An Efficient, Fully Adaptive Deadlock Recovery Scheme: *DISHA* *

Anjan K. V. Timothy Mark Pinkston

Electrical Engineering - Systems Department, University of Southern California
3740 McClintock Avenue, EEB-208, Los Angeles, CA 90089 - 2562
{*anjan@truth, tpink@charity*}*.usc.edu; http://www.usc.edu/dept/ceng/faculty.html/pinkston/home.html*

Abstract:

This paper presents a simple, efficient and cost effective routing strategy that considers deadlock recovery as opposed to prevention. Performance is optimized in the absence of deadlocks by allowing maximum flexibility in routing. Disha supports true fully adaptive routing where all virtual channels at each node are available to packets without regard for deadlocks. Deadlock cycles, upon forming, are efficiently broken by progressively routing one of the blocked packets through a deadlock-free lane. This lane is implemented using a central "floating" deadlock buffer resource in routers which is accessible to all neighboring routers along the path. Simulations show that the Disha scheme results in superior performance and is extremely simple, ensuring quick recovery from deadlocks and enabling the design of fast routers.

1.0 Introduction:

The interconnection network is the backbone for communication in a multiprocessor/multi-computer environment. System performance is determined not only by the effective utilization of multiple processor nodes, but also, to a large extent, by efficient communication amongst the nodes. For this reason, many schemes have been proposed which incorporate wormhole switching [21] and adaptive routing [16] to increase communication efficiency.

In such schemes, the critical issue of deadlocks must be addressed. Deadlocks in routing occur as a result of cyclic waits for network resources by packets. Avoidance has been the traditional solution to deadlocks. Schemes based on deadlock avoidance generally suffer from losses in adaptivity and/or increased hardware complexity which negatively impact performance.

Consider first how adaptivity can suffer. The Turn Model [20], which does not require virtual channels, prevents deadlock by prohibiting those turns that could result in the formation of cycles in the channel dependency graph [9]. As a result, the adaptivity of routing algorithms based on the Turn Model is limited. The West-first algorithm, for example, demands that all packets be routed non-adaptively west first before they can be routed adaptively in other directions. Hence, if packets encounter congestion along the west direction, they must block, resulting in poor performance. Simulation studies show that such schemes produce unbalanced traffic conditions and often are outperformed even by non-adaptive algorithms [8]. Such partially adaptive schemes also cannot be fault tolerant.

Another partially adaptive deadlock avoidance scheme is Planar-Adaptive Routing[4], which requires three virtual channels for n-dimensional mesh networks. It ensures deadlock freedom by restricting adaptivity to at most two dimensions at a time and structuring the passage of packets from one adaptive plane to another. Again, adaptivity suffers because some idle channels along minimal paths in the n-2 other dimensions are automatically excluded by the routing algorithm since they lie outside of the adaptive plane.

To increase adaptivity, Linder and Harden [19] proposed a fully adaptive deadlock avoidance routing algorithm that requires an exponential growth (based on dimensionality) in the number of virtual channels to eliminate cycles in the channel dependency graph. Even with this large number of virtual channels, only a small subset of those channels are available to each packet as they are grouped into ordered levels or classes. Dally and Aoki's Static Routing Algorithm [11] is similar in that it, too, groups virtual channels into ordered classes based on packet dimension reversals (higher to lower dimension traversals). This restricted use of virtual channels to ensure proper ordering for deadlock avoidance results in inefficient use of resources and, hence, falls well short of *true* full adaptivity.

Consider next how avoidance schemes can increase hardware complexity. Increasing adaptivity (for deadlock avoidance routing) by adding additional virtual channels generally results in increased router switch and Virtual Channel Controller (VCC) complexity. A study by Chien [5] shows that the increased complexity adversely affects machine performance by increasing router clock cycle time. In fact, deterministic schemes could outperform certain adaptive schemes which require many virtual channels. This

* *The research described in this paper was supported in part by an NSF Research Initiation Award, grant ECS-9411587, and by a grant from the Zumberg Fund and the Powell Fund.*

ISCA '95, Santa Margherita Ligure Italy
© 1995 ACM 0-89791-698-0/95/0006...$3.50

is a motivation behind recent adaptive avoidance schemes which minimize the required number of virtual channels.

Dally and Aoki's Dynamic Routing Algorithm [11] is a more efficient fully adaptive scheme that relaxes the restrictions which force virtual channel ordering. This reduces the number required. Deadlocks are avoided by not allowing cycles in the packet wait-for graph. Each packet keeps track of the number of dimension reversals it suffers. A packet is routed adaptively on any of the channels in the adaptive class until blocked with all suitable output channels being used by packets with equal or lower values of dimension reversals. The packet is then forced onto the deterministic class of channels which is deadlock-free and must remain there until routed to its destination. Here, a packet's dimension reversals (relative to other packets at a given router) ultimately places an upper bound on adaptivity.

Duato's fully adaptive deadlock avoidance algorithm [12, 13] provides additional flexibility in routing. Similar to the previous scheme, virtual channels are divided into adaptive and deterministic classes. Packets can be routed with full adaptivity using any available adaptive channel and can be routed with partial adaptivity using the deterministic channels which form escape paths. Only on blockage must a packet use an escape channel at the congested router; at subsequent routers, it is free to go back onto the adaptive channels (if free). The existence of a connected escape path which has no cycles in its extended channel dependency graph is sufficient for deadlock avoidance [13].

Both schemes require few additional virtual channels above that required for deterministic routing, but inefficiencies still exist. For example, if one uses Chien's model [5] to determine that the optimum number of virtual channels for the expected load conditions is three for a toroidal k-ary n-cube network, these schemes can use only one virtual channel for fully adaptive routing. Chien [5] shows that every additional virtual channel requires a 30 - 50% increase in load to justify its decrease in router speed. This being the case, it seems unjustified to allow fully adaptive routing on only a fraction of the total number of virtual channels. Likewise, if one determines that the optimum number of virtual channels for flow control purposes is two for a toroidal network, fully adaptive routing is not permitted by either scheme.

Deadlock recovery is a viable alternative to avoidance. Simulation studies show that deadlocks generally are rare [3, 17, 23]. Because deadlocks are infrequent events, recovery seems to make more sense than prevention. In [23], an abort-and-retry mechanism is proposed as a technique for preventing and/or recovering from deadlock. Compressionless Routing [17] is another fully adaptive deadlock recovery scheme based on abort-and-retry that simply kills deadlocked packets. Although these schemes are simple and do not require virtual channels, they do have some implementation disadvantages. For instance, in Compressionless Routing, padding decreases effective channel utilization; when the packet sizes are small compared to the network diameter or the buffers at routers are deep, the overhead due to padding is large. Packets have to be stored to allow retransmission if necessary, and killed packets suffer increased latencies. Moreover, injector and receiver interfaces, additional counters, and status lines increase hardware complexity.

We believe routing should be fully adaptive in its truest sense and that the router design should be extremely simple. The goal here is to make the common case fast and *Disha* is a solution to this. It is not advantageous to limit adaptivity at the expense of an infrequent event. Doing so negatively impacts performance and the fault-tolerance capabilities of the system. Likewise, it is not advantageous to devote virtual channels specifically to prevent deadlocks; virtual channels should be used only as a means of improving flow control [10] such as with adaptive routing.

Disha is a deadlock recovery strategy that provides a framework for supporting deadlock-free fully adaptive wormhole routing. It effectively decouples routing flexibility from flow control resources by dedicating nominal hardware towards efficient deadlock recovery. Fully adaptive routing is permitted with no virtual channels, irrespective of the network topology. Virtual channels limit only the flow capabilities of the network, not routing adaptivity. *Disha* therefore offers the following advantages: 1) *true* fully adaptive wormhole routing, 2) applicability to any interconnection network topology, 3) no virtual channels required for prevention of deadlocks, 4) simple design of fast routers, and 5) no abort-and-retry of deadlocked packets.

The remainder of the paper is organized as follows. Section 2 describes the proposed scheme. Section 3 discusses implementation issues. Section 4 presents simulation and performance results. Finally, Section 5 gives our conclusions and future work. A proof that the *Disha* scheme safely recovers from deadlock events for arbitrary interconnection networks is given in the Appendix.

2.0 *DISHA* Recovery Scheme:

Disha[†] aims at optimizing routing performance in the absence of deadlocks. This is achieved by allowing routing to be fully adaptive and efficiently dealing with the rare cases when deadlock does occur. This scheme requires nominal hardware to be devoted to deadlock recovery.

Disha operates under the most relaxed condition of permitting unrestricted routing on all existing virtual channels or "edge" buffers (hereafter, edge buffers and virtual channels are used interchangeably). This results in *true* fully adaptive routing; even non-minimal routing is permitted. Recovery from deadlocks is through a single additional flit buffer at each node, which is the minimum required for a non-abort-and-retry recovery scheme. This "Deadlock Buffer" is a special input buffer central to each router and is to be used only when deadlock is presumed. A Deadlock Buffer can be accessed from all neighboring nodes

[†] *Disha* means "*direction*" in *Hindi.*

and, consequently, constitutes a shared resource. Deadlock Buffers form a deadlock-free lane during recovery which can be visualized as a "floating" virtual channel. On deadlock, one of the packets in the cycle is switched to the deadlock-free lane and routed minimally along this path until it reaches its destination where it will be consumed (sunk) to break a dependency cycle. Routers designed in this fashion are not only simple but also potentially faster.

To illustrate how a potential deadlock situation can be recovered from, we provide the following example. In Figure 1, the special buffer path can be viewed as an alternate network interleaved with the original. Using this alternate path, a packet, say P_1, which was waiting on P_2, upon detection that it is deadlocked, can reach its destination where it will be sunk. This breaks the deadlock cycle and other packets, P_4, P_3 and P_2, are able to proceed. Recovery is achieved. It should be noted that allowing any one of the packets to proceed breaks the entire deadlock cycle. Also, there is no ordering as to which packet should be placed on the deadlock-free lane.

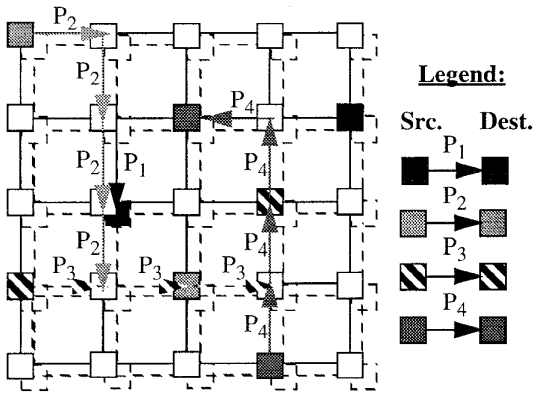

Fig.1 shows deadlock recovery with *Disha*.

Disha is reminiscent of Duato's algorithm [12] of having two virtual networks -- one susceptible to deadlocks (possibly adaptive) and the other deadlock-free (possibly deterministic). However, there are significant differences. Escape paths in Duato's scheme use edge buffers. Depending on the topology (mesh or torus) more than one virtual channel might be required. However, the escape channel in *Disha* is a single Deadlock Buffer central to the router. This buffer is shared between neighboring nodes and, unlike edge buffers, is not dedicated to any path.

In *Disha*, the cost complexity of the VCC remains unchanged as the Deadlock Buffer is a central input resource as opposed to an edge resource. The crossbar complexity is increased by just one at the input and is unchanged at the output. Note that additional virtual channels needed for escape paths in preventive schemes can increase the crossbar complexity by the node degree and similarly increase the complexity of the VCC.

Recovery from deadlocks can be done either sequentially or concurrently. Under sequential recovery, the routers need to obtain exclusive access to the Deadlock Buffer path before they are allowed to place a packet on it. Mutual exclusion could be implemented with a Token as is done in [1]. Once exclusive access is obtained (i.e., by capturing the Token), a router can place a deadlocked packet on the output channel with the corresponding status line asserted to indicate that this packet should be placed on the Deadlock Buffer at the next router. The packet continues to use Deadlock Buffers at subsequent nodes until it is delivered at its destination. The proof that this packet is guaranteed to reach its destination and, thereby, break a deadlock cycle is presented in the Appendix. Concurrent recovery does not require routers to obtain exclusive access to the Deadlock Buffer path before they are allowed to route deadlocked packets. By appropriately structuring routing on the Deadlock Buffers, concurrent recovery is possible [2]. This allows parallel recovery from single deadlock cycles and simultaneous parallel recovery from multiple cycles. This, too, requires just a single central Deadlock Buffer and completely eliminates the Token which is a possible single point of failure. Concurrent recovery is not discussed in detail as it is beyond the scope of this paper.

Disha handles livelocks in a similar fashion to what is found in most other schemes, i.e., by placing an upper bound on the number of misroutes allowed.

3.0 Implementing *Disha*:

Disha requires minimal additional hardware to be devoted to recovery. This section discusses these issues.

3.1 Deadlock Detection:

Deadlocks can be detected using a time-dependent selection function similar to those suggested in [15, 17]. The router waits for the arrival of a packet. On receiving one, it resets $T_{elapsed}$ which keeps track of the number of clock cycles this router is unable to send out the header. If unable to send out the header, $T_{elapsed}$ is incremented. The router continues to increment $T_{elapsed}$ on every cycle until it is successful in sending out the *header* or until the time-out (T_{out}) interval for that packet has been reached. When $T_{elapsed} > T_{out}$, the router changes its state to "deadlocked".

The selection of a proper time-out interval is important to obtaining optimum performance. Deadlock feeds upon itself in that if cycles are not broken quickly, more and more routers can get engulfed in cycles, and the entire network could come to a standstill. Large T_{out}'s can limit peak obtainable performance. On the other hand, small T_{out}'s trigger *false* deadlock detections. The choice of T_{out} thus becomes critical. Section 4 briefly examines this issue and shows how performance varies with different time-out's.

3.2 Hardware Requirements:

The concept of a central Deadlock Buffer is similar to the central queue proposed in the Chaos router [18] for virtual cut-through switching. The queue in the Chaos router

receives messages through a bus from all the input frames. The Deadlock Buffer could be implemented similarly as described in [3]. Nominal additional logic is required to implement the deadlock detection mechanism described previously [3].

Status information is required to select the appropriate input buffer: either one of the edge buffers or the Deadlock Buffer. For no additional cost, this information could be encoded on existing status lines required to distinguish between virtual channels as long as the number of virtual channels is not an exact power of two. For sequential recovery implemented with a Token, a hardwired Token path and Token propagation/capture circuit are additional. The asynchronous Token implementation proposed in [1] drastically reduces propagation time and hides the Token capture latency. Finally, a reconfiguration buffer might be required with certain crossbar allocation policies as explained below.

3.3 Deadlock Recovery with *Disha*:

Routers can be implemented with internal flow control so that the edge output buffers are guaranteed to clear. In doing so, the VCC cooperatively controls with the internal flow controller intranode data flow. It accepts only those flits that can be delivered based on internode flow control. Such an implementation could in fact reduce the critical path through the router [6].

Router switch (crossbar) connections can be allocated to messages either on a flit-by-flit basis or a packet-by-packet basis. Flit-by-flit allocation requires the crossbar to be reconfigured on every flit transmission. This applies to router implementations which multiplex input and/or output ports at the crossbar switch. Packet-by-packet allocation results in the crossbar being reconfigured anew for every packet transmission as opposed to every flit. This policy applies to router implementations that do not multiplex input nor output ports at the crossbar switch. Multiplexing of inputs could cause a loss in bandwidth due to internal data blocking and is normally not done. Data through delay could be reduced by not multiplexing output ports [6].

The operation of *Disha* for a flit-by-flit crossbar allocation policy is explained below. For simplicity we illustrate operation with only one virtual channel per physical channel. Operation is similar with multiple virtual channels. Figure 2a shows a possible initial state of routers R_a, R_b and R_c. Here, packet P_1 is blocked by P_2 and is unable to proceed. Router R_a is unable to send out the *header* of P_1 for T_{out} clock cycles and assumes packet P_1 to be deadlocked. Router R_a now sends out the *header* of P_1 with the corresponding *Status* line asserted. Router R_b places the incoming *header* of P_1 on the Deadlock Buffer, bypassing the normal input buffer currently occupied by packet P_2. Packet P_1 now follows the Deadlock Buffer path through subsequent routers to reach its destination. These actions are indicated in Figure 2b.

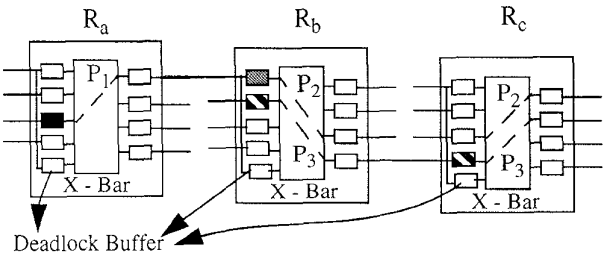

Fig. 2a shows a possible deadlock scenario where P_1 would otherwise wait on P_2 indefinitely.

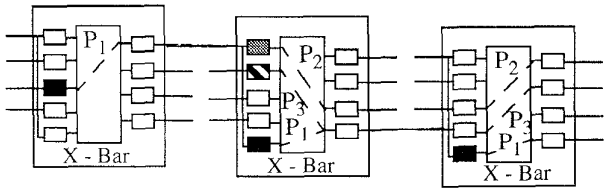

Fig. 2b shows how deadlocks are resolved. P_1 is routed around P_2 on the Deadlock Buffer to break the cycle.

The operation for a packet-by-packet router switch allocation policy is explained below. In the absence of deadlocks, the router crossbar is configured as specified by the decision and arbitration logic. However, in a deadlock situation, it could so happen that a deadlocked packet on the Deadlock Buffer requires the same output connection as is currently being used by some other packet from an edge input buffer. Because the crossbar is not reconfigured after every flit transmission, the crossbar must be reconfigured to connect the Deadlock Buffer to the output terminal once the required output buffer has cleared. The decision logic thus needs to remember the state of the crossbar before it was reconfigured so that it can reconnect the suspended input buffer back to the preempted output buffer once deadlock has cleared.

This is illustrated using the same example as in Figure 2. At router R_b, packet P_1 on the Deadlock Buffer requires the same output channel that is allocated to packet P_3 in one of the edge input buffers. In such a case, router R_b reconfigures the crossbar, connecting the Deadlock Buffer to the buffer associated with this output channel after first storing the crossbar state in a reconfiguration buffer for later reconnection. This buffer does not need to store the entire crossbar state, but only that of the input (at most one) that was disconnected. Because internal flow control allows the output buffer at router R_b occupied by P_3 to clear, packet P_1 can proceed. Data routed on the Deadlock Buffer is guaranteed to reach its destination. Once packet P_1 has cleared, the crossbar is returned to its initial configuration using the information stored in the reconfiguration buffer. Crossbar reconfiguration is essential because packet P_3 may be blocked further upstream (and never clear), leading to a deadlocked configuration where the packet in the Deadlock Buffer (at this router) cannot advance even though the output buffer is clear.

The flits of packet P_3 in router R_c could have moved forward as shown in Figure 2b. Although this might lead to additional discontinuities in flit reception beyond that suffered from the multiplexing of virtual channels onto the physical channel, all flits of packet P_3 are bound to reach their destination safely. The commandeering of output edge buffers for deadlock recovery is occasional given that the number of deadlock occurrences are infrequent. Simulations in Section 4 show that less than 2% of the total packets injected are susceptible to deadlock. Therefore, it is expected that only a small percentage of the packets suffer additional discontinuities due to the preemption of output buffers by Deadlock Buffers.

3.4 Cost Model:

Here, we compare the additional cost of implementing *Disha* with the *-Channels Router [7], which prevents deadlocks based on Duato's theory [12]. In drawing this comparison, we use typical router parameters and the cost model given in [5]. The network is assumed to be a 2D mesh with three virtual channels per physical channel. The routing latency for path set-up is greater than that for flow control (data through). To maximize performance, the router clock is assumed to run at the data through cycle time. Path set-up for every packet is required only once at each router and is implemented in multiple clock cycles [6]. Any additional latency that may arise in implementing path set-up in *Disha* can therefore be hidden.

The data through cycle time based on the model in [5] is given as $T_{data\ through} = T_{fc} + T_{cb} + T_{vcc}$, where T_{fc} is the flow controller delay, T_{cb} is the delay through the crossbar and T_{vcc} is the virtual channel controller delay. For a 0.8 micron CMOS process, these module delays are given by $T_{fc} = 2.2$ ns, $T_{cb} = 0.4 + 0.6\log P$ ns, and $T_{vcc} = 1.24 + 0.6\log V$ ns, where P is the number of inputs to the crossbar and V is the number of virtual channels that the VCC must multiplex onto a physical channel. Substituting typical router parameters into this model, the data through delay for the *-Channels router is $T_{*\text{-Channels}} = 7$ ns.

The central Deadlock Buffer in *Disha* increases the size of the crossbar by one. The size of the VCC remains unchanged. The data through delay for *Disha* can be computed as $T_{disha} = 7.1$ ns. The negligible increase in data through latency is overridden by the advantages that *Disha* provides in routing flexibility as shown by simulations in the next section. This is because *Disha* provides fully adaptive routing on all buffers. The *-Channels Router, on the other hand, can provide fully adaptive routing on only one of the virtual channels for a torus and on only two virtual channels if the topology is mesh-type.

4.0 Performance of *Disha*:

This section of the paper evaluates the performance of *Disha* with sequential recovery using a modified version of FLITSIM 2.0†. Deadlocked packets are assumed to gain exclusive access to the Deadlock Buffers by seizing a Token.

†FLITSIM was developed by Patrick Gaughan (currently at the University of Alabama) et al., at Georgia Institute of Technology.

4.1 Simulation Model:

All simulations are run on a 16 by 16 two dimensional torus with four virtual channels per physical channel and full-duplex links. Messages are 32 flits long. A buffer depth of two is selected (shallow buffers keep the routers simple and reduce clock cycle time). All algorithms simulated use one injection and reception channel per node. Equal clock cycle is assumed for all schemes being compared. Load-Rate is a fraction of full load, defined as the load at which all channels in the network are used simultaneously (maximum network capacity).

4.2 Deadlock Characterization:

This work is premised on the notion that deadlocks are rare. Figure 3a verifies this for two widely varying time-out thresholds, 4 and 64. The figure shows the number of packets that seize the Token normalized with respect to the number of packets that are delivered by the network. These simulations are done for uniform traffic and a maximum misroute of three. Deadlocks are even less frequent with minimal routing. The number of deadlocks are closely related to the number of packets that seize the Token. Based on these results, it is safe to infer that deadlocks are extremely rare up to the point at which the network saturates as can be seen from Figure 3b. This confirms earlier measurements by Kim, et al. [17].

Figure 3b conveys the additional improvement in network performance that a properly selected time-out could provide. Small time-outs trigger false deadlock detection while large ones unduly delay detection of true deadlocks. Time-outs of 8 and 16 are found to be appropriate, although simulations show that the optimum threshold value depends on the traffic pattern, the message length and topology. To adapt to different network conditions and topologies, T_{out} could be programmable to vary dynamically. This is an area of future research. The default time-out assumed for the remainder of our simulations is 8 clock cycles, although performance should only improve if the time-out interval were fine-tuned.

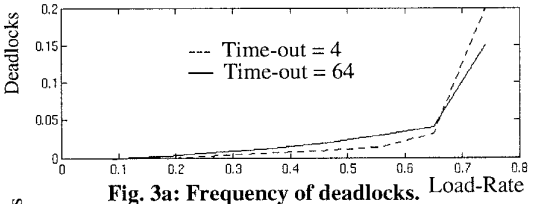

Fig. 3a: Frequency of deadlocks.

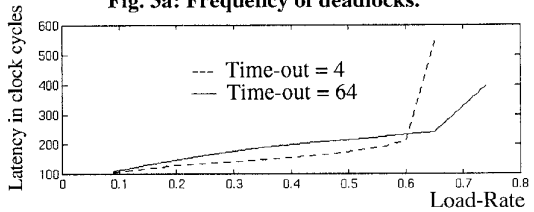

Fig. 3b: Selection of time-out.

4.3 Simulation Results:

A few words regarding the routing algorithm assumed in our simulations are in order. The selection function chooses a free output channel from the set supplied by the routing function. *Random* selects a free channel from the set at random. *Minimum congestion* chooses the channel in the direction in which most virtual channels are free and generally provides a substantial improvement. *Dally* (Dally and Aoki's Dynamic Algorithm) is the only one simulated with a minimum congestion selection function. Dimension ordered routing (*DOR*) is non-adaptive. For *Turn* (Turn Model), the Negative-First Algorithm is used which supposedly gives the best results among those derived using this model [20]. *Duato* is based on the conditions derived in [12]. Even with the relaxed set of conditions suggested in [13], we do not expect the resulting performance to reach *Disha*. *Disha* is simulated for two different misroute values: M=0 and M=3. Any virtual channel along the minimal path can be used for M=0 whereas any virtual channel along any path can be used for M=3, as long as the misroute count is less than four.

4.3.1 Uniform Traffic:

Under uniform traffic, each node sends packets to all other nodes with equal probability. Figure 4 compares *Disha* with preventive schemes under uniform traffic and four virtual channels. *Disha* considerably outperforms the other schemes. With no misrouting, latency increases linearly with load. It will be interesting to evaluate performance with multiple injection channels to see when the network becomes a bottleneck. With M=3, *Disha* saturates at 0.65, and *Duato* is a distant second at 0.35. The fact that *DOR* outperforms the partially adaptive *Turn* and the fully adaptive *Dally* is not surprising as it preserves the traffic's uniformity. The peak throughput for *Disha* (M=0) is 35% greater than that for *Duato*. Moreover, the network can sustain this peak throughput. For other schemes, messages block faster than they are drained and can remain blocked for long periods of time. Throughput drops off accordingly.

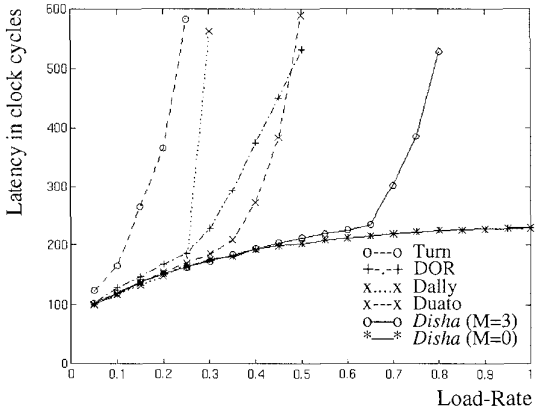

Fig. 4: Comparison for Uniform traffic.

4.3.2 Non-Uniform Traffic:

We now compare *Disha* with deadlock avoidance schemes for various other (regular) traffic patterns. The first non-uniform traffic pattern is *bit-reversal*. Here, a node with coordinates a_{n-1}, a_{n-2},..., a_1, a_0 sends messages to the node with co-ordinates a_0, a_1,..., a_{n-2}, a_{n-1}. With this traffic, nodes along a given row send messages to nodes along a given column. *Disha*, as expected, significantly outperforms all other algorithms. *Disha* with M=0 saturates at around 0.7 and with M=3 at around 0.45. The peak throughput for *Disha* is measured to be 50% greater than *Duato* and can be sustained.

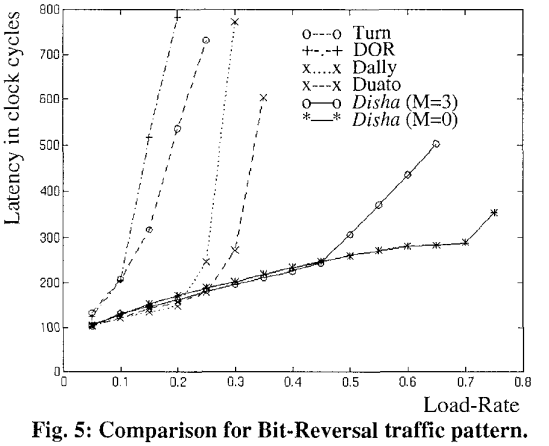

Fig. 5: Comparison for Bit-Reversal traffic pattern.

For the *matrix transpose* traffic pattern, a node with coordinates (x,y) sends to node (y,x). The 2D graph can be visualized as a number of concentric squares on which the source and destination pairs lie. *Disha* with no misrouting offers the best results, saturating at around 0.7 which is more than twice that of *Duato*. Here, too, the peak throughput for *Disha* is 50% greater than *Duato*, but this peak value is not sustained.

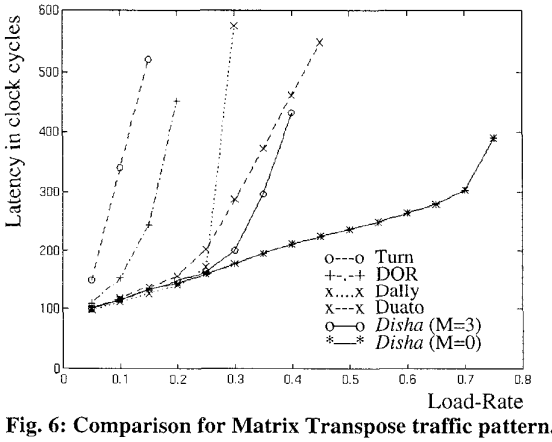

Fig. 6: Comparison for Matrix Transpose traffic pattern.

Finally, a comparison is made for *hot spot* traffic conditions. In these simulations, we assumed that up to 5% of the network traffic is hot spot in nature, and the number of

locations at which these patterns exist is one. This implies that 5% of all network traffic is trying to reach one destination that is randomly selected. Hot spot traffic causes early saturation for all schemes.The performance of *Disha* with M=3 is only slightly better than that of *Duato*. With M=0, *Disha* is outperformed by all other schemes. The result that misrouting is beneficial for hot spots is interesting. With hot spots and no misrouting, it is observed that the deadlock count increases exponentially. Misrouting therefore aids in routing around deadlock congestion for our recovery scheme.

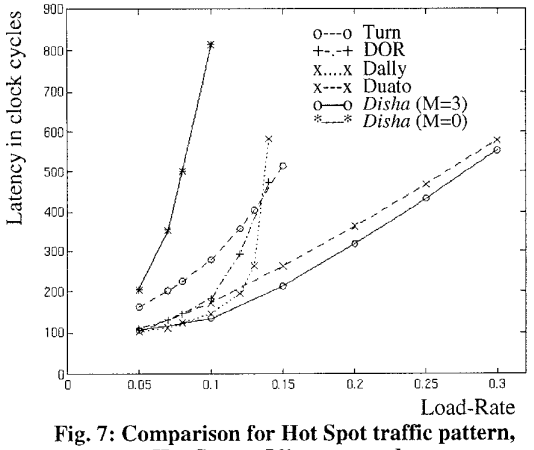

Fig. 7: Comparison for Hot Spot traffic pattern, Hot Spot = 5% to one node.

Simulations results for other non-uniform traffic patterns are found in [3]. Similar favorable results are observed for *Disha*.

4.3.3 Discussion:

Disha provides excellent performance. On average, it doubles the saturation load and improves throughput by 50% over the best deadlock avoidance scheme. Even after saturation, latency increases at a much slower rate when compared to the other schemes. The performance increase is largely due to the fact that in *Disha*, more virtual channels are available to packets at each node than that for the other schemes. There is no classification of virtual channels nor is there any ordering among them. The Deadlock Buffer is the escape path and is not a bottleneck due to deadlocks being rare. These results are obtained without any fine-tuning of *Disha* to improve results. The frequency of deadlocks can be reduced by increasing the number of reception channels at nodes to quickly drain packets. Injection limitation schemes such as throttling [11, 14] that maintain load below the saturation point could be used to reduce deadlocks even further.

Simulation results demonstrate that with misrouting, the performance of *Disha* deteriorates except when there are hot spots in the network. Performance degradation with misrouting could be attributed to increased network resources being held by blocked packets in wormhole switching. This increases the probability that packets deadlock and takes up valuable network bandwidth. Because recovery from deadlocks is sequential in our simulations, significant drop in performance results. We therefore conclude that misrouting is not beneficial and should be restricted to bypassing faults or hot spot congestion. Intelligent routers could adjust routing flexibility (degree of adaptivity, misrouting, etc.) based on local conditions or perhaps user-defined traffic hints. This remains as future work.

The performance of *Dally* and *Duato* are very similar. *Duato* eliminates cycles from the extended channel dependency graph whereas *Dally* eliminates cycles in the packet wait-for graph. *Duato* allows packets on escape channels to come back onto the adaptive channels whereas *Dally* does not, giving *Duato* more flexibility in how escape paths are used. On the other hand, in *Dally* blocked packets resort to the heavily congested escape channels only when absolutely necessary, giving it more flexibility in when escape paths are used. Also, *Dally* was simulated with minimum congestion. Perhaps these factors play a role in the differences and/or similarities of their performance results.

The performance of both *Duato* and *Disha* can be substantially enhanced by using a *minimum-congestion* selection function as was done in *Dally*.

5.0 Conclusions and Future Work:

Disha provides the means for safely incorporating *true* fully adaptive wormhole routing to support efficient and fault-tolerant communication in a parallel processing environment. The scheme is universal in that it can be applied to arbitrary network topologies. It employs deadlock recovery as opposed to prevention with the objective of making the common case faster. Consequently, it does not *require* virtual channels for deadlock-free fully adaptive routing. If, on the other hand, virtual channels are implemented, they serve the dual role of increasing flow-control as well as routing adaptivity. Hardware devoted to recovery is minimal, enabling the routers to be fast. This novel idea proves to be very effective as confirmed by exhaustive simulation.

Simulations show that the number of potential deadlock situations is very small up to the point at which the network saturates. Simulations also show that our deadlock recovery scheme, which uses all virtual channels for fully adaptive routing, performs significantly better than deadlock avoidance schemes, which must devote some portion of the virtual channels for deadlock prevention. What's more, the recovery scheme is not based on abort-and-retry which can increase latency. Routing complexity is minimal, resulting in a fast router clock. Furthermore, *Disha* performs well under bursty traffic and provides good fault-tolerance capability.

There are many areas of future research resulting from this work. A more complete characterization of deadlocks is necessary to extract maximum performance from a recovery scheme, such as *Disha*. As has been shown in this paper, the

performance of *Disha* depends on the selection of a suitable time-out interval. The optimum value of time-out will not be the same for different message lengths, buffer depths, traffic patterns, topology, etc. Such a study will help in fine-tuning the system to obtain peak performance. Reducing the probability of deadlocks is another related area of interest. It would also be interesting to see by how much performance is enhanced with concurrent recovery [2], which allows recovery from a number of simultaneous deadlock occurrences. Dynamically varying the degree of adaptivity, time-out and misroute count depending on the source and destination addresses and local traffic conditions could also be an interesting topic. Finally, more work can be done to increase fault-tolerance and decrease external channel delay, i.e., with optical interconnects [22].

Acknowledgments:

The authors are extremely grateful to Binh Vien Dao at Georgia Institute of Technology for helping us set up Flitsim 2.0, which was used in all the simulations presented in this paper. Enlightening discussions with Jose Duato have proved invaluable. Helpful suggestions from Sudhakar Yalamanchili also were very useful. We thank these individuals and others who have reviewed this work.

Appendix:

Theorem for Deadlock Recovery:

The *Disha* technique can be applied to any interconnection network. The definitions given below are abbreviated versions of those found in [12] and are included here only for completeness of the proof of our theorem. Strictly speaking, the theory developed in [12] cannot be directly applied to *Disha* because *Disha* uses both edge and central buffers. Additionally, the routing function is different for edge and central buffers. Thus, the routing function must consider the buffer where the packet is stored at the current node. A new formal theory which is a modification of [12, 13] is proposed in [2].

Definition 1:

An *interconnection network I* is a strongly connected directed multigraph, $I = G(N,C)$. The vertices N represent the set of processing nodes and the arcs C represent the set of communication channels connecting the nodes. The source and destination nodes of a channel c_i are s_i and d_i, respectively.

Definition 2:

An *adaptive routing function* $R : N \times N \rightarrow \prod(C)$, where $\prod(C)$ is the power set of C, supplies a set of alternative output channels from the current node n_c to the destination node n_d, $R(n_c, n_d) = \{c_1, c_2, ..., c_p\}$ where $c_1, c_2, ..., c_p \in C$. In particular, $R(n, n) = \emptyset$, $\forall\, n \in N$.

Definition 3:

The routing function R for a given interconnection network I is *connected* iff $\forall\, x, y \in N, x \neq y$, it is possible to establish a *path* $P(x, y) \subset \prod(C)$ between x and y using the channels supplied by R.

Definition 4:

A *routing subfunction* R_1 for a given routing function R is one that supplies a subset of the channels supplied by R. Thus R_1 restricts the routing options supplied by R. The set of channels supplied by R_1 is C_1. The routing subfunction can be expressed as $R_1(n_c, n_d) = R(n_c, n_d) \cap C_1\ \forall\, n_c, n_d \in N$. Needless to say, $C_1 \subseteq C$.

Definition 5:

Given an interconnection network I, a routing function R and a pair of channels $c_i, c_j \in C$, there is a *direct dependency* from c_i to c_j iff c_j can be used immediately after c_i by messages destined to some node n.

Definition 6:

Given an interconnection network I, a routing function R a channel subset $C_1 \in C$ which defines a routing subfunction R_1 and a pair of channels $c_i, c_j \in C$, there is an *indirect dependency* from c_i to c_j iff it is possible to establish a path from s_i to d_j for messages destined to some node n. The first and last channels in that path are c_i and c_j, and they are the only ones supplied by R_1. As c_i and c_j are not adjacent, some other channels belonging to $C - C_1$ are used between them.

Definition 7:

A *channel dependency graph* D for a given interconnection network I and a routing function R is a directed graph $D = G(C, E)$. The vertices of D are the channels of I and the arcs of D are pairs of channel (c_i, c_j) such that there is a direct dependency from c_i to c_j.

Definition 8:

The *extended channel dependency graph* D for a given interconnection network I and a routing subfunction R_1, is a directed graph, $D_E = G(C_1, E_E)$. The vertices of D_E are the channels supplied by the routing subfunction R_1. The arcs of D_E are the pairs of channels (c_i, c_j) such that there is either a direct dependency or an indirect dependency from c_i to c_j.

Definition 9:

A *configuration* is an assignment of a list of flits to each queue, all of them belonging to the same packet. A configuration is *legal* iff the queue capacity is not exceeded and all the flits stored in the queue have been sent there by the routing function.

Definition 10:

A *deadlocked configuration* for a given interconnect network I and routing function R, is a non-empty legal configuration in which no header flit is one step from its destination and no header flit can advance because the queues for all the alternative output channels supplied by the routing function are full. Data and tail flits cannot advance because the next channel reserved by their packet header has a full queue. Also, in a deadlocked configuration, there is no packet whose header flit has already reached its destination.

We make the following assumptions about our implementation.

Assumption 1:

The router is implemented with internal flow control such that output buffers clear.

Assumption 2:

All buffers have access to the Deadlock Buffer at neighboring nodes.

Assumption 3:

A packet that has been placed on the Deadlock Buffer at a node can use only Deadlock Buffers at subsequent nodes until it is delivered at its destination. The channels formed using the Deadlock Buffers constitutes the set C_1. In *Disha*, routing on the Deadlock Buffers is restricted to be minimal.

Assumption 4:

Mutual exclusion on the Deadlock Buffers is implemented with a Token[1]. There can be at most one Token travelling across all nodes in the network at any given instant of time,

Assumption 5:

Deadlock Buffers can be used by packets only to recover from deadlocks. A packet can be routed on the Deadlock Buffers iff the propagating Token has been captured by that packet. Instantaneous with this action, the token is inhibited from further propagation until deadlock is resolved.

Assumption 6:

The captured Token is released by the destination node that receives a packet header on the Deadlock Buffer. This packet is the one that was originally deadlocked and initiated the need for a Token capture.

Lemma 1:

The routing subfunction R_1 is connected.

Proof:

Every node is provided with a Deadlock Buffer. From Assumptions 2 and 4, it is obvious that given any pair of nodes *(x, y)* it is possible to establish a path from *x* to *y* using the Deadlock Buffers. Thus R_1 is connected.

Lemma 2:

There are no cycles formed by indirect dependencies in the extended channel dependency graph.

Proof:

Assumption 3 implies that we cannot find a pair of channels (c_i, c_j) supplied by R_1 such that the channels between them are not supplied by R_1. If c_i is supplied by R_1 then all subsequent channels must be supplied by R_1. Consequently, there are no indirect dependencies in the extended channel dependency graph.

Lemma 3:

The routing subfunction R_1 has no cycles in its extended channel dependency graph.

Proof:

Deadlock Buffers constitute the set C_1. From Assumption 6, at any given instant of time, there can be at most one packet P_i on the Deadlock Buffers that does not have its header delivered at its destination. Other packets on the Deadlock Buffers have at least one flit delivered at their respective destinations. These packets cannot be involved in any deadlocked configuration (Definition 10). From Assumption 3 routing on the Deadlock Buffers is restricted to be minimal. Hence, packet P_i cannot be deadlocked with itself. Thus none of the packets that are currently on Deadlock Buffers are involved in any deadlocked configuration. Lemma 2 shows that there are no cycles formed by indirect dependencies. It follows that a deadlocked configuration does not involve any of the Deadlock Buffers. Therefore, there exists a subset of channels C_1 that defines a routing subfunction R_1 which has no cycles in its extended channel dependency graph.

Theorem 1:

An adaptive routing function R for an interconnection network I is deadlock-free if there exists a subset of channels $C_1 \subseteq C$ that defines a routing subfunction R_1 which is connected and has no cycles in its extended channel dependency graph D_E.

Proof:

The reader is referred to Duato[12] for a formal proof.

Theorem 2:

"The network can safely recover from deadlocks when deadlocked packets are sequentially routed on the Deadlock Buffers."

Proof:

Lemmas 1 and 3 prove that the routing function R_1 is connected and has no cycles in its extended channel dependency graph. The theorem is proved by simply inserting these results into Theorem 1.

■

To get an intuitive feel of the proof, assume that the network is deadlocked. There may be one or more dependency cycles. Assume that there are 'k' dependency cycles. A packet can be placed on the cycle-free Deadlock Buffer path. Following this path the packet is guaranteed to reach its destination where it will be sunk. This eliminates one cycle, reducing the number of cycles to (k-1). By induction, it follows that the network safely recovers from deadlocks.

References:

[1] 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, IEEE* Computer Society, April 1995.

[2] Anjan K. V., Timothy Mark Pinkston, and Jose Duato. Concurrent Deadlock Recovery in *Disha*. Submitted to the *International Conference on Computer Design*, October 1995.

[3] Anjan K. V. and Timothy Mark Pinkston. *DISHA*: An Efficient, Fully Adaptive Deadlock Recovery Scheme. *CENG Technical Report 94-23, Department of Electrical Engineering - Systems, University of Southern California,* Los Angeles, CA 90089-2562, November 1994.

[4] Andrew A. Chien and J.H. Kim. Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors. In *Proceedings of the 19th International Symposium on Computer Architecture, IEEE* Computer Society, pages 268-277, May 1992.

[5] Andrew A. Chien. A Cost and Speed Model for k-ary n-cube Wormhole Routers. In *Proceedings of the symposium on Hot Interconnects, IEEE* Computer Society, August 1993.

[6] K. Aoyama. Design Issues in Implementing an Adaptive Router. *Master's Thesis, University of Illinois*, Department of Computer Science, 1304 W. Springfield Avenue, Urbana, Illinois., January 1993.

[7] P. Berman, L. Gravano, G. Pifarre, and J. Sanz. Adaptive deadlock and livelock free routing with all minimal paths in torus networks. In *Proceedings of the Symposium on Parallel Algorithms and Architectures,* 1992.

[8] R. V. Boppana and S. Chalasani. A Comparison of Adaptive Wormhole Routing Algorithms. In *Proceedings of 20th International Symposium on Computer Architecture, IEEE* Computer Society, pages 351-360, May 1993.

[9] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36(5): 547-553, May 1987.

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

[11] W. Dally and H. Aoki. Deadlock-free Adaptive Routing in Multicomputer Networks using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems,* Vol. 4, No. 4, pages 466-475, April 1993.

[12] 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.

[13] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. In *Proceedings of the International Conference on Parallel Processing*, CRC Press, pages I142-I149, August 1994.

[14] J. Duato. Deadlock-Free Adaptive Routing Algorithms for the 3D-Torus: Limitations and Solutions. In *Proceedings of Parallel Architectures and Languages Europe 93*, June 1993.

[15] J. Duato. Improving the Efficiency of Virtual Channels with Time-Dependent Selection Functions. In *Proceedings of Parallel Architectures and Languages Europe 92*, June 1992.

[16] Patrick T. Gaughan and Sudhakar Yalamanchili. Adaptive Routing Protocols for Hypercube Interconnection Networks. *IEEE Computer*, pages 12-22, May 1993.

[17] J. Kim, Z. Liu, and A. Chien. Compressionless Routing: A Framework for Adaptive and Fault-tolerant Routing. In *Proceedings of the 21st International Symposium on Computer Architecture, IEEE* Computer Society, pages 289-300, April 1994.

[18] S. Konstantinidou and L. Snyder. Chaos Router: architecture and performance. In *Proceedings of the 18th International Symposium on Computer Architecture*, pages 212-221, May 1991.

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

[20] L. Ni and C. Glass. The Turn Model for Adaptive Routing. In *Proceedings of the 19th International Symposium on Computer Architecture, IEEE* Computer Society, pages 278-287, May 1992.

[21] L. Ni and P.K. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. IEEE *Computer*, pages 62-76, February 1993.

[22] Timothy Mark Pinkston. Design Considerations for Optical Interconnects in Parallel Computers. In *Proceedings of the First International Workshop on Massively Parallel Processing using Optical Interconnects, IEEE* Computer Society, pages 306-322, April 1994.

[23] Douglas S. Reeves, Edward F. Gehringer, and Anil Chandiramani. Adaptive Routing and Deadlock Recovery: A Simulation Study. In *Proceedings of the 4th Conference on Hypercube Concurrent Computers and Applications,* March 1989.