel used a previous time through the cycle. However, by assumption 5, this message cannot prevent other messages from using these channels when the channels become free, so this message cannot create a deadlock configuration due solely to the livelock problem. Hence, deadlock freedom can be proved even for routing algorithms that are not livelock-free.

5 Necessary and Sufficient Condition

Most techniques for proving that routing algorithms for wormhole routing are deadlock-free require that the channel dependency graph be acyclic in some manner. The idea of waiting channels was introduced independently by Lin, McKinley, and Ni [22], however, the methodology used in this paper is novel. The channel dependency graph describes the order in which channels can be *used*. However, from definition 12, it is clear that any deadlock configuration is based on the waiting channels, rather than the channels a message could use. The routing algorithm may allow a message to use a channel when the channel is free, even if the message is not permitted to wait for this channel when the channel is busy. This is our motivation for using the CWG, since it ignores dependencies that cannot result in deadlock. Since the channel waiting graph is a subset of the channel dependency graph, requiring an acyclic channel waiting graph is less restrictive than requiring an acyclic channel dependency graph.

Theorem 1 *If routing algorithm R_A is wait-connected and the CWG for R_A is acyclic, then R_A is deadlock-free.*

Proof. R_A is wait-connected, so every message always has a waiting channel when all output channels are busy. Assume there is a deadlock configuration involving n messages. If $n = 1$, then there is a cycle in the CWG from a channel to itself, which is not possible since the CWG is acyclic. Otherwise, ($\forall i < n$) there is an edge in the CWG from every channel used by m_i to the channel used by m_{i+1} for which m_i is waiting (call this channel c'_{i+1}). There is also an edge in the CWG from every channel used by m_n to the channel used by m_1 for which m_n is waiting (call this channel c'_1). Hence, there is an edge in the CWG from c'_i to c'_{i+1} ($\forall i < n$) and from c'_n to c'_1. However, the CWG for R_A is acyclic, so no such set of edges is possible. Therefore, no deadlock configuration exists and R_A is deadlock-free. $\square$

An edge in the CWG requires only the existence of some path from a channel to a waiting channel. The specific intermediate channels used between a channel and a waiting channel are not considered when creating the CWG. It is possible that a cycle arises from the simultaneous use of the same channel by more than one message. For this reason, we divide cycles in the CWG into two classes: False Resource Cycles and True Cycles. A False Resource Cycle is a cycle in the CWG that *requires* at least one channel to be used simultaneously by more than one message in order to create the cycle. Obviously, a False Resource Cycle cannot occur, since this is physically impossible. Therefore, a False Resource Cycle cannot be used to create a deadlock configuration. A True Cycle is a cycle in the CWG that permits every message in the cycle to use different channels. In section 6, we provide a more complete description of False Resource Cycles and present a technique for distinguishing between False Resource Cycles and True Cycles.

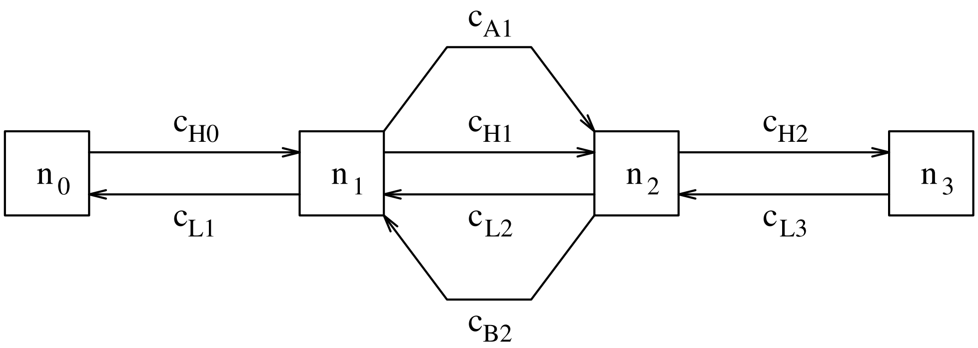

Figure 1: **Duato's Example of an Incoherent Routing Algorithm**

To illustrate the difference between False Resource Cycles and True Cycles, Duato's example [12] of an incoherent routing algorithm is presented. The processors and channels are shown in figure 1. The routing algorithm permits only minimal routing, with the exception of channel c_{B2}. Channel c_{B2} can be used by a message destined for only node n_3. Clearly, this routing algorithm is not coherent, since a message from n_2 to n_3 can be routed through n_1 using channel c_{B2}. However, a message from n_2 to n_1 cannot use channel c_{B2}.

The CWG for this routing algorithm has a False Resource Cycle and a True Cycle. A message whose input channel is c_{B2} can wait for c_{A1} or c_{H1}. If the message waits for c_{A1}, there is a True Cycle from c_{A1} to c_{A1} that uses c_{B2}. Otherwise, the True Cycle is a cycle from c_{H1} to c_{H1} that uses c_{B2}. There is a False Resource Cycle that involves two messages. A message that uses c_{A1} and c_{B2} and waits for c_{H1} and a message that uses c_{H1} and c_{B2} and waits for c_{A1}. Obviously, this False

Resource Cycle exists only because both messages simultaneously use c_{B2}, which is impossible.

A message is unable to proceed when all output channels the message is permitted to use are busy. This situation can be resolved in one of two different ways: (1) The message could wait for a specific output channel to become free or (2) The message could wait until *any* permitted output channel becomes free. For case (1), the routing algorithm cannot choose an arbitrary channel for which to wait, but must choose a channel for which waiting is permitted. Although it is possible that the routing algorithm has a choice of more than one waiting channel, once a waiting channel is chosen, the message must wait for that specific channel to become free. For case (2), the routing algorithm also has the possibility of waiting on just a subset of more than one output channel. In fact, case (2) includes any routing algorithm that does not conform to case (1). That is, any routing algorithm that does not select a specific waiting channel and wait for that channel until it becomes free. We first prove a necessary and sufficient condition for routing algorithms that belong to case (1), followed by a necessary and sufficient condition for routing algorithms that belong to case (2).

Theorem 2 *A routing algorithm, R_A, that requires a message to wait for a specific output channel is deadlock-free iff R_A is wait-connected and the CWG for R_A has no True Cycles.*

Proof. First notice that R_A is wait-connected by definition. From theorem 1, an acyclic CWG is a sufficient condition for deadlock freedom. A False Resource Cycle cannot result in deadlock, so any False Resource Cycles can be ignored. Since there are no True Cycles, the routing algorithm is deadlock-free.

To prove necessity, assume that a True Cycle with n messages exists. A deadlock configuration can be created from this True Cycle. For each $i < n$, allow message m_i to use channel c_i', some additional channels if necessary, and then wait to acquire channel c_{i+1}' in use by message m_{i+1}. (Assume that m_i and c_i' are defined as before.) Similarly, message m_n uses channel c_n' and waits for channel c_1'. Since each message, m_i, in the cycle uses different channels, it is possible to generate a set of messages that are able to use the appropriate channel(s) and then wait for the appropriate channel. To force m_i to wait for the appropriate channel, it is necessary to guarantee that *any* output channel m_i could use is busy. For any output channel available to m_i that is also available to the source, assume the source has injected a message that is using this channel. However, since R_A is of the form $R : C \times N \times N \rightarrow C^p$, it is possible that some of the output channels available to m_i can be used only by messages arriving on the input channel used by m_i. For these output channels, assume that a previous message, m_j, used this input channel and was forwarded on one of the output channels. In addition, the length of m_j is assumed to be short enough that it releases the input channel that m_i uses, however, m_j is long enough that it occupies the output channel at this node. By assumption 2, m_j can remain in the network (and thus hold this output channel) for an indefinite amount of time. Hence, it is always possible to force m_i to wait for c'_{i+1}. Clearly, each message in the set is waiting for a channel used by another message in the set and none of the messages can make progress. Therefore, a deadlock configuration can always be constructed from a True Cycle. $\square$

For routing algorithms that permit a message to wait for *any* of the output channels to become free, an acyclic CWG is not a necessary condition. Since each blocked message can select any output channel, messages may be able to avoid channels that form cycles in the CWG by using an alternative channel that is not part of a cycle. However, deadlock can be avoided only if at least one of the output channels is *guaranteed* to become free. For this reason, we selectively remove edges from the CWG to resolve all True Cycles as long as the routing algorithm for the resulting graph, called CWG', remains wait-connected. We next prove that if no such CWG' exists, then the routing algorithm is not deadlock-free. However, if such a CWG' does exist, then the following theorem can be used to prove deadlock freedom:

Theorem 3 *A routing algorithm, R_A, that permits a message to wait for any output channel is deadlock-free iff R_A is wait-connected for some CWG' and this CWG' has no True Cycles.*

Proof. If R_A is wait-connected for the CWG and the CWG has no True Cycles, then the result follows immediately from theorem 2, with $CWG = CWG'$. Assume the CWG contains True Cycles. In this case, R_A must be wait-connected for some CWG' without True Cycles.

We first prove sufficiency. Consider a potential deadlock configuration for R_A, involving a cycle of n messages ($n > 0$). This requires that every message in the configuration is waiting for a channel held by itself or another message in the cycle. However, the message actually waits for all the output channels. Since R_A is wait-connected for CWG', at least one of the waiting channels for each message is in CWG'. An output channel in CWG' eventually becomes free and some message in the set is forwarded. However, there is no guarantee that the output channel the message, m_i, actually acquires is a channel in CWG'. (It is possible that m_i is forwarded along a different channel before the output channel in CWG' becomes free.) If m_i has reached its destination, then the cycle has been resolved. Otherwise, whether or not m_i acquires a channel in CWG', m_i can acquire an output channel in CWG' at the next router because R_A is wait-connected for CWG'. Hence, one of the messages can always be routed and a deadlock configuration cannot occur.

We now prove necessity by showing that the routing algorithm is not deadlock-free if there is no wait-connected CWG' without True Cycles. No wait-connected CWG' without True Cycles exists, so it is possible to generate a set of messages that have no waiting channel that is guaranteed to become available. Furthermore, these messages are all blocking each other, since otherwise it would be possible to guarantee that a waiting channel does become free. Hence, this set of messages form a deadlock configuration. However, since the routing algorithm is deadlock-free, no deadlock configurations are possible. Therefore, a wait-connected CWG' without True Cycles must exist. $\square$

It is possible to prove that the incoherent routing algorithm previously discussed is deadlock-free by using the necessary and sufficient condition just proposed. The routing algorithm is not deadlock-free if a message waits for a specific channel to become free. Consider two messages, one from n_0 to n_3 and one from n_1 to n_3. Assume the message from n_1 to n_3 uses c_{H1} and c_{H2} and the message from n_0 to n_3 uses c_{H0}, c_{A1}, and c_{B2}. If this second message waits for c_{A1}, then a deadlock configuration occurs. If this message instead waits for c_{H1}, then a deadlock configuration occurs when the first message uses c_{A1} instead of c_{H1} and the second message uses c_{H1} instead of c_{A1}. Since the routing algorithm does not know which channels have already been used by a message when selecting a waiting channel, the routing algorithm is not deadlock-free if the message waits for a specific channel.. On the other hand, if the message waits for both channels c_{A1} and c_{H1}, then by theorem 3 the routing algorithm is deadlock-free. Figure 2 depicts the CWG for this routing algorithm. As discussed previously, there are True Cycles and a False Resource Cycle in this channel waiting graph.

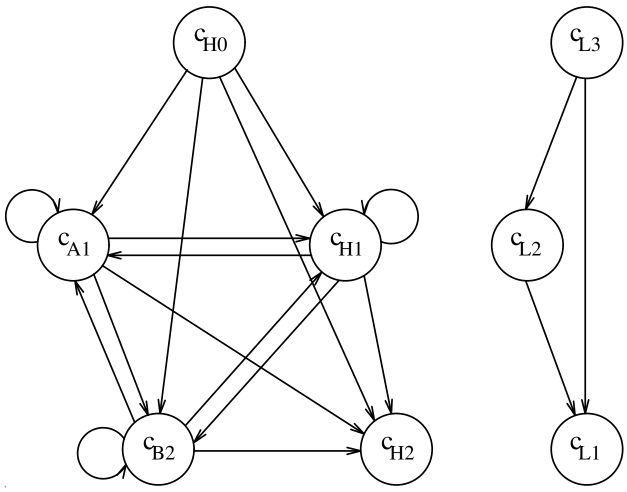

Figure 2: **The CWG for the Incoherent Routing Algorithm**

The CWG has True Cycles, however it is possible to create a CWG' with no True Cycles since the routing algorithm is deadlock-free. Edges are removed from the CWG to create CWG' using the following observations:

- Since any message waiting for c_{H2} is guaranteed to reach n_3 and both c_{B2} and c_{H2} can be used only for messages whose destination is n_3, no message is required to wait for c_{B2}. Therefore, the routing algorithm remains wait-connected for CWG' even if all edges to c_{B2} are removed from the CWG.

- A message that arrives on c_{B2} after using c_{A1} eventually acquires c_{H1}, since the message using c_{H1} is destined for either n_2 or n_3. If the destination is n_2, then the message has reached its destination and c_{H1} will become free. If the destination is n_3, then the message using c_{H1} is either using c_{H2} or waiting for c_{H2}. Since any message waiting for c_{H2} eventually reaches n_3, c_{H1} eventually becomes free. Similarly, a message that arrives on c_{B2} after using c_{H1} eventually acquires c_{A1}. Therefore, the edge from c_{A1} to c_{A1} and the edge from c_{H1} to c_{H1}

can be removed from CWG' to create a new CWG' and the routing algorithm is still wait-connected for this new CWG'.

Figure 3 depicts CWG' after these two modifications have been made. There are no True Cycles in CWG'. The only cycle in CWG' is a False Resource Cycle from c_{A1} to c_{H1} and back to c_{A1}, which requires both messages in the cycle to use c_{B2}. Since R_A is wait-connected for CWG', deadlock freedom follows immediately from theorem 3.

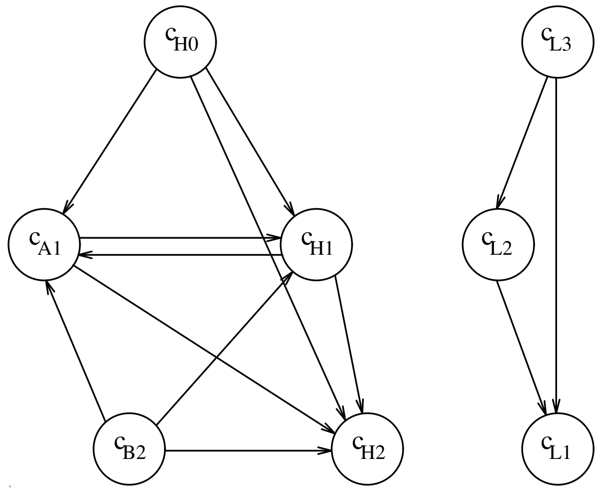

Figure 3: **The final CWG' for the Incoherent Routing Algorithm**

6 False Resource Cycles

The CWG is a static graph, however, the dependencies that arise among the channels are dynamic. False Resource Cycles capture this notion of dynamic dependencies among the channels. *When two edges in the CWG both require the use of a common channel, then these two dependencies cannot occur simultaneously.* Cycles in the CWG that are formed from such dependencies cannot occur in reality, and hence, cannot lead to a deadlock configuration.

False Resource Cycles could arise with minimal or nonminimal routing algorithms. Duato's incoherent routing algorithm is an example of the latter. We now present an example of a False

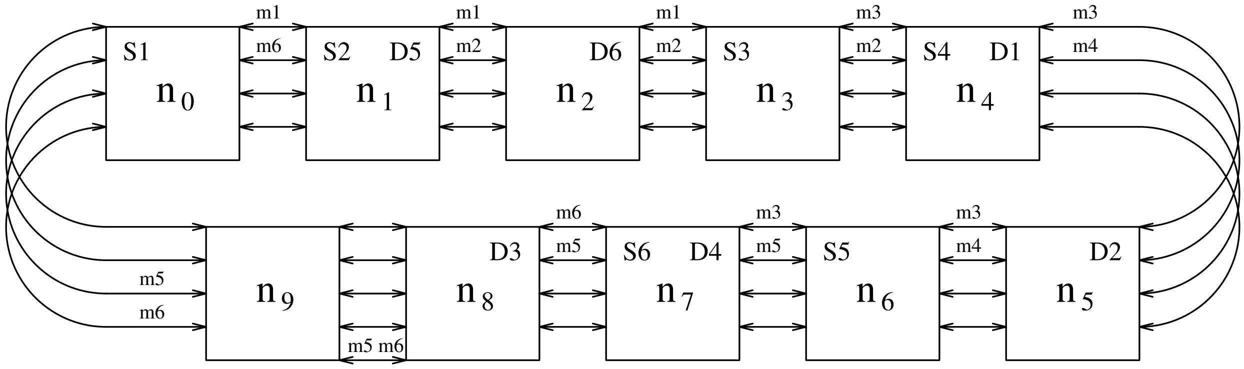

Figure 4: **A False Resource Cycle for Minimal Routing**

Resource Cycle using minimal routing. The routing algorithm is presented for the ring (1D torus) in figure 4. The routing algorithm permits a message to use VC_1 if the destination of the message is an even numbered node and VC_2 if the destination is an odd numbered node. A message on VC_1 or VC_2 stays on VC_1 or VC_2, respectively, with two exceptions: (1) After a message uses a wrap-around channel, the message switches from VC_i to $VC_{i+2 \pmod 4}$ and (2) A message that uses VC_5 at node n_8 uses VC_3 if the destination is an odd numbered node and VC_4 if the destination is an even numbered node. It is possible to create a cycle in the CWG only if two messages arrive at node n_8 and *both* messages leave n_8 on VC_5. Clearly, both messages can acquire VC_5 at the same time, so this is a False Resource Cycle. There are no True Cycles in the CWG, so the routing algorithm is deadlock-free.

In order to create a cycle, each message in the set, m_i, must acquire c'_i before m_{i-1} arrives at c'_i. This is always possible with a True Cycle, since each message uses different channels. A False Resource Cycle, however, requires that at least two messages in the cycle share a channel. A False Resource Cycle can arise in one of two ways. Either a channel in the cycle is shared, or a channel outside the cycle is shared by at least two messages in the cycle prior to entering the cycle. If the shared channel is part of the cycle, then each message, m_i, that uses the shared channel has already acquired c'_i. Hence, the False Resource Cycle would be a True Cycle if m_i could reach c'_{i+1} without using a shared channel. On the other hand, if the shared channel is used prior to the channels in the cycle, then m_i cannot acquire c'_i before m_{i-1}. However, the False Resource Cycle would be a True

Cycle if m_i could reach c_i' without using a shared channel.

The first possibility is a False Resource Cycle involving only the channels that form the cycle. We need to consider channels at only those nodes that are used by more than one message in the cycle. A message that can form its part of the cycle without using any node used by another message in the cycle cannot be using a shared channel. Hence, ignore any message that can route through nodes that no other message in the cycle can use. Also, ignore any message that can route using channels that no other message in the cycle can use, even if other messages share a node with this message. For the remaining messages, see if a channel-disjoint path can be found for each message. This is done by arbitrarily choosing a path for one of the remaining messages. Continue selecting channel-disjoint paths for the other remaining messages. If a channel-disjoint path can be found for each message, then there is no False Resource Cycle formed with the channels used in the cycle. Otherwise, backtrack, selecting a different channel-disjoint path for the previous message. If all possible paths have been considered and there is still no channel-disjoint path for all messages, then this is a False Resource Cycle.

The second possibility is a False Resource Cycle created only when at least two messages share a channel that is used prior to the cycle. This is possible only with routing algorithms that are not suffix-closed, because if a routing algorithm is suffix-closed the cycle can be formed without using channels outside the cycle. For these routing algorithms, first, ignore any message, m_i, that can route from c_i' to c_{i+1}' without using any channels before c_i'. Next, ignore any message that could route to c_i' by using nodes that no other message in the cycle can use. Also, ignore any message that can route using channels that no other message in the cycle can use. For the remaining messages, consider the paths that a message could take to reach c_i'. Proceed as in the previous case, selecting channel-disjoint paths and backtracking as necessary. If a channel-disjoint path can be found for each message, then this is a True Cycle. Otherwise, this cycle may be a False Resource Cycle. We do not have an algorithmic method of determining this.

A message that uses a shared channel outside the cycle can be short enough to release this channel after it has been used and hold only the channels that form this message's part of the cycle. Hence, even though the channel is shared by more than one message in the cycle, the channel could be shared consecutively rather than only simultaneously. The cycle is a True Cycle if the cycle can be formed even when all the shared channels are used consecutively rather than simultaneously. However, if even one shared channel must be used simultaneously by more than one message in the cycle, then this is a False Resource Cycle.

7 Identifying CWG'

In this section, a formal design methodology for reducing the CWG to CWG' is described. This methodology is needed only for routing algorithms that do not require a message to wait for a specific output channel. The design methodology requires the identification of all cycles in the CWG. Since the problem of counting the number of cycles in a directed graph is NP-hard [14], an exponential time algorithm is used for finding CWG'. Other general techniques for proving deadlock freedom also require exponential time in the worst case [10, 12, 22]. The algorithm for reducing the CWG to CWG' has the following steps:

1. Create a list, L, of all cycles in the CWG. Each entry in L contains three sets: s_1, the set of edges that form the cycle, s_2,the set of edges in the cycle the algorithm has attempted to remove, and s_3, the set of edges that are currently removed from the cycle. Initially, $s_2 = s_3 = \emptyset$ for all cycles in L.

2. Remove all False Resource Cycles from L. This is done by examining each cycle in L using the method described in section 6 or by informally examining CWG. Then number the True Cycles $TC_i (i = 1, 2, \ldots)$ and let $i = 1$.

3. Remove an edge from TC_i as long as the routing algorithm remains wait-connected. Any edge that cannot be removed with the routing algorithm still wait-connected is added to s_2.

4. If no edge in TC_i can be removed with the routing algorithm still wait-connected, then backtrack to TC_{i-1}. Before backtracking, reset $s_2 = s_3 = \emptyset$ for TC_i.

5. Otherwise, TC_i can be resolved, so add this edge to s_2 and s_3 for every cycle in L that contains this edge. Increment i and select a True Cycle, TC_i, from L with $s_3 = \emptyset$.

6. Repeat steps 3 – 5 until all True Cycles in L have been resolved or the routing algorithm has backtracked to TC_1 and the algorithm has already attempted to remove every edge TC_1. If all True Cycles in L have been resolved, then the routing algorithm is deadlock-free and CWG' consists of the edges that have not been removed. Otherwise, no acyclic CWG' exists for this routing algorithm, so the routing algorithm is not deadlock-free.

8 Routing Examples

In order to demonstrate the usefulness of the necessary and sufficient condition, a partially adaptive nonminimal routing algorithm for n-dimensional meshes is proposed. The routing algorithm does not require any virtual channels and does not have an acyclic channel dependency graph. Our proof technique is also used to show deadlock freedom for a fully adaptive minimal hypercube routing algorithm. The necessary and sufficient condition is then used to prove that any relaxation of the restrictions imposed by the routing algorithm introduces the possibility of deadlock. Deadlock freedom cannot be proved for either routing algorithm using Duato's necessary and sufficient condition.

8.1 Previous Routing Algorithms

Torus and hypercube topologies can be characterized as k-ary n-cubes, where k is the radix and n is the dimension. For example, an 8D hypercube is a 2-ary 8-cube and a 16×16 torus is a 16-ary 2-cube. An n-dimensional mesh is similar to a torus, except a mesh does not have wrap-around channels.

Partially adaptive routing algorithms for wormhole-routed hypercubes and meshes have been proposed by many authors [2, 4, 9, 16, 18, 21, 28]. Boura and Das [2] as well as Glass and Ni [16, 18] have proposed methods for producing partially adaptive routing algorithms for hypercubes and meshes. The partially adaptive routing algorithm for hypercubes proposed by Li [21] has the additional advantage of ensuring that the multiple paths are edge-disjoint for many source-destination pairs. All three algorithms require only a single virtual channel per physical channel. In section 8.2, we show how to increase the adaptiveness in a mesh without virtual channels by using a routing algorithm that permits cycles in the channel dependency graph.

The hypercube routing algorithm proposed by Draper and Ghosh [9] uses two virtual channels for each physical channel. Each message routes in dimension order along the first set of channels, but may skip some dimensions in which the message needs to route. The message then routes in dimension order along the second set of channels. The message can no longer skip dimensions and must wait for the channels to become free. The hypercube routing algorithm proposed by Yang and Tsai [28] also requires two virtual channels per physical channel. A message first uses any dimension in which it needs to route in a positive direction. When the message finishes with all such dimensions or finds them all busy, the message repeats this process for all negative directions. The message then switches to the second set of virtual channels and routes first in all remaining positive directions and then all negative directions, waiting for busy channels when necessary.

A fully adaptive hypercube routing algorithm has been proposed by several authors [10, 19, 22, 27]. This routing algorithm requires two virtual channels per physical channel. A message routes in dimension order along the first set of virtual channels. Each message also has the possibility of routing in any dimension that moves the message closer to the destination along the second set of virtual channels.

Besides routing algorithms proposed specifically for hypercubes, any routing algorithm for arbitrary dimension meshes or tori can be extended to hypercubes, since an n-dimensional hypercube is a special case of an n-dimensional mesh or torus. Only a few fully adaptive routing algorithms have been designed for n-dimensional mesh and torus topologies [1, 7, 20, 23, 26]. When restricted to hypercubes, the routing algorithms proposed by Jesshope, Miller, and Yantchev [20], and by Linder and Harden [23] require more virtual channels than the routing algorithm proposed by Duato [10]. The routing algorithm proposed by Dally and Aoki [7] is more restrictive than the routing algorithm proposed by Duato. The routing algorithm proposed by Berman, *et al.* [1], when restricted to hypercubes, is equivalent to the routing algorithm proposed by Duato. The routing algorithm proposed by Schwiebert and Jayasimha [26], when modified for hypercubes, is more adaptive than the routing algorithm proposed by Duato.

The routing algorithm proposed by Duato [10] uses the first set of virtual channels for nonadaptive routing and the second set of virtual channels for fully adaptive routing. In section 8.3, we show how to increase adaptiveness by using the first set of virtual channels for partially adaptive routing.

8.2 Mesh Routing Algorithm

We now propose a partially adaptive nonminimal routing algorithm for meshes, which does not require an acyclic channel dependency graph and does not require any virtual channels. (For a 2D mesh, this routing algorithm is similar to *north-last*, proposed by Glass and Ni [18], although our routing algorithm permits messages to make more use of $180°$ turns.) This new routing algorithm, called the Highest Positive Last routing algorithm, is defined below.

Let h be the highest dimension in which the message still needs to route in the *negative* direction to reach the destination. The following set of restrictions is applied to the channels:

- A message can use any channel in a dimension lower than h, even if the message does not need to route in that direction, so nonminimal routing is permitted. A message can also route in the negative direction of dimension h.

- A message that needs to route in only positive directions must use the positive channels in increasing dimension order, although the message is permitted to misroute in the negative direction of a higher dimension if desired.

- A message is allowed to make a $180°$ turn from the negative direction to the positive direction of the same dimension when the message needs to route in the positive direction. A message is permitted to make a $180°$ turn from the positive direction to the negative direction of the same dimension only when the message needs to route in the negative direction of this dimension and some higher dimension.

- If all output channels a message can use are busy, the message waits for the channel in the negative direction of dimension h. If the message needs to route only in positive directions, then the message waits for the channel in the positive direction of the lowest dimension in which the message needs to route.

The Highest Positive Last routing algorithm has a routing relation of the form $R : C \times N \times N \rightarrow C^p$. For example, consider a message that needs to route only East and North. When the message is due South of its destination, the message is permitted to route South if the input channel was East. However, the message cannot route South if the input channel was North. (If this restriction is not imposed, the routing algorithm is not deadlock-free.) Also, routing algorithms of this type are not coherent [13], because a message can route West and immediately route East. The message has now routed through the same processor twice. However, a routing algorithm is not coherent if a message can route through any processor more than once. Furthermore, the routing algorithm is not coherent even for minimal paths. For example, on an $8 \times 8 \times 8$ mesh, a message from $(4, 4, 2)$ to $(5, 5, 1)$ can route through $(4, 5, 2)$. However, a message from $(4, 4, 2)$ to $(5, 5, 2)$ cannot route through $(4, 5, 2)$. Finally, the routing algorithm does not have an acyclic channel dependency graph, but does have an acyclic channel waiting graph. For example, a message that needs to route in the negative direction of the Z dimension can use the channels in the X and Y dimensions in any order, but always waits for the channel in the negative direction of the Z dimension. Duato's proof technique requires the existence of an acyclic extended channel dependency graph for some connected subset of the channels [12]. Since there is just one channel between neighboring nodes and the channel dependency graph has cycles, Duato's proof technique cannot be applied to the Highest Positive Last routing algorithm.

The routing algorithm is substantially more adaptive than any previously proposed mesh routing algorithm that requires only one channel. Previous partially adaptive routing algorithms have required an acyclic channel dependency graph. For example, Glass and Ni [18] propose the *negative-first* routing algorithm for n-dimensional meshes. The negative-first routing algorithm requires messages to route in all negative directions and then all positive directions. The negative-first routing algorithm also permits nonminimal routing on the negative channels before using any positive channels. Glass and Ni prove that an acyclic channel dependency graph prohibits at least $n(n-1)$

90° turns for an n-dimensional mesh. With negative-first routing, a message cannot use a negative channel after using a positive channel, so each of the n positive channels is prohibited from using the negative channel in the other $n-1$ dimensions. The Highest Positive Last routing algorithm prevents a message from using a channel in a lower dimension after using the positive channel in a higher dimension, which restricts $n(n-1)$ 90° turns. However, the restrictions imposed by the Highest Positive Last routing algorithm are not absolute. A message can use a positive channel before a channel in a lower dimension whenever the message needs to route in the negative direction of some still higher dimension. This is a significant relaxation of the routing restrictions, especially on the positive channel in the lower dimensions. Although the positive channel in dimension i cannot be used before the positive or negative channel in a lower dimension, this restriction is removed when the message also needs to route in the negative direction of any dimension higher than i. The lower the dimension, the more likely this restriction can be avoided.

Theorem 4 *The Highest Positive Last routing algorithm is deadlock-free.*

Proof. Since the routing algorithm requires a message to wait for a specific output channel, the necessary and sufficient condition proved in theorem 2 is used to prove deadlock freedom. Any cycle in the CWG requires at least two dimensions, since a message routing in the positive direction cannot *wait for* a channel in the negative direction without first routing in the negative direction of another dimension. Consider any potential cycle in the CWG involving the channels used by the messages that form the cycle. It is possible to partition this cycle into two hyperplanes by dividing the cycle along the highest dimension used in the cycle. For the cycle to exist, some message must use a channel in the positive direction of the highest dimension in the cycle and then wait for a channel in the positive half of the cycle. In other words, some message must cross the partition in the positive direction and then wait for a channel in this half of the partition for the cycle to exist. However, a message is not permitted to use a channel in the positive direction and then wait for a channel in a lower dimension without first using a channel in the negative direction of a higher dimension. Since the message has already used the positive direction of the highest dimension in the cycle, the message has not used the negative channel in a still higher dimension and this channel is not held by any other message in the set. Thus, no message in the cycle can satisfy this requirement and the CWG is acyclic. Deadlock freedom follows immediately from theorem 2. $\square$

8.3 Hypercube Routing Algorithm

The following conventions are used in this section. Channels are assumed to be bidirectional virtual channels. $VC^i_{n\pm}$ is used to denote virtual channel n in the $\pm$ direction of dimension i. For example, VC^3_{1-} is virtual channel one in the negative direction of the third dimension. An asterisk in the superscript denotes all dimensions. Thus, VC^*_{2-} denotes the second virtual channel in the negative direction of all dimensions.

Each processor of an n-dimensional hypercube can be uniquely labeled using an n bit string. The source of a message is denoted $S = (s_{n-1}, s_{n-2}, \ldots, s_1, s_0)$ and the destination is denoted $D = (d_{n-1}, d_{n-2}, \ldots, d_1, d_0)$. A message routes from the source to the destination by routing in dimensions in which the corresponding bit in the source differs from that of the destination. The message routes in the *positive* direction of dimension i if $s_i = 0$ and $d_i = 1$. Similarly, the message routes in the *negative* direction of dimension i if $s_i = 1$ and $d_i = 0$. With minimal routing, a message routes in each dimension at most once and a message does not route in dimension i if $s_i = d_i$.

We now propose a fully adaptive minimal routing algorithm for hypercubes. All previous fully adaptive routing algorithms specifically for hypercubes require that the first set of virtual channels be used only for nonadaptive routing. This new routing algorithm permits the first set of virtual channels to be used for partially adaptive routing. The routing algorithm, called the Enhanced Fully Adaptive routing algorithm, is defined by the following steps:

- Assign two bidirectional virtual channels to each dimension.
- Allow a message to route along the second virtual channel at any time.

Let l be the lowest dimension in which the message still needs to route. The following set of restrictions is applied to the first set of virtual channels:

- A message that needs to route in the negative direction of dimension l can use any of the first set of virtual channels.
- A message that needs to route in the positive direction of dimension l *must* use VC^l_{1+}.
- If all output channels a message can use are busy, the message waits for VC^l_1.

The routing algorithm is substantially more adaptive than any previously proposed fully adaptive hypercube routing algorithm. The Enhanced Fully Adaptive routing algorithm restricts a message from using the first virtual channel in the positive direction after using the first virtual channel in a higher dimension. However, a message can use the first virtual channel in a higher dimension first whenever the message needs to route in the negative direction of the lowest dimension in which the message still needs to route. This is a significant relaxation of the routing restrictions, especially compared with using dimension order routing for the first set of virtual channels.

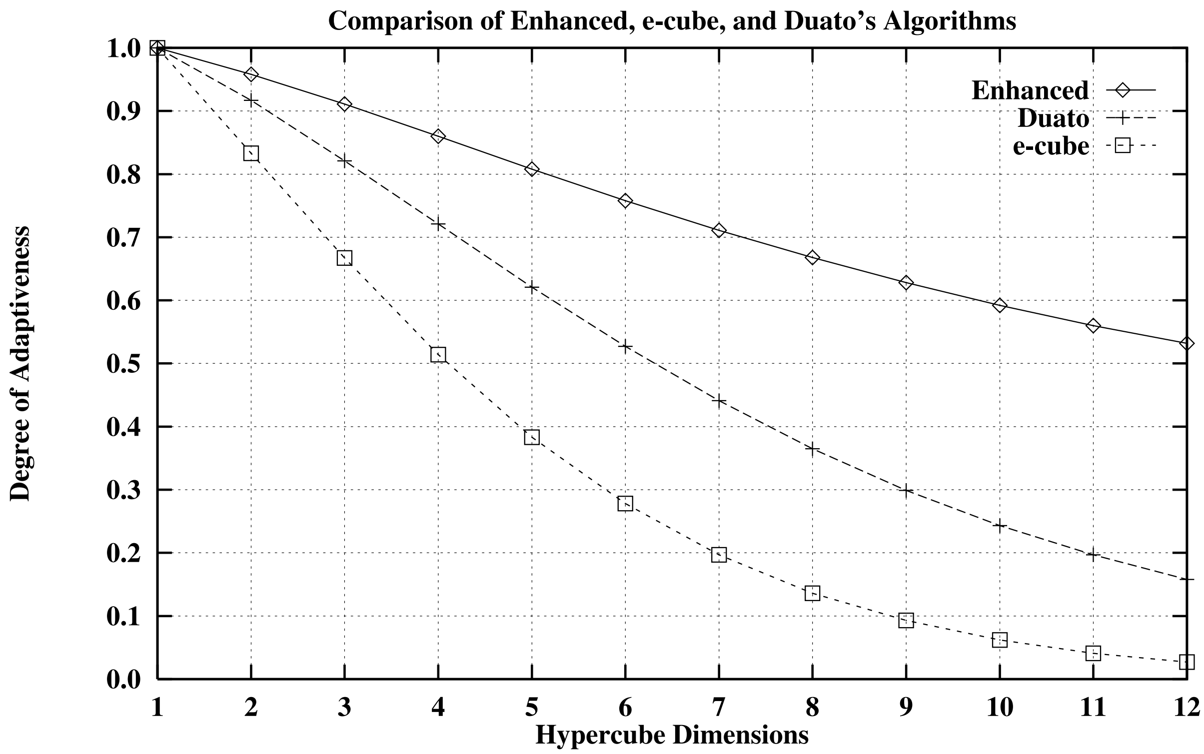

Figure 5: **Degree of Adaptiveness for Hypercube Routing Algorithms**

The degree of adaptiveness is the proportion of virtual paths permitted by the routing algorithm, compared to a routing algorithm with no restrictions, averaged over all source-destination pairs [18]. The degree of adaptiveness for both routing algorithms is shown in figure 5. For comparison, the degree of adaptiveness for e-cube (nonadaptive dimension order) routing is also shown. Paradoxically, the degree of adaptiveness is not zero for nonadaptive routing. This occurs because nonadaptive routing always allows one of paths and the degree of adaptiveness would be zero only if there were *no* permitted paths. There are relatively few paths when the source is near the destination, so even a single path can represent a relative large proportion of the paths. For example, when the distance between the source and destination is two, nonadaptive routing can use half the paths.

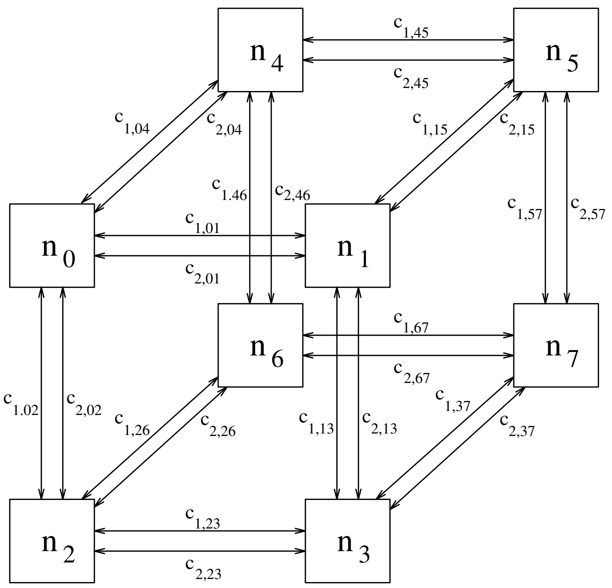

Figure 6: **3D Hypercube with 2 Virtual Channels per Physical Channel**

Duato's necessary and sufficient condition [12] cannot be used to prove that the Enhanced Fully Adaptive routing algorithm is deadlock-free. Duato's proof technique requires the existence of an acyclic extended channel dependency graph for some connected subset of the channels. Regardless of which channels are chosen, there are cycles formed by direct dependencies among the channels. Consider the 3-dimensional hypercube in figure 6. The routing algorithm places no restrictions on channel usage for messages routing from n_1 to n_7, from n_3 to n_5 and from n_7 to n_1. Also, a message from n_5 to n_2 can use any channel between n_5 and n_3 before it routes from n_3 to n_2. Of course, this is not a counter-example to Duato's necessary and sufficient condition, since the Enhanced Fully Adaptive routing algorithm is not coherent. For example, a message from n_5 to n_2 can route through n_1 using the first virtual channel. However, a message from n_5 to n_3 cannot route through n_1 using the first virtual channel.

Theorem 5 *The Enhanced Fully Adaptive routing algorithm is deadlock-free.*

Proof. This proof uses the necessary and sufficient condition proved in theorem 2. R_A is wait-connected, since a message is always permitted to wait for virtual channel one in the lowest dimension in which the message needs to route. Since a message can wait for only VC_1^*, any cycle in the CWG must be created from waiting dependencies among the VC_1^* channels (although VC_2^* channels could be used to create these dependencies). Since minimal routing is used, any cycle in the CWG requires at least two dimensions and must use both directions of each dimension in the cycle. Let l be the lowest dimension of the potential cycle. A cycle in the CWG requires one of two situations: either a message, m_i, waits for VC_{1+}^l after using VC_1^* in a higher dimension of the cycle or m_i uses VC_{1+}^l or VC_{2+}^l after using VC_1^* in a higher dimension and then waits for VC_1^* in a different higher dimension. Since l is the lowest dimension in the cycle, m_i does not need to route in any dimension lower than l after using VC_1^* in a higher dimension. Otherwise m_i would be waiting for a channel in this lower dimension and l would not be the lowest dimension in the cycle. Clearly, neither situation can occur, since m_i cannot use VC_1^* in a higher dimension when m_i needs to route in VC_+^l and l is the lowest dimension in which m_i needs to route. Therefore, the routing algorithm is deadlock-free. $\square$

Theorem 6 *No restrictions imposed on the Enhanced Fully Adaptive routing algorithm can be relaxed without permitting a deadlock configuration.*

Proof. There are no restrictions on the use of the second set of virtual channels, so the restrictions on only the first set of virtual channels must be considered. The only restriction on the first set of virtual channels is that a channel in a higher dimension cannot be used by a message that needs to route in the positive direction of the lowest dimension in which the message needs to route. Without loss of generality, let l be the lowest dimension and assume that the channel used in a higher dimension is the first virtual channel in the positive direction of dimension i. Then there is an edge in the CWG from VC^i_{1+} to VC^l_{1+}. The CWG already has edges from VC^l_{1+} to VC^i_{1-}, from VC^i_{1-} to VC^l_{1-}, and from VC^l_{1-} to VC^i_{1+}. These edges form a True Cycle. A message waits for only VC^l_1 and the routing algorithm is not deadlock-free if VC^l_1 is not a waiting channel. Thus, from theorem 2, the routing algorithm is not longer deadlock-free. □

9 Conclusion

A necessary and sufficient condition for proving deadlock freedom for both nonadaptive and adaptive routing algorithms using wormhole routing has been proposed. The usefulness of this proof technique has been demonstrated with a partially adaptive nonminimal routing algorithm for n-dimensional meshes. This routing algorithm is more adaptive than any other routing algorithm proposed for meshes without additional channels. The proof technique described in this paper has also been used to prove deadlock freedom for a fully adaptive minimal routing algorithm for hypercubes. This new hypercube routing algorithm is substantially more adaptive than any previous fully adaptive routing algorithm for hypercubes. The necessary and sufficient condition can also be used to develop adaptive routing algorithms for the torus and other topologies.

We have shown that the restrictions on the Enhanced Fully Adaptive routing algorithm cannot be relaxed without creating a deadlock configuration. It is possible, although unlikely, that a routing algorithm with a different set of restrictions could be deadlock-free while being less restrictive. The number of restrictions cannot be reduced, however, since there is only one restriction for each pair of dimensions.

A partial result has been given for distinguishing between False Resource Cycles and True Cycles. This technique can be applied to any suffix-closed routing algorithm, which represents the all but a very restricted class of routing algorithms. In practice, the identification of False Resource Cycles should not require resorting to this formal technique. A formal approach is provided, however, for cases where the irregularity of the interconnection network or the routing algorithm prevent a straightforward analysis.

The proof of theorem 3 requires the identification of CWG'. This is needed only for routing algorithms that do not require a message to wait for a specific output channel. In general, it should be straightforward to determine CWG' for regular topologies such as k-ary n-cubes and meshes. Most existing topologies are regular, and this regularity is usually incorporated into the routing algorithm. However, a formal design methodology has been provided for those cases where it is difficult to reduce the CWG to CWG'. This automates the task of proving deadlock freedom and should be of use for routing algorithm designers.

Acknowledgments

The authors thank José Duato for many helpful suggestions. The authors also thank Yu-Chee Tseng for insightful comments on the idea of channel waiting; Jeff May for valuable discussions concerning the necessary and sufficient condition; and Kant Patel, Dave Lutz, and Anish Arora for many thoughtful comments which have improved the quality of the paper.

References

[1] P. Berman, L. Gravano, G. Pifarré, and J. Sanz. Adaptive Deadlock- and Livelock-Free Routing With All Minimal Paths in Torus Networks. In *Proceedings of the* 4^{th} *Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 3–12, 1992.

[2] Y. M. Boura and C. R. Das. A Class of Partially Adaptive Routing Algorithms for n_dimensional Meshes. In *International Conference on Parallel Processing*, volume III, pages 175–182, 1993.

[3] A. A. Chien. A Cost and Speed Model for k-ary n-cube Wormhole Routers. In *Hot Interconnects '93*, August 1993.

[4] A. A. Chien and J. H. Kim. Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors. In *Proceedings of the* 19^{th} *Annual International Symposium on Computer Architecture*, pages 268–277, 1992.

[5] W. J. Dally. Fine-Grain Message-Passing Concurrent Computers. In *Proceedings of the Third Conference on Hypercube Concurrent Computers*, volume 1, pages 2–12, 1988.

[6] W. J. Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[7] W. J. Dally and H. Aoki. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[8] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[9] J. T. Draper and J. Ghosh. Multipath E-Cube Algorithms (MECA) for Adaptive Wormhole Routing and Broadcasting in k-ary n-cubes. In *International Parallel Processing Symposium*, pages 407–410, 1992.

[10] J. Duato. On the Design of Deadlock-Free Adaptive Routing Algorithms for Multicomputers: Design Methodologies. In *Parallel Architectures and Languages Europe 91*, volume I, pages 390–405, 1991.

[11] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[12] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. In *International Conference on Parallel Processing*, volume I, pages 142–149, 1994.

[13] J. Duato. Private Communication, May 1994.

[14] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, New York, NY, 1979.

[15] P. T. Gaughan and S. Yalamanchili. Adaptive Routing Protocols for Hypercube Interconnection Networks. *IEEE Computer*, 26(5):12–23, May 1993.

[16] C. Glass and L. M. Ni. Adaptive Routing in Mesh-Connected Networks. In 12^{th} *International Conference on Distributed Computing Systems*, pages 12–19, 1992.

[17] C. Glass and L. M. Ni. Maximally Fully Adaptive Routing in 2D Meshes. In *International Conference on Parallel Processing*, volume I, pages 101–104, 1992.

[18] C. Glass and L. M. Ni. The Turn Model for Adaptive Routing. In *Proceedings of the* 19^{th} *Annual International Symposium on Computer Architecture*, pages 278–287, 1992.

[19] L. Gravano, G. Pifarré, G. Denicolay, and J. Sanz. Adaptive Deadlock-free Worm-hole Routing in Hypercubes. In *International Parallel Processing Symposium*, pages 512–515, 1992.

[20] C. R. Jesshope, P. R. Miller, and J. T. Yantchev. High Performance Communications in Processor Networks. In *Proceedings of the* 16^{th} *Annual International Symposium on Computer Architecture*, pages 150–157, 1989.

[21] Q. Li. Minimum Deadlock-Free Message Routing Restrictions in Binary Hypercubes. *Journal of Parallel and Distributed Computing*, 15(2):153–159, June 1992.

[22] X. Lin, P. K. McKinley, and L. M. Ni. The Message Flow Model for Routing in Wormhole-Routed Networks. In *International Conference on Parallel Processing*, volume I, pages 294–297, 1993.

[23] D. H. Linder and J. C. Harden. An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes. *IEEE Transactions on Computers*, 40(1):2–12, January 1991.

[24] J. May, D. N. Jayasimha, and K. Patel. Comparison of Multiplexing Schemes for Wormhole-Routed Distributed Memory Multiprocessors. In 1^{st} *International Workshop on Parallel Processing*, December 1994. To appear.

[25] L. M. Ni and P. K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, 26(2):62–76, February 1993.

[26] L. Schwiebert and D. N. Jayasimha. Optimal Fully Adaptive Wormhole Routing for Meshes. In *Supercomputing '93*, pages 782–791, 1993. An extended version of this result to appear in the Journal of Parallel and Distributed Computing.

[27] C.-C. Su and K. G. Shin. Adaptive Deadlock-Free Routing in Multicomputers Using Only One Extra Virtual Channel. In *International Conference on Parallel Processing*, volume I, pages 227–231, 1993.

[28] C. S. Yang and Y. M. Tsai. Fault-Tolerant and Deadlock-Free Wormhole Routing in the Hypercube Network. In *International Conference on Parallel and Distributed Systems*, pages 9–16, 1992.