Techniques for Efficient Network Utilization in Multiprocessor/Multicomputer Systems

Yong Ho Song and Timothy Mark Pinkston

CENG Technical Report 01-01

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562
(213) 740-4475

January 2001

Efficient Handling of Message-Dependent Deadlock in Multiprocessor/Multicomputer Systems

Yong Ho Song and Timothy Mark Pinkston
SMART Interconnects Group
Department of Electrical Engineering – Systems
University of Southern California
Los Angeles, CA 90089-2562, USA
E-mail: {yongho,tpink}@charity.usc.edu

Abstract

Handling deadlocks is essential for providing reliable communication paths between processing nodes in multiprocessor/multicomputer systems. The existence of multiple message types and associated inter-message dependencies may cause message-dependent deadlock in networks that are designed to be free of routing deadlock. Most methods currently used for dealing with message-dependent deadlocks require more system resources than are necessary and/or do not use system resources efficiently. This may have an adverse effect on system performance if resources are scarce. In this paper, we evaluate different approaches for handling message-dependent deadlocks, and we propose an alternative technique based on progressive deadlock recovery. Results show that the proposed technique relaxes restrictions considerably, allowing the routing of packets and the handling of message-dependent deadlocks to be much more efficient–particularly when network resources are scarce.

1 Introduction

In multiprocessor/multicomputer systems, efficient and reliable communication amongst processing nodes is crucial for achieving high performance. However, deadlock anomalies occurring as a result of cyclic hold-and-wait dependencies by messages on network resources reduce communication efficiency and reliability, consequently degrading network and system performance considerably. Thus, it is important to guard against deadlock in such a way as not to impose overly restrictive measures that can under-utilize network resources.

Recent research [1, 2, 3, 4, 5] has focused on the development of very efficient network routing techniques that either avoid or recover from routing-dependent deadlock,

but these techniques assume that messages[1] in the network always sink upon arrival at their destinations. That is, it is assumed that the delivery of messages is not coupled in any way to the injection (generation) or reception (consumption) of any other message in the network or at network endpoints. This simplifying assumption is valid for networks with homogeneous message types, but it inaccurately represents network behavior when heterogeneous messages are routed in which dependencies between different message types exist. Deadlock-free routing algorithms designed using that assumption may provide efficient and deadlock-free communication paths between network endpoints (thus eliminating routing-dependent deadlocks), however they are still susceptible to deadlocks arising from the interactions and dependencies created at network endpoints, between different message types.

Data transactions in multiprocessor/multicomputer systems consist of various types of messages. The most generic message types used to exchange information between communicating entities are *request* and *reply*. In addition to these, many other message types may be defined by the communication protocol of the system. At any given end node in the system, there can be a coupling referred to as *message dependencies*. There may not be a direct or indirect coupling between all message types, and each message type may have several message sub-types for which there is no coupling. However, a distinct class of message dependency is created for each pair of message types for which a direct coupling exists and is transferred to network resources.[2]

A *message dependency chain* represents a partially (or totally) ordered list of message dependencies allowed by the communication protocol. We define the partial order relation "$\prec$" between two message types m_1 and m_2 by the following: $m_1 \prec m_2$ if and only if m_2 can be generated by a node receiving m_1 for some data transaction. Message type m_2 is said to be *subordinate* to m_1, and all message types subordinate to m_2 are also subordinate to m_1. The final message type at the end of the message dependency chain is said to be a *terminating* message type. The number of message types allowed within a message dependency chain is referred to as the *chain length*. For example, if for all data transactions a system defines only two message types, *request* and *reply*, and the message types establish the dependency relation *request* $\prec$ *reply*, then the chain length is two.

Message dependencies occurring at network endpoints (i.e., on injection and reception resources) may prevent messages from sinking at their destinations. When they are added to the complete set of resource dependencies, knotted cycles [6] may form

[1] Messages can be divided into packets. However, as both are routable units of information, there is no distinction between the two regarding deadlock. Therefore, both terms are used interchangeably.

[2] The class of message dependency created by any pair of message sub-types composing two message types is the same as that created by the two message types.

along escape resources [1], resulting in possible deadlock. We refer to such deadlocks as *message-dependent deadlocks*. As is the case for routing-dependent deadlock, approaches for addressing message-dependent deadlock can be based either on avoidance or on recovery. The primary distinction between these approaches is the trade-off made between routing freedom and deadlock formation. The advantages of techniques based on these approaches, therefore, depend on how frequently deadlocks occur and how efficient (in terms of resource cost and utilization) messages can be routed while guarding against deadlocks. Our previous work [7] shows that message-dependent deadlocks occur very infrequently even when network resources are scarce. This motivates us to investigate alternative message-dependent deadlock handling techniques which are less restrictive in the common case.

In this paper, we propose a new technique for handling message-dependent deadlock that is based on progressive deadlock recovery. We also evaluate various approaches for handling them, including the proposed technique. Analysis is performed by simulating CC-NUMA[3] systems with communication protocols similar to the S-1 Multiprocessor [8, 9], Origin2000 Multiprocessor [10], and the Alpha-21364 processor [11]. Using benchmark application traces, we measure the actual message traffic loads generated during execution as well as the frequency of message-dependent deadlocks. The deadlock statistics gathered allow us to gain insight into the severity of this problem in realistic environments. The traffic statistics gathered from these benchmark applications are used later as references for the interpretation of some other simulation results obtained using synthetic traffic loads. Synthetically generated traffic loads are used to stress the network in order to obtain a broader evaluation of the various message-dependent deadlock handling techniques. Critical network and network interface parameters are varied across the simulations to observe their effect on performance.

The remainder of this paper is organized as follows. Section 2 reviews schemes for avoiding message-dependent deadlocks, Section 3 proposes a new scheme based on progressive deadlock recovery. Section 4 presents evaluation methodology and performance results. Related work is briefly discussed in Section 5 and, finally, conclusions are given in Section 6.

[3]CC-NUMA stands for cache coherent non-uniform memory architecture.

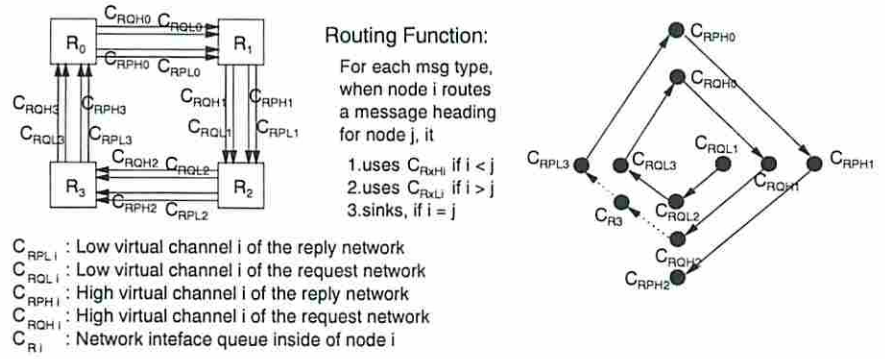

Figure 1: Separation of request and reply networks avoids cyclic dependencies in the channel dependency graph, but it reduces channel utilization.

2 Handling Message-Dependent Deadlocks

2.1 Strict Deadlock Avoidance

Message-dependent deadlock can be avoided by enforcing routing restrictions on network resources used to escape deadlock such that all dependencies on those resources, including message dependencies, are acyclic [1, 12]. Alternatively, they can be avoided while allowing cyclic dependencies on escape resources by requiring some subset of escape resources to be large enough such that they can never become fully occupied. Since sufficient resources and/or routing restrictions on a set of resources always prevent the formation of deadlock, these techniques for handling deadlock are said to be based on *deadlock avoidance*. The second technique can be implemented by providing enough buffer space in each node's network interface message queues to hold at least as many messages as can be supplied as in [13, 14, 15]. Although simple to implement, this technique is not very scalable since the size of the message queues grows as $O(P \times M)$ messages, where P is the number of processor nodes and M is the number of outstanding messages allowed by each node.

The first technique for avoiding message-dependent deadlock is more scalable and more commonly used. One way of guaranteeing acyclic dependencies on escape resources is to provide logically independent communication networks for each message type, implemented as either physical or virtual channels [16, 17, 18, 19]. The partial ordering on message dependencies defined by the communication protocol is transferred to the logical networks so that the usage of network resources is acyclic. Figure 1 illustrates this for the example of a four node ring system which allows a message dependency chain length of two, i.e., $request \prec reply$.

With this technique, the size of network resources does not influence deadlock properties, but at least as many logical networks are required as the length of the message

dependency chain. For example, the Cavallino router/network-interface chip [19] (which can be used to build networks with thousands of nodes) has network interface message queues of less than 1536 bytes each but requires two logically separated networks to handle request–reply dependency. Such partitioning of network resources decreases potential resource utilization and overall performance, particularly when message dependencies are abundant and resources (i.e., virtual channels) are scarce.

For example, consider a system which supports a message dependency chain length of four such that $m_1 \prec m_2 \prec m_3 \prec m_4$. Two virtual channels are required for each message type m_i to escape from routing-dependent deadlocks in a torus network [20]. A total of eight virtual channels are required to escape from message-dependent deadlock, and only one of these is potentially available to each message. If sixteen virtual channels were implemented, only three would be available to each message. In general, the availability of virtual channels is limited to $(1 + (C/L - E_r))$ such that $C \geq E_m$, where L is the message dependency chain length, E_r is the minimum number of virtual channels required to escape from routing-dependent deadlock for a given network, $E_m = L \times E_r$ is the minimum number of virtual channels required to escape from message-dependent deadlock (including routing-dependent deadlocks) for a given network, and C is the number of virtual channels implemented. Channel availability can be increased if all channels other than the minimum number required to escape from message-dependent deadlock are shared amongst all message types, as proposed in [21]. That is, the upper limit on virtual channel availability is increased to $(1 + (C - E_m))$. Nevertheless, restrictions enforced on escape channels (i.e., only one channel out of E_m is available) limits overall potential channel utilization to well below 100%.

Evidently, the main disadvantage of avoiding deadlock by disallowing cyclic dependencies on escape resources is the number of partitioned logical networks required. It is possible to reduce the number of partitions by allowing different message types to use the same logical network and removing message(s) from cyclic dependencies only when a potential deadlock situation is detected. Since a detection mechanism and recovery action are required to resolve the potential deadlock situation, this technique for handling message-dependent deadlock is said to be based on *deadlock recovery*.

2.2 Deadlock Recovery

There are many ways in which potential message-dependent deadlock situations can be detected and resolved. They can be detected at a node's network interface when three conditions are met, as in [10, 22, 23]: (1) both the input and output queues allocated to a message type and its subordinate message type fill up beyond a threshold value, (2)

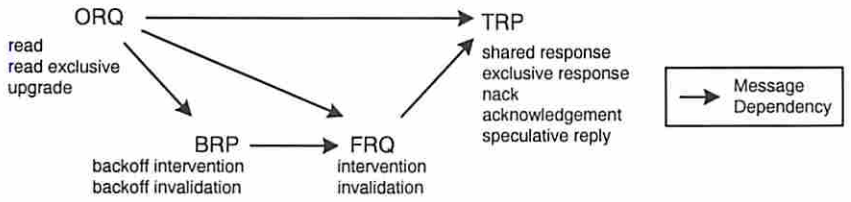

Figure 2: The total ordering among message types (shown in uppercase) and message sub-types (shown in lower case) in the Origin 2000. Note that BRP occurs only if a message-dependent deadlock is detected, otherwise the maximum chain length is three.

the message type at the head of the input queue is one that generates a non-terminating message type, and (3) the first two conditions continue for more than a threshold time-out period. They can be resolved by killing and later re-injecting messages (i.e., *regressive recovery*), deflecting messages out of resources involved in cyclic dependency by converting them from a non-terminating type to a terminating type (i.e., *deflective recovery*), or by progressively routing messages using resources along a path that is guaranteed to sink (i.e., *progressive recovery*). The actions taken by regressive and deflective recovery increase the number of messages needed to complete each data transaction, whereas progressive recovery does not—all packets always make progress toward their destinations and never regeneratively regress or deflect.

An example of a CC-NUMA system which uses the detection mechanism and deflective recovery technique discussed above is the Origin2000 multiprocessor [10]. The cache coherence protocol permits message dependency chains shown in Figure 2. An original request message (ORQ) arriving at a home node is forwarded to the owner or sharers as a forwarded-request message (FRQ) before being responded to by a terminating reply message (TRP) if the home node is unable to fulfill the request. Thus, *in the absence of deadlock*, the length of the message dependency chain can be two ($ORQ \prec TRP$) or three ($ORQ \prec FRQ \prec TRP$). Deadlocks can be strictly avoided by partitioning resources into three separate logical networks, one for each message type. However, to reduce the number of logical networks to two, the system allows both ORQ and FRQ messages to use the same *request network* and TRP messages to use a *reply network*. Message-dependent deadlocks can now potentially form on the request network, which must be resolved once detected.

If a potential deadlock situation is detected at a home node, the node takes ORQ messages that would generate FRQ messages off from the head of the input request queue and deflects back to the requesters backoff-reply (BRP) messages.[4] These mes-

[4]In the DASH system, the removal of ORP messages from the queue head continues until one is found

sages contain owner or sharer information that allows the requester to generate a FRQ message(s) directly to the intended target(s) without further intervention from the home node. They are additional messages needed to carry out the data transaction. That is, the $ORQ \prec FRQ \prec TRP$ message dependency chain is converted into a $ORQ \prec BRP \prec FRQ \prec TRP$ chain *only when potential message-dependent deadlock is detected.* Although the message dependency chain length is increased during recovery, the number of logical networks implemented need not increase. The system allows BRP messages to use the same reply network as TRP messages and strictly avoids message-dependent deadlock on the reply network using the second avoidance technique: space in the input reply queue of the requester is preallocated for the responses of all outstanding ORQ messages.

As can be seen with the Origin2000, an appropriate combination of deadlock avoidance and recovery can reduce the amount of network resources (logical networks in the case of Origin2000) required to handle deadlock, as compared to strictly avoiding deadlock. This allows network resources to be used more efficiently and increases communication performance. However, as potential deadlock situations have been shown to occur mainly when the system nears a saturated state [7, 24], resolving potential deadlock situations by increasing the number of messages required to complete data transactions only exacerbates the problem. This is the case for regressive "abort-and-retry" recovery and deflective "backoff" recovery. Like these techniques, progressive recovery efficiently utilizes network resources in the common case of there being no message-dependent deadlock by relaxing routing restrictions that strictly avoid them. However, unlike these techniques, it resolves potential deadlock situations more efficiently by using the same number of messages as that required to strictly avoid deadlock. Below, one such progressive message-dependent deadlock recovery technique is proposed.

3 The Proposed Technique

The technique proposed for handling message-dependent deadlock is derived from a progressive deadlock recovery technique proposed previously for handling routing-dependent deadlocks, called *Disha Sequential* [25, 5]. In *Disha Sequential*, each router has a centralized flit-sized deadlock buffer (DB) used to progressively route potentially deadlocked messages once detected. Only one message is allowed to use the set of deadlock buffers at any given time, implemented by a token passing and capture mechanism. A circulating token visits all nodes, is captured by a node containing a potentially deadlocked message, and is released once the message reaches its destination. With this mechanism, the routing function defined on the set of deadlock buffers provides a connected and deadlock-free

that would generate a TRP or until the output request queue is below its threshold value.

recovery path to/from any two network endpoints. The *Disha Sequential* scheme is applicable to all networks and is proved to safely recover from all potential routing-dependent deadlocks [5].

The proposed technique extends the notion of a *Disha*-like recovery path *between* network endpoints to one that *includes* network endpoints. Hence, the circulating token must also visit all network interfaces attached to each router node, and a deadlock buffer (referred to as a deadlock message buffer or DMB) must also be provided in each network interface. The size of the DMB is determined by the minimum unit of information on which end-to-end error detection/protection (i.e., ECC, checksum) is performed. Typically, this is at the packet level, requiring these deadlock buffers to be at least packet-sized. This is also the minimum size of the network interface input and output message queues; however, larger input/output queues are typically used to increase performance. All network resources can be completely shared by all message types. That is, network virtual channels between network endpoints and network interface input/output queues at network endpoints can be independent of message type.[5] Relaxing resource allocation and routing restrictions in this way maximizes the utilization of resources but does not strictly prevent message-dependent deadlock from potentially forming.

A block diagram of the network interface architecture needed to support this recovery technique is depicted in Figure 3. When the memory controller needs to send a response message to n individual destination nodes, it is assumed to generate n messages each for point-to-point communication instead of one message for multicast communication. It is also assumed that the memory controller processes a message only if the output message queue in the network interface provides a sufficient amount of free space for the subordinate message(s). When the memory controller of a node produces an output message for which one or more subordinate messages are expected to return to the node, it preallocates internal resources (e.g., missing status handling registers or MSHRs implemented in the lockup-free cache) in order to sink the subordinate messages successfully, as is typically done in real systems.

Potential message-dependent deadlocks can be detected at network interfaces as described previously. Once detected, a potential deadlock situation is resolved following the *Extended Disha Sequential* recovery procedure shown in Figure 4. After the circulating token is captured at the network interface, the non-terminating message type at the head of the input queue m_i is processed by the memory controller. The generated subordinate message type m_j ($m_i \prec m_j$) is put into the DMB at the network interface and routed over the recovery lane to its destination node's DMB with the token. The source and

[5]This does not preclude the separation of resources to prevent some higher priority message types from being blocked behind other message types.

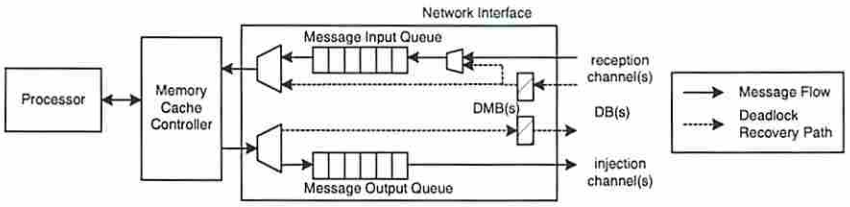

Figure 3: Network interface for progressive recovery of message-dependent deadlocks.

the destination of the message are referred to as a *token sender* and a *token receiver*, respectively.

To prevent the resources for deadlock recovery from being involved in deadlocked, the message arriving at the DMB of its destination should be guaranteed to leave the DMB. Hence, if the input queue at the destination is full, the memory controller is preempted after it completes its current operation and processes the message. If the processed message m_j is a terminating message type or if it is non-terminating and the newly generated subordinate message m_k ($m_j \prec m_k$) is successfully put into the output queue, the token is returned to the token sender over the DB lane. Otherwise, the recovery process continues by putting the newly generated message type m_k into the DMB at that node for routing over the recovery lane. In this case, the token will be reused to deliver m_k through the set of DBs, where the node becomes a token sender with respect to the subordinate message type.

This recovery process continues throughout the length of the message dependency chain until either the subordinate message is delivered to an output queue or an input queuethe, or terminating message type is eventually generated, which will be consumed directly by a memory controller or by an input queue. Each token receiver returns the token to the associated token sender. If the memory controller of a token receiver generates multiple subordinate messages for the message type being processed (i.e., $m_i \prec m_{j_1}, m_{j_2} ... m_{j_n}$), the node repeatedly uses the token in order to deliver each subordinate message m_{j_m} ($1 \leq m \leq n$) to its destination before returning the token to that node's sender. All potentially deadlocked messages that undergo this progressive recovery procedure are said to be *rescued*. If the token returns to the node which initiated the recovery process and the node has no more messages to deliver with the token, the deadlock recovery process ends. At this point, the token is released for re-circulation. A proof that the *Extended Disha Sequential* technique safely recovers from all potential message-dependent deadlocks is given in the Appendix by extending the theory provided in [5, 26].

Extended Disha Sequential maximizes routing freedom and resource utilization. However, the token presents a single point-of-failure, and only one message-dependent deadlock

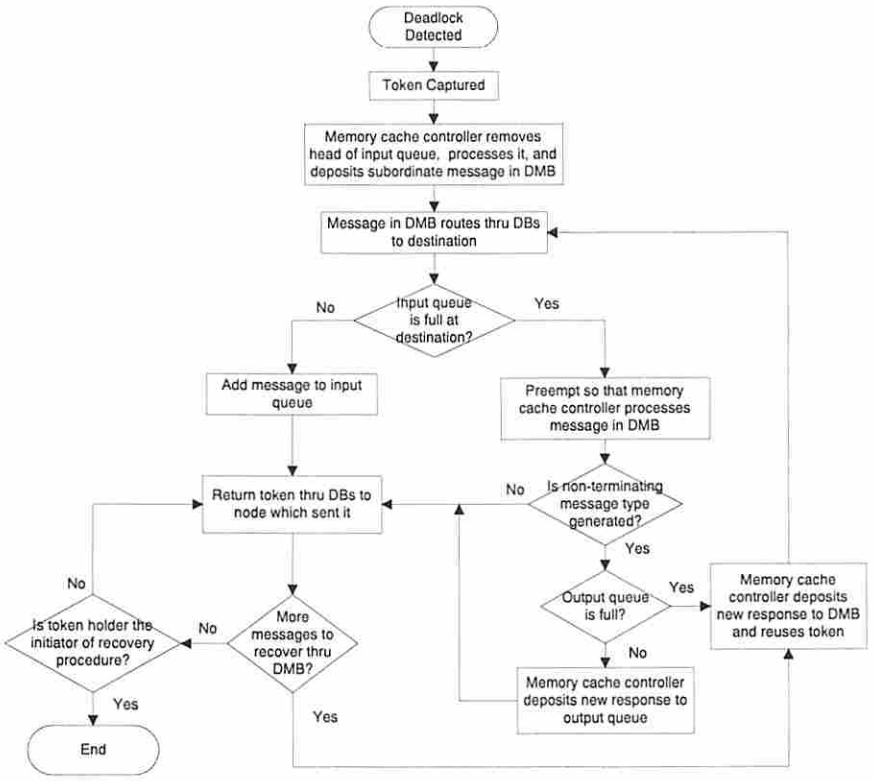

Figure 4: Flowchart for message-dependent deadlock handling based on progressive deadlock recovery.

can be resolved at a time. In dealing with the first shortcoming, it is necessary to have a reliable token management mechanism. For instance, the token can be transmitted as a control packet multiplexed over network bandwidth with data packets. The path taken by the token can be logical and, thus, configurable as opposed to being hardwired in order to increase reliability. The second shortcoming is not a problem as long as the frequency of message-dependent deadlocks is low. Section 4.2 confirms previous work [7] which indicates that message-dependent deadlocks rarely occur.

The proposed technique differs from its predecessor (*Disha Sequential*) in the following ways. First, it extends the set of recovery resources to include network endpoints, i.e., the network interfaces. Second, the token path includes network endpoints, and the token can be captured either by a network interface or by a network router. Third, if the token is captured by a network interface, it is released for re-circulation by the same network interface. Finally, the token may be reused during the rescue of a potentially deadlocked message to deliver its subordinate message(s).

4 Characterization and Performance Evaluation

4.1 Simulation Methodology

Empirical analysis is performed using FlexSim 1.2 [24], a flit-level network simulator developed by the *SMART* Interconnects Group at USC. The simulator performs flit-level traffic flow within the interconnection network and maintains data structures that represent resource allocations and dependencies (resource wait-for relationships) occurring within the network.

Deadlock detection based on the channel wait-for graph(CWG) model implemented in FlexSim 1.2 is augmented to include message-level activities and dependencies in network interfaces. The CWG-based deadlock detection identifies all the cycles in CWG to examine the existence of knots [26] (deadlocks) every 50 cycles. However, this approach suffers from an explosive increase in the number of CWG cycles as network load increases. A newly added feature allows end nodes to detect potential deadlocks using locally available information when the number of CWG cycles is prohibitively large. A deadlock is presumed to have occurred if both the input and output message queues at a node remain full for more than a threshold value, T network cycles, without making any progress. A threshold of 25 cycles is assumed since detection using the CWG method typically takes 25 cycles on average. Deadlock frequency is measured in relative terms by the normalized number of deadlocks—which is the ratio of the number of deadlocks to the number of messages delivered.

Two classes of simulation input sets are used for this work: execution traces and synthetic traffic. We first simulate network models with execution traces of benchmark applications to measure the actual amount of message traffic in the network and the frequency of message-dependent deadlocks generated throughout the execution. Unlike synthesized simulation inputs, real application traces have dynamic and sometimes unpredictable access patterns that might drive the system into deadlock situations. Simulations with application traces give us not only realistic references on the interpretation of simulation results based on synthetic traffic, but also insight on the severity of the deadlock problem in realistic environments. However, these simulations oftentimes do not stress the network enough to sufficiently analyze the behavior of a particular anomaly being studied. In the particular case of deadlocks, usually the network must near its saturation point before deadlocks occur. It is, therefore, useful to conduct simulations driven by both trace as well as synthesized traffic loads.

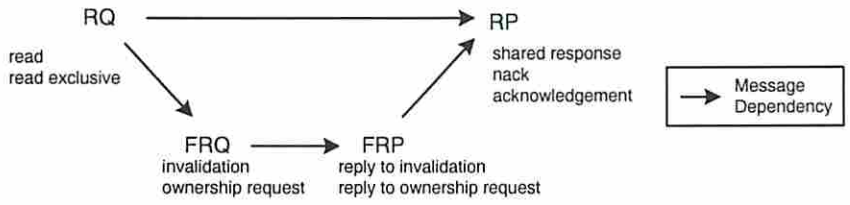

Figure 5: The total ordering among message types (shown in uppercase) and message subtypes (shown in lower case) for application-driven experiments (MSI cache coherence).

4.2 Characterization using Application Traces

4.2.1 Simulation Environment

Traces are gathered through execution-driven simulation and are used to drive our network simulator. A trace contains a full set of data access activities to L1 cache by each processor in RSIM [27], an execution-driven simulator with enabled ILP features and full-mapped, invalidation-based directory cache coherence protocol capability. All data accesses of each application are recorded into a trace file along with timing information in order to preserve traffic burstiness properties over the network. The simulation traces are gathered from four applications in the Splash-2 benchmark suite (FFT, LU, Radix and Water). The actual amount of network traffic is measured by counting the number and frequency of messages injected into the network. This is later used to analyze and compare the results of simulations with synthetic loads. The benchmark applications are run with the following default parameters: 16 processors, 16 byte request header, 4 bytes per flit, 64 byte cache line, 64 KByte cache size and release memory consistency model.

To simulate data access behavior in CC-NUMA systems, FlexSim incorporates a three-state MSI cache coherence protocol with full-mapped directory (see Figure 5).[6] The following default settings are used for trace-driven simulation: 4×4 torus network, channel queue size of 2 flits, input and output message queues of size 16 messages, and four virtual channels per physical channel. Routing dependent deadlocks are strictly avoided using Duato's protocol [28], thus isolating message dependent-deadlock events if they occur.

4.2.2 Characterization Results

The load rate distributions of the simulated benchmark applications are shown in Figure 6. All benchmarks except for Radix have very low network loads that are insufficient to drive the network to saturation (see corresponding figures for synthetic traffic loads shown in

[6]The message dependencies of this protocol are similar to those of the S-1 Multiprocessor [8] which uses the Censier and Feautrier cache coherence protocol [9].

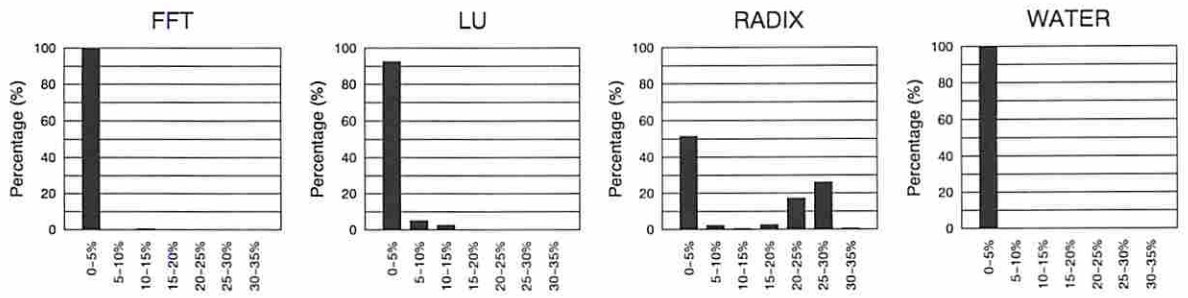

Figure 6: Load rate distributions of the benchmark applications.

Section 4.3.2). For FFT, LU and Water, the network load remains under 5% of total network capacity for more than 92-99% of the execution time. Only Radix generates network loads up to 30% of network capacity, but under 5% for about 50% of the entire execution time.

Table 1 shows the distributions of response types to request messages for the four Splash benchmark applications. *Direct Reply* is the case when the home node sends a reply message directly back to the requester. *Invalidation* and *Forwarding* are the cases when the home node sends a forwarded request message(s) to the sharer and/or the owner of the requested data block respectively. The reply to the forwarded request is sent to the home where a reply message is sent to the requester. The maximum message dependency chain length is 4, but the average length is a little more than 2 for all benchmarks except Water.

During the simulations, *no applications experienced message dependent deadlocks.* This is expected since network load is so low. For Radix, the average network load measures 19.4% of network capacity which is typically much lower than the network saturation point. Simulation results based on synthetic traffic patterns obtained in previous work indicate that a network experiences no message-dependent deadlocks until it reaches deep saturation [7]. This being the case, network load is increased by doubling and quadrupling the bristling factor such that 2 and 4 nodes share the same router in 2×4 and 2×2 torus networks, respectively. However, *no deadlock was observed with the bristled networks for all applications.* For Radix, average network load still remains under 27% and 33% of the total network capacity when the network is bristled by a factor of 2 and 4, respectively. This is near the saturation point, but not enough to drive the network into deep saturation.

Table 1: Types and frequencies of responses to request messages.

Application	Direct Reply	Invalidation	Forwarding
FFT	98.7%	0.9%	0.4%
LU	96.5%	3.0%	0.5%
Radix	95.5%	3.6%	0.8%
Water	15.2%	50.1%	34.7%

4.3 Performance Evaluation using Synthetic Loads

4.3.1 Simulation Environment

Regular k-ary n-cube toroidal networks with the default parameters given in Table 2 are simulated unless specified otherwise. The simulator generates request messages—the first message type in all message dependency chains—at the rate specified as a simulation parameter. All other subordinate message types are generated automatically upon completion of servicing messages at end-nodes. Three message-dependent deadlock handling techniques are evaluated: a strict avoidance (SA) technique similar to that used in the Alpha 21364 processor, the deflective recovery (DR) technique used in the Origin2000 multiprocessor, and our proposed progressive recovery (PR) technique.

Table 2: Default simulation parameters for FlexSim.

Parameters	Values
Network Topology	8×8 torus
Link Transmission	Full-duplex
Switch Technique	Wormhole
Message Length	4 (Request) / 20 (Reply) flits
Bristling Factor	1 processor/node
Virtual Channels per Link	4 virtual channels
Flit Buffers per Channel	2 flits
Message Types	2 (Request and Reply)
Message Service Time	40 clocks
Message Traffic Patterns	Random
Message Queue Size	16 messages

For SA, both message-dependent deadlocks and routing-dependent deadlocks are strictly avoided by providing logically separate sets of escape channels and message queues for each message type. Resource dependencies are allowed in only one direction, i.e., from a message type to its subordinate message type(s). For DR, routing-dependent deadlocks are strictly avoided but message-dependent deadlocks are possible since only two sets of escape channels and message queues are used—one for a *request* network and the other for

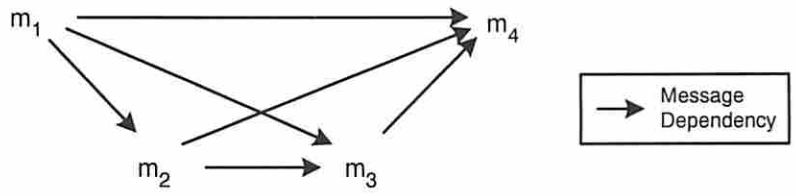

Figure 7: The ordering among message types (m_i) for the generic cache coherence protocol used in the experiments driven by synthetic traffic loads.

a *reply* network, as discussed in Section 2.2. For PR, both types of deadlocks are possible since True Fully Adaptive Routing [5] is used. This allows unrestricted use of all channel and message queue resources, as discussed in Section 3. For both SA and DR, Dimension Order Routing [20] is used unless enough virtual channels are provided to allow for adaptive routing using Duato's Protocol [1], i.e., $C > E_m$ for SA or $C > 2 \times E_r$ for DR. When a message-dependent deadlock is detected in DR or PR, minimum recovery action is taken, which consists of recovering only one message from deadlocked resources.

Various combinations of message dependencies and chain lengths are simulated to mimic the behavior of various cache coherence protocols. The possible message dependencies chains for the Origin2000 and S-1 multiprocessors are shown in Figures 2 and 5. A generic message dependency chain that allows chain lengths of two, three, and four is shown in Figure 7, which is what we use in our analysis. As can be seen, the cache coherence protocol of the Origin2000, S-1 (and MSI), and any other architecture with chain lengths no greater that four can easily be mapped to this generic protocol. For instance, the Origin2000 protocol has $m_1 = ORQ$, $m_2 = BRP$, $m_3 = FRQ$, and $m_4 = TRP$. The S-1 (and MSI) protocol has $m_1 = RQ$, $m_2 = FRQ$, $m_3 = FRP$, and $m_4 = RP$.

The frequency of use of each message type depends on the cache coherence protocol, the status of requested memory blocks in memory (i.e., at the home directory), and the behavior of the application. To generalize and simplify our experiments, we assume five possible message type distributions (data transaction patterns) as shown in Table 3. PAT100 represents a protocol with only two message types (i.e., as in message passing distributed memory systems) or a protocol with greater than two message types in which all blocks are owned by the home node (i.e., in a shared memory system). The first three Splash-2 applications exhibit behavior close to this, e.g., chain lengths of two 95-99% of the time. Patterns PAT721 through PAT271 represent a protocol with four message types in which some blocks are owned or shared by nodes other than the home node to varying degrees. The Water benchmark is an example application exhibiting similar behavior, e.g., chain length of two 15% of the time. PAT280 represents an Origin2000-like protocol in

which no message-dependent deadlocks are detected. In this case, chain lengths are of at most three message types. For each of the patterns with chain lengths greater than two, it is assumed that there is only one sharer node for each block in a shared state; more sharers could be modeled with the effect of increasing the network load.

Table 3: Simulated message type distributions (data transaction patterns.

Traffic Patterns	Dependency Chain Lengths			Message Type Distribution			
	2	3	4	m_1	m_2	m_3	m_4
PAT100	100%	0%	0%	50.0%	0%	0%	50.0%
PAT721	70%	20%	10%	47.7%	12.4%	4.2%	47.7%
PAT451	40%	50%	10%	37.1%	22.1%	3.7%	37.1%
PAT271	20%	70%	10%	34.5%	27.6%	3.4%	34.5%
PAT280	20%	80%	0%	35.7%	0%	28.6%	35.7%

Simulations are run for network loads up to a point just beyond saturation, which is also the point at which deadlocks typically start to form, as indicated in [7, 24]. Each run lasts for 30,000 simulation cycles beyond steady state. Performance results are plotted in Burton Normal Form [29] such that the x-axis plots throughput, the y-axis plots average latency, and points of the curves are values of average latency and throughput for increasing values of applied load. Latency is measured in terms of network cycles and includes message queue waiting time as well as network routing time. Throughput is measured as normalized delivered traffic in flits/node/cycle. The effects of recovering from message-dependent deadlocks on latency and throughput are factored into the performance results if they occur.

4.3.2 Performance Results

We compare the performance of the various techniques first by varying the number of virtual channels per physical link from 4 to 8 to 16 to 64. Figures 8, 9, and 10 plot performance assuming 4, 8, and 16 virtual channels, respectively. Since each physical link must have more than 4 virtual channels to implement SA when the chain length is greater than two, only results for DR and PR are given for simulations with 4 virtual channels for all patterns except PAT100. Likewise, for PAT100, DR is not valid, so no results are given. As there is no significant difference between the results for 16 and 64 virtual channels, the latter are not plotted.

Let us first consider the results for 4 virtual channels, shown in Figure 8. Up to the network load at which throughput is 20%, the performance gap between the schemes remains under 15% in terms of average message latency. Under these lightly loaded conditions, few network resources are sufficient to deliver messages without notable congestion. Beyond

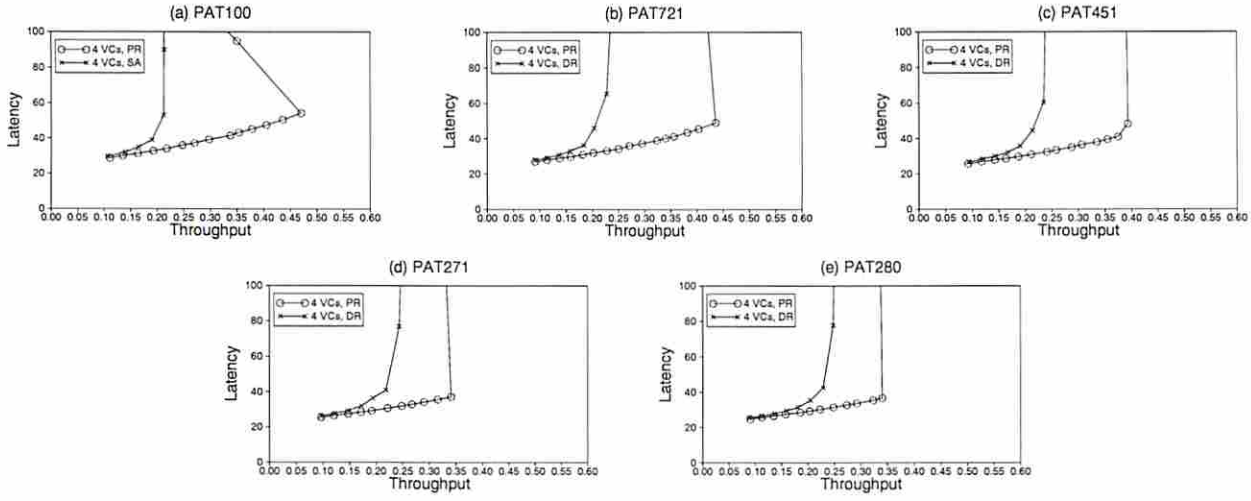

Figure 8: Network throughput and latency for the networks with message-dependent deadlock avoidance and recovery. (a) PAT100, (b) PAT721, (c) PAT451, (d) PAT271 and (e) PAT280 transaction patterns in 8×8 bidirectional torus with 4 virtual channels.

this load, however, DR and SA suffer from poor throughput and high average latency. This is due to inefficient utilization of network channel resources. In SA, network partitioning and restricted use of network resources needed to strictly avoid message-dependent deadlock lead to unbalanced use of network resources and, hence, increased congestion. With only four virtual channels, DR experiences the same influences on congestion as SA except that message-dependent deadlock is not strictly avoided for chain lengths greater than two. In contrast, PR yields up to 100% more throughput than DR for PAT721 and over 100% more throughput than SR for PAT100. As the average chain length increases, however, the difference in improvement reduces but is still substantial. The reduction in throughput is due to network endpoints experiencing message coupling, due to the message queues being shared by all message types in PR. This phenomenon is also observed in the networks with 8 and 16 virtual channels, and is explain in greater detail later.

Let us now consider the results for 8 channels, shown in Figure 9. For the same reasons as given above, the SA saturates at an early load due to only one of the eight virtual channels being available for routing of each message type. This unbalanced use of network resources is particularly acute when the message distribution is concentrated on only a few of the message types within a given message dependency chain, as with PAT721 and PAT421. However, when the chain length is only two, as with PAT100, three of the eight virtual channels (or five if the method in [21] is used) are available for routing of each message type. This provides more balanced use of network resources and sufficient routing freedom to make the difference between SA and PR negligible. A similar effect is

seen with DR in which the network is partitioned into only two logical network partitions. Therefore, in those patterns with chain lengths greater than two, most of the traffic is nearly equally distributed over all network resources as is the case with PR, making the difference between DR and PR practically negligible.

Finally, Figure 10 plots results for 16 virtual channels. Since the maximum chain length is four, the additional channels can be used to distribute traffic more evenly. That is, three (or nine [21]) of the sixteen virtual channels are available for routing of each message type for SA, and seven (or 13 [21]) are available for DR. All sixteen are available for PR. However, beyond a certain number of virtual channels, the difference in performance falls off dramatically since the use of network link bandwidth is already fairly evenly utilized [30].

However, another important and more significant effect is observed when traffic balancing is not an issue. As mentioned briefly in the discussion of results for 4 virtual channel, the sharing of message queues at network endpoints causes some degree of message coupling between heterogeneous message types. This occurs with DR for two message types and with PR for four message types. Both of these schemes have lower throughput than SA due to the fact that message coupling (and blocking) at network endpoints is a dominant factor that can limit performance. However, with both schemes, it is possible to separate message queues according to each message type (as is required by SA) and realize increased performance. This is shown in Figure 11 assuming PAT271. When each message type uses its own input and output message queues (denoted as Q4 in the figure), both the DR and PR schemes outperform SA. On the other hand, when message queues are shared between heterogeneous message types, inter-message coupling and blocking in the message queues are performance bottlenecks.

5 Related Work

Traditional solutions to message-dependent deadlock either provide network interfaces with sufficiently large queues or provide logically separate networks for each message type or group of message types. The IBM SP2 [13] which has a multistage interconnection network (MIN) communication backbone avoids message-dependent deadlock by guaranteeing sufficient queue space for each source-destination pair. Along with a configurable buffer space, end-to-end windowing flow-control is used such that messages are only sent when they are guaranteed to be sunk. The Cray T3D [17], Cray T3E [18], SGI Origin2000 [10], Intel Teraflops [19] and Stanford DASH [31] multiprocessor machines either strictly avoid message-dependent deadlocks by providing logically separate networks for each message type or deflectively recover from them by providing logical networks for

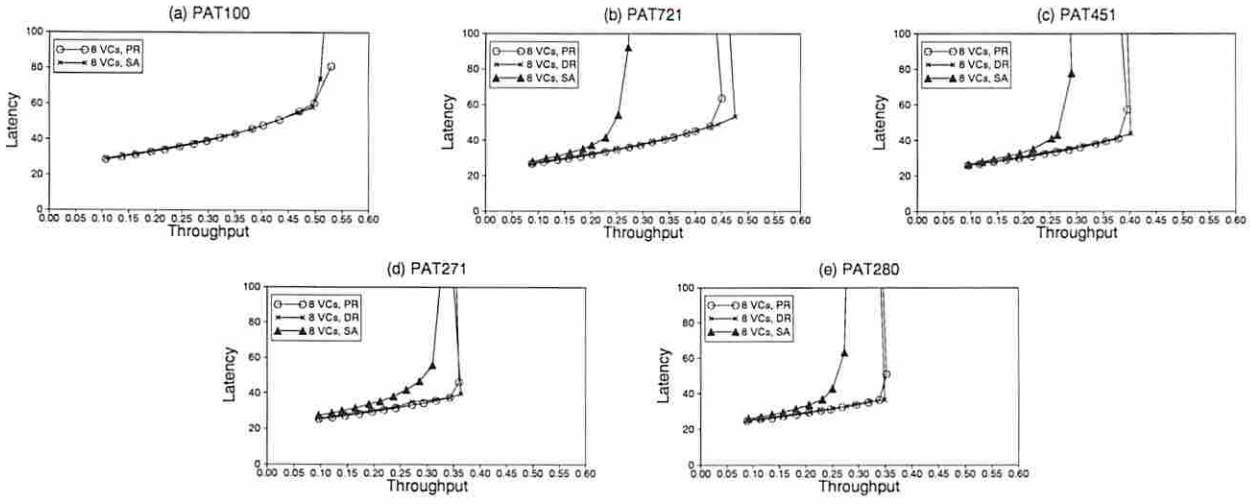

Figure 9: Network throughput and latency for the networks with message-dependent deadlock avoidance and recovery. (a) PAT100, (b) PAT721, (c) PAT451, (d) PAT271 and (e) PAT280 transaction patterns in 8×8 bidirectional torus with 8 virtual channels.

groups of message types and guaranteeing that some message types can always sink via deflection (or "backoff"). The idea of progressively recovering from potential deadlock situations in interconnection networks was first proposed in [25] and [32]. However, as proposed, the recovery algorithm is only valid for networks consisting of homogeneous messages. Thus, to the best of our knowledge, the technique presented in this paper is the first progressive message-dependent deadlock recovery scheme proposed. Finally, a recent study characterizing message-dependent deadlocks [7] using synthetic traffic loads provides results that are consistent with those discovered here using execution-driven application traces.

6 Conclusion

In this paper, message-dependent deadlocks are described and various approaches for handling them are evaluated. A new technique is also proposed which relaxes restrictions considerably, allowing the routing of packets and the handling of message-dependent deadlocks to be much more efficient, particularly when network resources are scarce. Results indicate that message-dependent deadlocks rarely occur for typical traffic loads and network parameters. Results also indicate that the proposed progressive recovery technique outperforms its avoidance-based and deflective recovery-based counterparts for a wide range message type distributions, i.e., cache coherence protocols and data transaction patterns. When network resources are abundant, it is profitable to separate message queue resources at

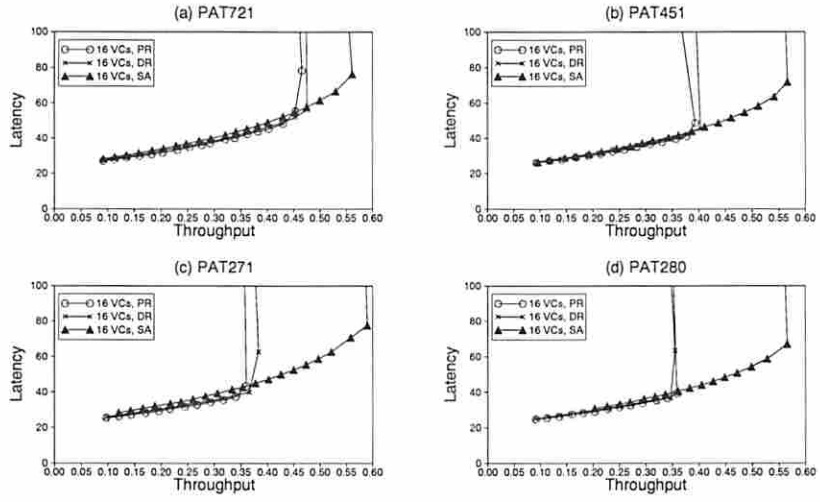

Figure 10: Network throughput and latency for the networks with message-dependent deadlock avoidance and recovery. (a) PAT721, (b) PAT451, (c) PAT271 and (d) PAT280 transaction patterns in 8×8 bidirectional torus with 16 virtual channels.

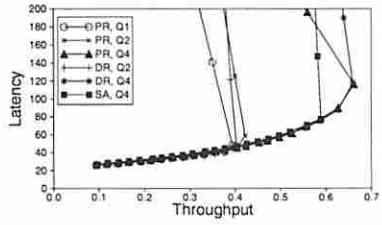

Figure 11: Network throughput and latency for the networks with message-dependent deadlock avoidance and recovery for various message buffer configurations in 8×8 bidirectional torus with 4 message types and 16 virtual channels.

network endpoints according to message type—not for deadlock avoidance purposes but, rather, for performance purposes. This reduces inter-message coupling and congestion at network endpoints but does not altogether prevent deadlock from occurring. Although the performance of avoidance-based techniques improves with increased network resources, the required partitioning of network resources (i.e., virtual channels) and limited routing freedom is overly restrictive. Considering the fact that many commercial systems implement sixteen or less virtual channels per physical link and use communication protocols consisting of message dependency chain lengths greater than two, the proposed *Extended Disha Sequential* progressive recovery technique presents an attractive alternative.

References

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

[2] L. Gravano, G. Pifarre, P. Berman, and J. Sanz. Adaptive Deadlock- and Livelock-Free Routing With all Minimal Paths in Torus Networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(12):1233–1251, December 1994.

[3] L. Schwiebert and D. N. Jayasimha. A Necessary and Sufficient Condition for Deadlock-free Wormhole Routing. *Journal of Parallel and Distributed Computing*, 32(1):103–117, January 1996.

[4] J. Kim, Z. Liu, and A. Chien. Compressionless Routing: A Framework for Adaptive and Fault-tolerant Routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):229–244, March 1997.

[5] Timothy Mark Pinkston. Flexible and Efficient Routing Based on Progressive Deadlock Recovery. *IEEE Transactions on Computers*, 48(7), July 1999.

[6] Sugath Warnakulasuriya and Timothy Mark Pinkston. A Formal Model of Message Blocking and Deadlock Resolution in Interconnection Networks. *To appear in IEEE Transactions on Parallel and Distributed Systems*, 1998.

[7] Yong Ho Song and Timothy Mark Pinkston. On Message-Dependent Deadlocks in Multiprocessor/Multicomputer Systems. *To appear in the Proceedings of the 7th International Conference on High Performance Computing*, December 2000.

[8] L. Widdoes Jr. and S. Correll. The S-1 Project: Developing High Performance Computers. In *Proc. COMPCON*, pages 282–291, Spring 1980.

[9] L. M. Censier and P. Feautrier. A New Solution to Coherence Problems in Multicache Systems. *IEEE Transactions on Computers*, C-27:1112–1118, December 1978.

[10] James Laudon and Daniel Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 241–251. IEEE Computer Society, June 1997.

[11] Alpha 21364 to Ease Memory Bottleneck. *Microprocessor Report*, 12, October 1998.

[12] J. Duato. A Necessary and Sufficient Condition for Deadlock-free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, October 1995.

[13] C.B. Stunkel et al. The SP2 high-performance switch. *IBM Systems Journal*, 34(2):185–204, 1995.

[14] A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B-H. Lim, K. Mackenzie, and D. Yeung. The MIT alewife machine: Architecture and performance. In *Proc. of the 22nd Annual Int'l Symp. on Computer Architecture (ISCA '95)*, pages 2–13, June 1995.

[15] W-D. Weber, S. Gold, P. Helland, T. Shimizu, T. Wicki, and W. Wilcke. The Mercury Interconnect Architecture: A Cost-effective Infrastructure for High-performance Servers. In *Proceedings of the 24th International Symposium on Computer Architecture*, pages 98–107. IEEE Computer Society, June 1997.

[16] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St. Pierre, David S. Wells, Monica C. Wong, Shaw-Wen Yang, and Robert Zak. The network architecture of the connection machine cm-5. In *Symposium on Parallel and Distributed Algorithms*, pages 272–285, 1992.

[17] Steve Scott and Greg Thorson. Optimized Routing in the Cray T3D. In *Proceedings of the Workshop on Parallel Computer Routing and Communication*, pages 281–294, May 1994.

[18] Steven L. Scott and Gregory M. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Proceedings of the Symposium on Hot Interconnects*, pages 147–156. IEEE Computer Society, August 1996.

[19] Joseph Carbonaro. Cavallino: The Teraflops Router and NIC. In *Proceedings of the Symposium on Hot Interconnects IV*, pages 157–160. IEEE Computer Society, August 1996.

[20] W. Dally and C. Seitz. Deadlock-free Message Routing in Multiprocessor Interconnection Net works. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.

[21] Jose F. Martinez, Josep Torrellas, and Jose Duato. Improving the Performance of Bristled CC-NUMA Systems Using Virtual Channels and Adaptivity. In *Proceedings of 13th International Conference on Supercomputing*, June 1999.

[22] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 148–159. IEEE Computer Society Press, 1990.

[23] David E. Culler, Jaswinder Pal Singh, and Anoop Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann Publishers, Inc., 1999.

[24] Sugath Warnakulasuriya and Timothy Mark Pinkston. Characterization of Deadlocks in k-ary n-cube Networks. *IEEE Transactions on Parallel and Distributed Systems*, to appear, 1999.

[25] Anjan K.V. and Timothy Mark Pinkston. An Efficient, Fully Adaptive Deadlock Recovery Scheme: *DISHA*. In *Proceedings of the 22nd International Symposium on Computer Architecture*, pages 201–210. IEEE Computer Society, June 1995.

[26] Sugath Warnakulasuriya and Timothy Mark Pinkston. A Formal Model of Message Blocking and Deadlock Resolution in Interconnection Networks. *IEEE Transactions on Parallel and Distributed Systems*, 11(3):212–229, March 2000.

[27] Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve. Rsim reference manual. version 1.0. Technical Report Techinical Report 9705, Department of Electrical and Computer Engineering, Rice University, July 1997.

[28] J.M. Martinez, P. Lopez, J. Duato, and T.M. Pinkston. Software-based Deadlock Recovery for True Fully Adaptive Routing in Wormhole Networks. In *Proceeding of the 1997 International Conference on Parallel Processing*, pages 182–189. IEEE Computer Society, August 1997.

[29] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach.* IEEE Computer Society Press, 1997.

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

[31] Daniel Lenoski et al. The stanford dash multiprocessor. *IEEE Computer*, 25(3):63–79, March 1992.

[32] Anjan K.V. and Timothy Mark Pinkston. *DISHA*: A Deadlock Recovery Scheme for Fully Adaptive Routing. In *Proceedings of the 9th International Parallel Processing Symposium*, pages 537–543. IEEE Computer Society, April 1995.

Appendix

Lemma 1 The *Extended Disha Sequential* technique allows the network to safely recover from all message-dependent deadlocks.

Proof Sketch The *Disha Sequential* [5] technique provides for a connected and deadlock-free recovery path between network endpoints. It is, therefore, free from routing-dependent deadlocks by design. It follows that only deadlocks involving resource dependencies at network endpoints (i.e., network interfaces) are possible. A mechanism capable of detecting potential message-dependent deadlocks (as discussed in Section 2.2) is assumed to exist in each network interface. The circulating token mechanism guarantees that there exists at least one message in each potential message-dependent deadlock that is rescued, and at most one rescued message at a time gains access to the set of recovery resources. During rescue, the token is used as the mechanism which grants access to recovery resources, and it is sent along with each message that is routed over recovery resources. The set of recovery resources themselves do not become involved in deadlock since messages using them during the rescue process are not allowed to occupy those resources indefinitely. There are four cases for all messages that have been granted access to the memory controller during the rescue process:

1. the message generates one or more subordinate messages, all of which can be delivered to the output queue at that node,
2. the message generates at most one subordinate message, which is terminating and cannot be delivered to the output queue at that node,
3. the message generates at most one subordinate message, which is non-terminating and cannot be delivered to the output queue at that node, or
4. the message generates more than one subordinate message, all of which cannot be delivered to the output queue at that node.

Let us consider each case separately. In Case 1, the subordinate message(s) are placed in the output message queue of the node after being generated. If this node is the node which originally captured the token or is the original token sender, the token is re-circulated and the message-dependent deadlock is resolved; otherwise, the token is sent back to the node which sent it, and the rescue process continues.

In Case 2, the subordinate terminating message is either delivered to the input queue at its destination (using DMB and DB recovery resources) or is sunk directly by the memory controller (via preemption) at its destination. In either case, the token that was sent along

with this message is sent back to the node which sent the subordinate message. If this node is the original token sender, the token is re-circulated and the message-dependent deadlock is resolved; otherwise, the token is sent back to the node which sent it, and the rescue process continues.

In Case 3, the subordinate non-terminating message is either delivered to the input queue at its destination or is processed by the memory controller (via preemption) at its destination. If processed by the memory controller at its destination, this message falls under one of the three cases mentioned above. If delivered, the token that was sent along with this message is sent back to the *sender* node which sent the subordinate message. If this node is the original token sender, the token is re-circulated and the message-dependent deadlock is resolved; otherwise, the token is sent back to the node which sent it, and the rescue process continues.

In Case 4, the first subordinate message that cannot be placed in the output queue of the node is either delivered to the input queue at its destination or is processed by the memory controller (via preemption) at its destination. If processed by the memory controller at its destination, this message falls under one of the three cases mentioned above. If delivered, the token that was sent along with this message is sent back to the *sender* node which sent the subordinate message. All other subordinate messages generated by the message being processed by the memory controller of that *sender* node undergo the same process as the first subordinate message, reusing the token. That is, they are either delivered to the input queue at their destination or are processed by the memory controller (via preemption) at their destination. This continues until the token for the last subordinate message has been received by the *sender* node. If this node is the original token sender, the token is re-circulated and the message-dependent deadlock is resolved; otherwise, the token is sent back to the node which sent it, and the rescue process continues.

When a rescued message tries to preempt the memory controller at the destination node, one of three situations exist. In the first case, the memory controller is idle, so preemption can take place immediately. In the second case, the memory controller is currently servicing a message from the input message queue, but is not rescuing it. In this case, the memory controller is guaranteed to complete the operation and deposit the subordinate message(s) in the output queue, making it available for preemption. In the third and final case, the memory controller is processing a message that is part of the message dependency chain of a message being rescued. In this case, local resources had to have been preallocated for sinking the incoming message, according to our assumption. Therefore, preemption is not needed to free-up recovery resources. Evidently, the token is reused for subordinate messages along a message dependency chain until one of them is

delivered or sinks. Nevertheless, the rescue process is guaranteed to terminate since each message dependency chain is acyclic and has a terminating message type. Therefore, in all cases, message-dependent deadlock is resolved as at least one message (and all subordinate messages generated by it) no longer wait only for resources involved in the deadlock [26]. □