Effective Use of Virtual Channels in Wormhole Routed Distributed Shared Memory Systems*

Donglai Dai and Dhabaleswar K. Panda
Dept. of Computer and Information Science
The Ohio State University, Columbus, OH 43210-1277
Tel: (614)-292-5199, Fax: (614)-292-2911
E-mail: {dai,panda}@cis.ohio-state.edu

Contact Author: Dhabaleswar K. Panda

Abstract: Multiple virtual channels have been studied extensively and implemented as a powerful mechanism for obtaining high performance in modern interconnection networks. However, the effective use of virtual channels in distributed shared memory (DSM) systems is non-trivial. In this paper we present two simple techniques for exploiting the promising performance benefit of virtual channels in DSM systems. First, we propose to use unequal number of virtual channels in the request and reply networks to harness performance benefit from the inherent unbalanced traffic in DSM systems. Next, we apply the virtual channel mechanism to injection and consumption channels to alleviate the performance bottleneck at the network interface. In addition, issues related to virtual channels are taken into account in designing non-FIFO coherence protocols. These new design strategies are evaluated through simulation using a set of representative benchmark applications. The effect of critical system parameters (such as node cache line size, switch routing delay, and network topology) on the new designs is also studied. Overall, our results show that with an appropriate non-FIFO cache coherence protocol, virtual channel mechanism can considerably reduce the time for shared memory accesses and the overall execution time of DSM applications. This study demonstrates that a carefully coordinated design strategy using the virtual channel mechanism can significantly improve the performance of distributed shared memory systems.

Keywords: Parallel architecture, distributed shared memory systems, performance modeling, interconnection networks, wormhole routing, topology, network interface, directory-based protocols, and cache coherence.

*This research is supported in part by NSF Career Award MIP-9502294 and NSF Grant CCR-9704512.

1 Introduction

Distributed Shared Memory (DSM) multiprocessors are emerging as the trend in building parallel systems because they provide much desired programmability [12, 11, 15, 13, 20]. However, the current generation DSM systems using wormhole interconnection network do not deliver very good performance due to the high latency associated with shared memory accesses. A significant portion of this latency is incurred by messages in the interconnection network [6], and a lot of research is currently taking place to reduce this latency [19, 27, 5, 22].

Virtual channels [7] have been proposed as an attractive mechanism to avoid deadlocks [1, 8], provide adaptivity [4], increase throughput [9], and reduce latency in wormhole routed networks [9]. Most of these studies on using virtual channels mechanism in wormhole networks have focused on synthetic traffic (uniform/non-uniform/hot-spot), arbitrary message lengths, and message generation intervals following some probabilistic distributions. Some crucial characteristics of DSM systems including the *cause-effect* relationship between messages (remote read/write request message followed by a reply message), *bi-modal* traffic (short messages reflecting control messages in a cache-coherency protocol and long messages reflecting transfer of cache-lines), and the periodic generation of messages by processors (based on the computational granularity) have not been considered. Most of these studies also ignore modeling network interface and assume that it contains infinite number of physical injection and consumption channels. Under such an assumption, no bandwidth bottleneck can ever occur between processing nodes and the network. Thus, the current studies on virtual channels do not provide any concrete guideline about using virtual channels in DSM systems.

In recent years, several commercial network switching products [17, 24, 16] have incorporated virtual channel mechanism as a major feature for improving network performance. Out of these products, the SGI Spider interconnect [17] is directly geared towards cache-coherent DSM systems. However, this study [17] does not clearly outline the performance benefit of virtual channels in DSM systems.

Recently, an application-driven study [27] has shown that only a negligible performance benefit exists in using virtual channels and adaptive routing in a DSM system. According to this study, the enhancements due to virtual channels and adaptive routing might not be justified considering the associated increase in router complexity. However, four important factors have been overlooked in this study:

- An equal number of virtual channels have been used in the request and reply networks, ignoring the fact that a typical request message is much shorter than a typical reply message in of a DSM system. The request and reply messages travel in separate (virtual) networks to avoid deadlocks [12]. Since the volume of traffic is not the same in the request and reply networks, it is not clear whether equal number of virtual channels assigned to each of these two networks can deliver performance.
- Network interfaces having a single injection channel and a single consumption channel have been used to connect nodes to routers having multiple virtual channels. This puts a severe limitation on the number of live messages in the network at any given time. Thus, increasing the number of virtual channels inside the network does not promise performance benefit.

- In-order message delivery has been enforced by the network interface adapter, and a simple FIFO cache coherence protocol has been used. In such a design, messages arriving out-of-order (due to virtual channels) have to wait at the network interface adapter until the arrivals of their logical preceding messages. Such an approach prevents the directory controller of the destination node from processing these messages in a timely fashion. This results in an increase in effective message latency and nullifies the latency reduction gained by the virtual channel mechanism.

- A performance degradation model [2] for implementing the virtual channel mechanism has been used in the evaluation. However, new techniques [17] are available today to design routers with a moderate number of virtual channels (up to 4 or 5) without increasing the routing decision delay for the header flit. Using these techniques, all tail flits can be properly pipelined and moved at the full link speed regardless of the routing delay of the header flit.

This leads to a set of interesting problems: 1) whether virtual channels can provide performance benefit for DSM systems and 2) how to allocate a set of virtual channels between the request and reply networks to obtain maximum performance benefit.

In this paper, we take on such challenges and study how to effectively use the virtual channel mechanism in DSM systems. First, we propose two design guidelines: 1) having a larger number of virtual channels in the reply network than that in the request network and 2) using an equal number of virtual channels inside the network and at the network interface. Second, we look at the related issues in coherence protocols and discuss why a non-FIFO protocol is very desirable in a DSM system using virtual channels. Third, we perform application-driven simulations to evaluate various design alternatives following the above guidelines. We show that a network configuration consisting of one request virtual channel, two reply virtual channels, and three virtual channels per injection/consumption channel provides the best design from a cost-performance perspective. Finally, we study the impact of several critical system parameters (such as cache line size, switch routing delay, and network topology) on the new designs. Overall, our results show that with an appropriate non-FIFO cache coherence protocol, using virtual channels can significantly reduce shared memory access waiting time and improve system performance. This study demonstrates that a carefully coordinated design using the virtual channel mechanism can considerably improve the performance of a distributed shared memory system. This study also provides valuable understanding on tradeoffs for designing efficient DSM systems using virtual channels.

The rest of this paper is organized as follows. Section 2 reviews issues related to virtual channels in DSM systems. Section 3 estimates communication requirements and identifies the limitations in the current DSM systems when multiple virtual channels are used. Section 4 proposes three solutions for resolving each of the limitations. Results of our simulation evaluation on design alternatives are discussed in Section 5. Section 6 briefly reviews related work. Concluding remarks are made in Section 7.

2 Current Generation DSM Networks and Coherence Protocols

In this section we provide a brief overview of the current generation DSM systems and focus on the important issues related to the networks and cache coherence protocols.

2.1 Virtual Channels and Virtual Lanes

Virtual channels [7] over a physical channel have been proposed and implemented [17, 24] as a mechanism to avoid deadlocks and improve throughput in wormhole routed networks. These channels divide the buffering storage at a router and compete (on demand) for the bandwidth of the physical channel. A group of such virtual channels are called *virtual lanes* if any member of this group can be used for transferring a message [9]. A message (or worm) consists of a set of flits and uses one virtual channel at each hop of the network while propagating towards its destination. Thus, the flits of a worm form a pipeline across routers of a network. The routing algorithm associated with the routers decides which virtual channel to be used by the worm at each hop based on its destination. In this paper, we concentrate on the popular dimensional order routing [21].

2.2 Dual Request-Reply Networks

As mentioned earlier, the request and reply messages in a DSM system form strong cycle dependencies with one another. For instance, a dependency exists between a read request message and the subsequent reply message containing the desired data. To avoid deadlocks caused by such dependencies, two logically separate networks are commonly used in practice [12, 13]. One of the two networks, known as the *request network*, is used to transfer all types of request messages. The other, known as the *reply network*, is used to transfer all types of reply messages.

Earlier research has indicated that the request and reply networks that are physically separated (duplicated links and routers) operate far below their saturation point most of the time during the execution of typical applications [13, 20]. To be cost-effective, the current generation DSM systems tend to employ dual virtual networks. A *virtual network* refers to a network in which the logical links are implemented using the virtual channel mechanism. If there are an equal number of virtual lanes between any two neighboring routers in the virtual network, it is called a multi-lane virtual network. The dual request-reply virtual network used in a DSM system is constructed as follows. The group of virtual channels over a single physical channel are partitioned into two non-overlapping sets. Each set is assigned to one virtual network, forming one logical link in that virtual network. Similar partition and assignment can be done for all the virtual channels over each physical channel, thus, resulting in two sets of logical links consisted of non-overlapping virtual channels. These two sets of logical links connect the routers and form the two logically separate virtual networks. It is to be noted that this dual virtual network shares the same set of physical routers and the same set of physical channels. As an example, Fig. 1 shows a dual request-reply virtual network on a 2x3 mesh-connected DSM system. The most important restriction of a dual virtual network is that any worm traveling in one virtual network can not use any virtual channel belonging to the other virtual network.

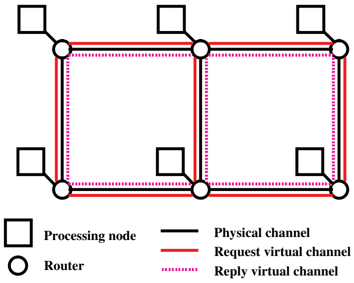

Figure 1: An example of 2x3 mesh-connected DSM system. In this system, the dual virtual network contains a single lane in the request network and a single lane in the reply network. All the channels are bidirectional.

2.3 Network Interface (NI)

A conventional network interface adapter, used in a DSM system for transferring messages between a processing node and the network, is shown in Fig. 2. This adapter contains DMA engines, a number of message sending/receiving buffers, the local bus interface, and the network interface (NI). The DMA engines use local bus interface (in full duplex or half duplex) to move entire messages between the sending/receiving buffers at the network adapter and the main memory in the processing node. The network interface is used to transfer the flits of messages between the sending/receiving buffers at the network adapter and the flit size buffers within the associated router/switch. The DMA engines synchronize the accesses to the the sending/receiving buffers between the local bus interface and the network interface. The network interface contains a (physical) *injection channel* and a *consumption channel*. A network interface is designed in such a way that the injection/consumption channel must be reserved exclusively for the time period when an entire message is injected or consumed.

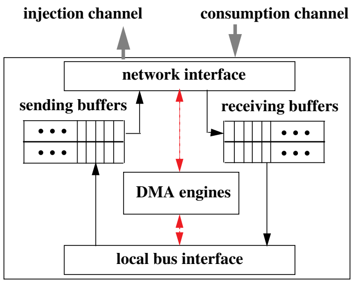

Figure 2: A conventional network interface adapter used in typical DSM systems. The network interface within the adapter contains a single physical injection channel and a single physical consumption channel.

2.4 Cache Coherence Protocols

Traditionally, directory-based write-invalidate cache coherence protocols have been used in DSM systems [12, 13]. Most of these protocols are designed by assuming the property of in-order message delivery. Such coherence protocols are commonly identified as FIFO coherence protocols. The property of in-order message delivery requires that messages between any pair of processing nodes are delivered to the destination in the order that they are injected into the network at the source. A network guaranteeing such in-order delivery is typically defined as a FIFO network. The main motivations for assuming a FIFO network in a coherence protocol are the following: 1) a simple protocol leads to lower cost in implementation; and 2) the correctness of the protocol can be proved by hand.

The justification for these motivations can be illustrated by an example as shown in Fig. 3. Consider a node N_1 issuing a write-miss request ($Rxq(1)$) to a memory block B_1 whose home node is N_2 and whose directory state is dirty-remote [12, 13]. Once the request Rxq arrives at node N_2, a flush request ($Fsh(2)$) is issued by N_2 to the current owner node N_3. Assume there is a conflict in the cache of N_3 and the dirty copy of B_1 needs to be sent back ($Wbk(3a)$) to N_2 right before the arrival of Fsh. Once Fsh reaches node N_3, a reply ($Frp(3b)$) is sent back by N_3 to N_2 as the acknowledgment. If the underlying network is a FIFO network, the message Wbk always reaches node N_2 before the message Frp. Thus, whenever Frp arrives at N_2, the reply ($Rxp(4)$) can be sent immediately to node N_1. On the other hand, if the underlying network is not a FIFO network, message Frp can arrive at N_2 either before or after Wbk. Thus, when Frp arrives, N_2 has to check if it has the latest copy of B_1. If so, the reply Rxp can be sent. Otherwise, other coherence actions like rejects or bookkeeping need to be taken. Similar complications also occur at the arrival of Wbk.

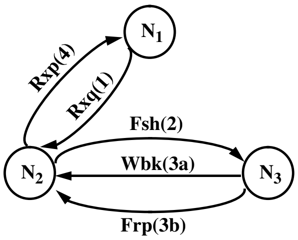

Figure 3: An example for the justification of FIFO coherence protocols.

3 Limitations of Current Approach

In this section we discuss the limiting factors in the design of current generation DSM systems.

3.1 Unbalanced Usage of Request-Reply Networks

The communication characteristics of a DSM system are very sensitive to three fundamental factors: the system configuration, the coherence protocol, and the application. Generally, the system configuration decides the sizes of messages. The coherence protocol decides the types and the number of messages for each basic transaction. The application decides the total number and the mixture of various basic transactions.

3.1.1 Sizes of Short and Long Messages

Two kinds of messages (short and long) are observed in a DSM system. Short messages are used for exchanging control information, and long messages are used for data transfer. A short message normally contains items like the type of the message, the node ID of the requester, and the block address of the global memory concerned. It also contains extra bits for network routing and fault-tolerance. A long message normally contains a complete copy of a memory block (cache line) in addition to all the information required for a short message. Thus, once a system configuration is fixed, the sizes of the short and long messages in the system are fixed. Let us denote the sizes of these messages as L_s and L_l, respectively.

3.1.2 Categories of DSM Messages

Regardless of the specific directory-based write-invalidation coherence protocol used in a DSM system, all coherence messages can be classified into four generic categories: authority, replacement, invalidation, and adjustment. The first category contains the basic types of messages that involve the permissions for using data. The second category contains the types of messages related to cache replacement. The third contains the messages occurring only in cache invalidation. The last category contains the messages related to accessing semi- or fully-locked cache lines or main memory blocks. Most known cache coherence protocols use a subset or variation of these four categories of messages. Table 1 shows the characteristics of traffic generated by each of these four categories in the request and reply networks. The numbers denoted in the table is defined as the total number of messages in each category during the complete execution of an application.

Table 1: The characteristics of traffic generated by each of the four generic categories of cache coherence messages in the request network and the reply network.

	Request Network			Reply Network		
Category	Typical Message	Size	Number	Typical Message	Size	Number
Authority	shared-read-request exclusive-read-request exclusive-upgrade-request update-home-request	L_s	n_1^q	shared-data-reply exclusive-data-reply exclusive-grant[1] update-data-reply	L_l	n_1^p
Replacement				dirty-replacement clean-replacement	L_s	n_2^p
Invalidation	invalidation-request	L_s	n_3^q	invalidation-ack	L_s	n_3^p
Adjustment				reject-signals wait-signals	L_s	n_4^p

3.1.3 Loads in Request-Reply Networks

Let us estimate the traffic loads (volumes) in the request and reply networks. We denote the load in the request network as V^q and the load in the reply network as V^p. Using the notations

[1]This is a short messge.

from Table 1, we have the following equations:

$$V^q = n_1^q L_s + n_3^q L_s \quad (1)$$

$$V^p = n_1^p L_l + n_2^p L_s + n_3^p L_s + n_4^p L_s \quad (2)$$

Assuming only point-to-point messages are used in the DSM system (based on current DSM systems), the following relationships hold.

$$n_1^q = n_1^p + n_4^p \quad (3)$$

$$n_3^q = n_3^p \quad (4)$$

In typical DSM systems, it can be assumed that $n_1^p \gg n_2^p$, $n_1^p \gg n_3^p$, and $n_1^p \gg n_4^p$. Combining these assumptions and Equations (1-4), we have the following approximation:

$$V^q/V^p \approx L_s/L_l \quad (5)$$

Equation 5 indicates that the traffic load in the reply network is much heavier than that in the request network.

3.1.4 Latencies in Request-Reply Networks

Let us denote the average sizes of the messages in the two networks as L^q and L^p, respectively. From Equations 1 and 2, we have:

$$L^q = V^q/(n_1^q + n_3^q) = L_s \quad (6)$$

$$L^p = V^p/(n_1^p + n_2^p + n_3^p + n_4^p) \approx L_l \quad (7)$$

Assume the execution time of the application as T_{exec} and the number of nodes in the DSM system as N. Thus, the effective received load per node per time unit in the two networks, denoted as ρ^q and ρ^p respective, can be expressed as:

$$\rho^q = V^q/(N * T_{exec}) \quad (8)$$

$$\rho^p = V^p/(N * T_{exec}) \quad (9)$$

From Equations 5, 8, and 9, we have the following ratio:

$$\rho^q/\rho^p \approx L_s/L_l \quad (10)$$

Assume that the request and reply networks are physically separated and the nature of traffic in both network is uniform. The average message latencies in these two networks can be illustrated as shown in Fig. 4 according to the theoretical performance model of wormhole network with contention [9]. It indicates that the reply network typically operates at a higher load due to heavier traffic and leads to increased latency for reply messages.

It is to be noted that the interaction between messages in the two virtual networks when they share a single physical network has not be accounted in Fig. 4. Thus, it is quite desirable to reduce the latency of both the request and the reply message. However, the above discussion indicates that allocating equal resources (virtual channels) between the request and reply networks, as all DSM systems do, is not a good design strategy.

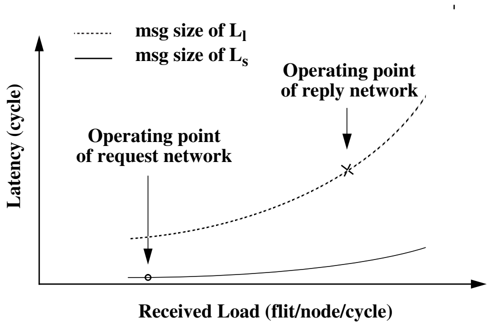

Figure 4: Comparing the theoretical latencies of messages in the request and reply networks by assuming these two networks being physically separated.

3.2 Bottleneck at the Network Interface (NI)

Now, we focus on the limitation of current generation NI. Communication in a network with multiple virtual channels using the conventional NI is done in the following way. First, at the sending side, a message is constructed within the sending buffers at the NI. Second, the (physical) injection channel at the NI gets reserved, and then a worm associated with the message starts moving into the network in a flit by flit fashion. As the header flit of the worm moves through each router towards its destination, a virtual channel gets reserved at each hop. Once the header flit reaches the NI of the destination, the (physical) consumption channel and a receiving buffer gets reserved. Finally, all the tail flits move towards the destination in a pipelined fashion.

During the entire process, contention can occur whenever the injection channel, the consumption channel, or any virtual channel inside the network is not available. Bubbles are inserted into the worm when some of the reserved virtual channels share the underlying physical channels with other worms in the network. All these factors contribute to an increase of the time interval that an injection/consumption channel must be reserved for transferring a message. Such increase of the time interval to transfer a message forces an injection/consumption channel to remain idle for a longer period of time. Thus, the usage of virtual channels in the network may reduce contention inside the network. But it aggravates contention at the NI. An example of such contention is illustrated in Fig. 5. Such contention at the NI has also been identified in the context of distributed memory systems in a separate research [3].

3.3 FIFO Restriction

As the VLSI technology advances, implementing a coherence protocol with moderate complexity is no longer a concern. New automatic or semi-automatic tools are available to help checking the correctness of the protocol. A FIFO coherence protocol is a natural fit in a small-scale DSM system where a single bus or switch is used. However, as the size of DSM systems grows, the network becomes more complex. Latest network technology tends to provide a much higher bandwidth and lower latency when the restriction of in-order message delivery is lifted.

It is possible to use a high-performance non-FIFO network in a DSM system together with a FIFO cache coherence protocol which enforces in-order delivery within the network

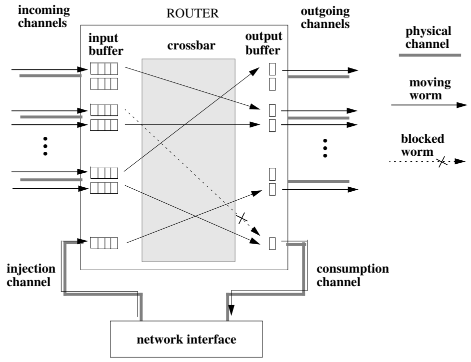

Figure 5: An example snapshot of a typical router and the network interface supporting one physical injection/consumption channel. There are three concurrently bypassing worms in the router. One worm is being injected from the NI to the router into the network. Two worms are terminating at this router.

adapters. As discussed earlier, any message arriving out-of-order has to wait in the network adapter until the arrival of its preceding messages. However, such an approach prevents the node (directory) controller from processing the messages in a timely fashion and leads to performance degradation.

4 Proposed Solutions

In this section, we propose a set of solutions which remove the limitations discussed in the previous section. These solutions lead to better designs for DSM systems. In the following section, we evaluate the new designs through simulation and establish guidelines for designing better systems.

4.1 Unbalanced Partitioning of Virtual Channels

As mentioned earlier, the contention in one virtual network (say the reply network) can affect the traffic of the other (the request network) because a single physical network is shared. In Section 3, we observed that the load in the reply network is considerably higher than that in the request network. Thus, for obtaining better performance, our basic idea is to reduce the average message latency in the reply network significantly with no or very small increase in the average message latency in the request network. Intuitively, more resources are needed in the reply network than the request network due to the higher volume of traffic in the reply network. We propose to accomplish this by using more virtual channels in the reply network. This provides the DSM system of having: 1) more number of reply messages (worms) in the system at any given time and 2) more opportunities for a reply worm to move around when contention occurs. Given a set of virtual channels per physical channel, an optimal unbalanced partitioning of the channels between the request and reply networks is difficult to derive using analytical modeling because it depends on too many factors. We will evaluate such interesting design alternatives using simulation in Section 5.

4.2 Supporting Virtual Injection/Consumption Channels at the NI

We propose to alleviate the bottleneck at the NI by supporting the virtual channel mechanism over the physical injection/consumption channel. We call these new virtual channels as *virtual injection/consumption channels*. At the router side, the implementation requires that the port used by injection/consumption channel be designed in a way similar to any ports in the router. This is easy to achieve. At the NI side, the implementation can be done with little additional cost. Since there are only two different sizes of messages (both relatively small), the total storage required for multiple simultaneous sending/receiving buffers is quite affordable. Simple control logic can be devised to operate the sending/receiving buffer as a set of small parallel units instead of a large strict FIFO. With such virtual channels at the NI, a worm is allowed to use the physical injection/consumption channel to transfer the flits from/into a different sending/receiving buffer when its preceding messages have not yet been entirely injected/consumed. Therefore, more than one worm can be injected/consumed at the NI at the same time in an interleaved manner.

4.3 Employing a Non-FIFO Coherence Protocol

With a careful examination, two important observations can be drawn about coherence protocols. First, the order of the messages corresponding to different memory blocks between any pair of nodes is not important for correctness. This observation means that the FIFO order is usually not required. This is because the majority of the messages being transferred between a pair of nodes during a short time period correspond to different memory blocks. Second, even for messages corresponding to the same block, it is helpful to have the node (directory) controller process an out-of-order message immediately. Such an out-of-order event occurs most often when a short message bypasses a long message. To process such a message requires modifying the status information only in the directory. Often this can be accomplished when the node controller is idle. Once the expected message arrives, the node controller can do the processing, which combines the effects of the two messages with almost the same amount of overhead as needed in processing the latter message alone.

The non-FIFO coherence protocol used in this study is somewhat similar to the one used in SGI Origin [20]. The most noticeable difference is that a normal four-node scheme is employed in our coherence protocol instead of the three-node forwarding scheme. Specifically, for a write request to an actively shared block, all invalidation acknowledgments are collected at the home node and the reply granting the ownership is sent from the home node. Similar sequences of actions occur for a read or write to an outstanding dirty block. A four-node scheme simplifies the design of the node architecture and the coherence protocol. However, similar performance benefits of the three-node scheme can be obtained when it is implemented.

5 Simulation Experiments and Results

In this section we present detailed simulation results comparing the performance of DSM system using various network and network interface configurations. These configurations are based on our solutions proposed in the previous section.

5.1 Simulation Experiments

This section describes our basic methodology and the default system and application parameters used in simulation experiments.

5.1.1 Simulation Environment

The hardware cache coherent multiprocessor we simulated has an architecture similar to the FLASH machine [13]. The system consists of 64 processing nodes interconnected in a topology of K-ary N-cube (2D mesh as default except when we study the impact of topology). Each node contains one processor. The processor is assumed to be a 300 MHz single-issue microprocessor with a perfect instruction cache and a 128 KB 2-way set associative data cache with a line size of 128 bytes. The cache is assumed to operate in dual-port mode using write-back and write-allocate policies. The instruction latencies, issue rules, and memory interface are modeled based on the DLX design [18]. The memory bus is assumed to be 8 bytes wide. On a memory block access, the first word of the block is returned in 30 processor cycles (100 ns). The successive words in the block follow in a pipelined fashion. There are 8 coalescing transaction buffers at each node used for resolving outstanding remote memory requests. Each buffer can hold one cache line and merge multiple writes into the line. The machine is assumed to be using a non-FIFO coherence protocol as discussed in section 4.3 and supporting the release consistency memory model [13, 20]. The synchronization protocol assumed in the system is the QOLB protocol, which is similar to the one used in DASH [12]. The node simulator models the internal structures and the queuing and contention at the node controller, the main memory, and the cache [23]. Table 2 shows the default memory hierarchy parameters, node controller occupancy delays, network parameters, and network interface parameters used in our simulations.

Table 2: Default system parameters used in simulation.

Memory Hierarchy Parameters		Network Parameters	
Processor frequency	300MHz	Network frequency	200MHz
Cache access	1 cycle	Channel width / Flit size	2 bytes
Cache line size (L)	128 bytes	Link Propagation	1 net cycle
Cache set associativity	2	Router switch delay	1 net cycle
Cache size per node	128 Kbytes	Routing delay	3 net cycles
Memory word width (W)	8 bytes/cycle	Physical network	1
Memory response delay	30 cycles	Virtual networks	2
Cache block fill time	30+L/W cycles	Topology	2D
Memory block access time	30+L/W cycles		
Size per trans. buffer	L bytes		
No. of trans. buffers	8		
Node Controller Occupancy		**Network Interface Parameters**	
Directory check	7 cycles	Outgoing message startup	15 cycles
Directory check and update	14 cycles	Incoming message dispatch	8 cycles
Each invalidation request	12 cycles	Control message size	6 bytes
Message forward	3 cycles	Data message size	6+L bytes

We used four applications — FFT, Radix, Barnes, and LU — in our simulation evaluations. All are real applications or challenging computational kernels ported from the Stanford SPLASH2 [25, 14] benchmark suite. Table 3 shows the basic communication patterns and problem sizes for the applications. These applications were compiled by *dlxcc* using the optimization level equivalent to O2 of *gcc*.

Table 3: Applications and the used input sizes.

Application	Communication Pattern	Problem Size
FFT	all-to-all, regular	$64K$ complex points
Radix	all-to-all, irregular	$1M$ keys, $1K$ radix, max $1M$
Barnes	hierarchical, irregular	$8K$ particles, $\theta = 1.0$, 4 time steps
LU	one-to-many, regular	512×512 doubles, 8×8 blocks

5.1.2 Network Configurations Evaluated

From earlier discussion, it is clear that there are many possible network configurations with a given number of virtual channels. For example, consider a DSM system which supports four virtual channels. Three different partitions of the virtual channels are possible. A (3,1) partition has three lanes in the request network and one lane in the reply network. The other two partitions can be denoted as (2,2) and (1,3). Now, the question is which partitioning scheme provides the best performance? Due to the complicated interference between the two virtual networks, the answer to this question is non-trivial. Additionally, we have the flexibility of using virtual injection/consumption channels at the NI. Since the number of lanes per physical injection and consumption channel are independent of each other and independent of the number of virtual channels inside the network, it leads to a large number of network configurations. To keep this simulation study under control, we evaluated the networks with moderate number (up to 5) of virtual channels.

For convenience of discussion, let us denote a network configuration by a tuple (q, p, i, c), where q, p, i, c denote the number of virtual lanes in the request network, reply network, per injection channel, and per consumption channel, respectively. When there is only one physical injection or consumption channel, $i = 1$ or $c = 1$ is used.

5.1.3 Performance Metrics

We present all our simulation results in two sets: the execution time and the network latency. This helps us to correlate the behaviors of the underlying network with the overall DSM system performance and to gain important insights into the network design issues. The overall execution time is broken down into four components: the CPU computation time (C), the memory read waiting time (R), the memory write waiting time (W), and the synchronization waiting time (S). All the times we present are normalized to that of the left most system configuration for each application except when stated otherwise. The network latencies for two types (request and reply) of messages are presented in absolute time (microseconds). The presented latencies are the average of the measured raw network latencies globally clocked from the start of the injection of the first flit at the sender till the completion of the consumption of the last flit at the receiver across all the messages of the same type.

5.2 Simulation Results

In this section, we first evaluate the performance benefit of virtual injection/consumption channels. Next, we compare the performance of various partitioning of virtual channels in the network. Finally, we examine the impact of three critical system parameters on the overall performance.

5.2.1 Effect of Virtual Injection and Consumption Channels

We first studied the effect of virtual injection channels and virtual consumption channels on the overall execution time of each benchmark application. The four configurations used in the experiments were: (1,1,1,1), (1,1,2,2), (2,2,1,1), and (2,2,4,4). The results are shown in Fig. 6.

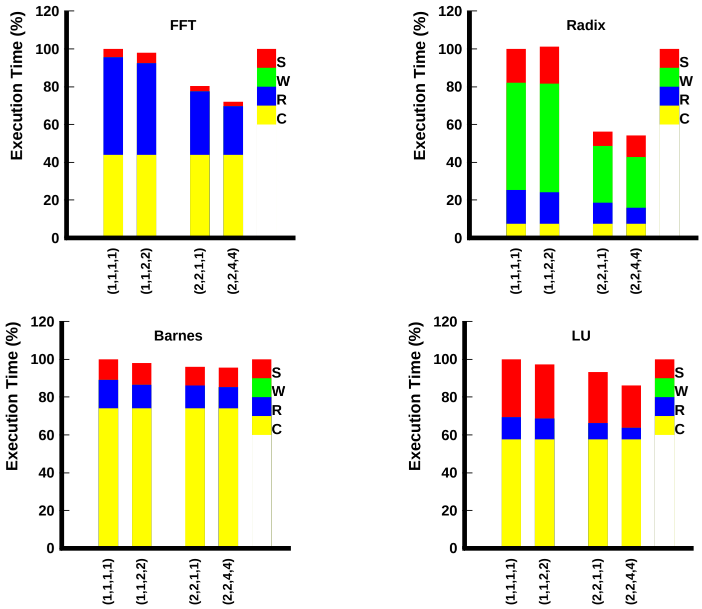

Figure 6: The effect of virtual injection and virtual consumption channels on the overall execution time of benchmark applications.

It can be observed that the overall execution time was reduced in all but one case when virtual injection/consumption channels were used compared to those when they were not. As the number of virtual channels increases and the bandwidth of the underlying physical channel remains constant, the overall system performance improves for all applications. The actual reduction in execution time varies across applications, with a maximum of 10.4% seen in FFT from configuration (2,2,1,1) to configuration (2,2,4,4).

One exception occurred in Radix when the configuration was changed from (1,1,1,1) to (1,1,2,2). Instead of decreasing, the overall execution time increased by 1.1%. The reason for this abnormality could be due to a change in the critical execution path of the application.

From the timing breakdowns, it can be observed that the CPU computation time remained almost constant across all network configurations in every application. This is expected for the following reason. The amount of computation per processing node is largely fixed for a given input data set and system size for these applications. Improving network performance helps to cut down various waiting times but not the CPU computation times.

It can be also observed that for all applications except Radix the write waiting time is negligible. The results indicate that our non-FIFO coherence protocol enforcing the release consistency model with a moderate number of transaction buffers is quite effective. Similar results can also be observed from remaining experiments.

To gain more insights into the direct effect of virtual injection/consumption channels on the network performance, let us examine the changes in the average message latencies in the request and reply networks. Figure 7 shows the results from the same set of experiments. It can be observed that the average message latency of request/reply network is usually smaller in systems which used virtual injection/consumption channels compared to that in systems which did not use them. This trend is more prominent for systems with two virtual channels in each network. This indicates that an incoming/outgoing message is more often blocked by another incoming/outgoing message at the NI in a system which supports virtual channels inside the network but not at the NI.

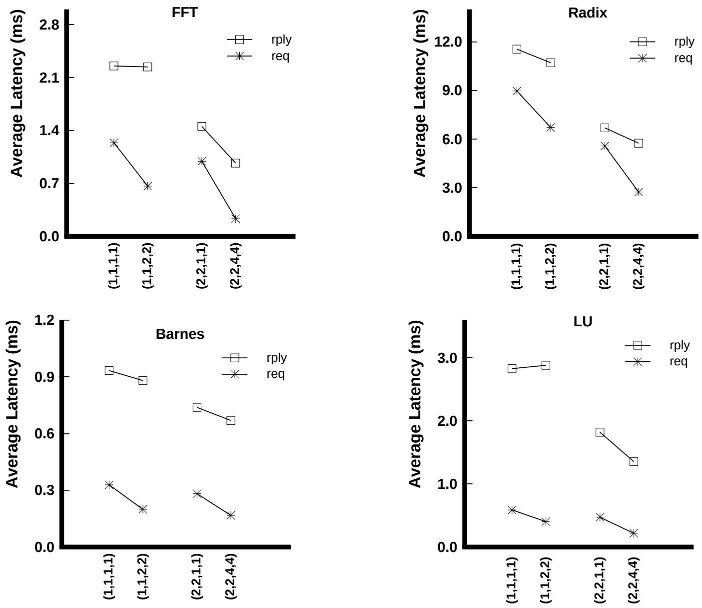

Figure 7: The effect of virtual injection and virtual consumption channels on the average message latencies in the request and reply networks.

Based on the above results, in the following subsections, we focus on network configurations with number of virtual injection/consumption channels at the NI being equal to the total number of virtual channels in the network.

5.2.2 Effect of Unbalanced Virtual Channel Partitioning

Next, we studied the effect of virtual channel partitioning between the request and reply networks. We performed evaluations on network configurations with no more than five virtual channels. There were nine different network configurations: (2,1,3,3) and (1,2,3,3) with 3 virtual channels; (3,1,4,4), (2,2,4,4), and (1,3,4,4) with 4 virtual channels; and (4,1,5,5), (3,2,5,5), (2,3,5,5), and (1,4,5,5) with 5 virtual channels. These configurations were compared with the (1,1,2,2) base configuration. The results of the execution times are shown in Fig. 8.

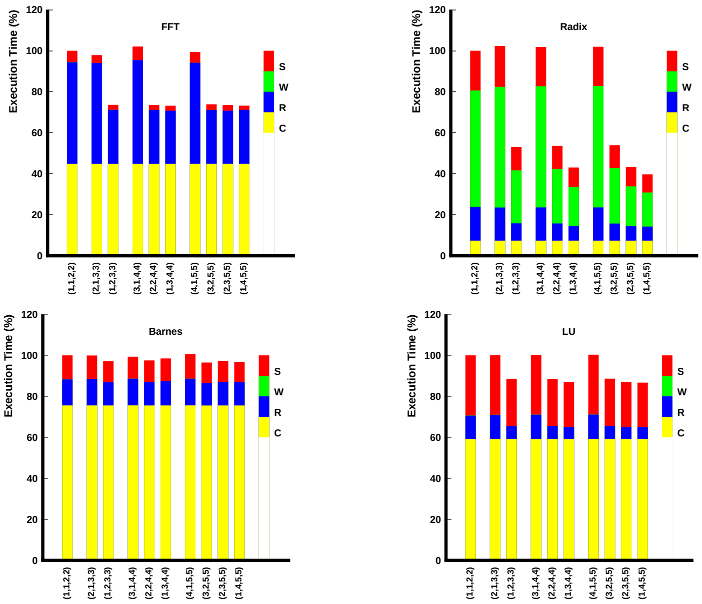

Figure 8: The effect of virtual channel partitioning on the overall execution time.

As expected, for all the applications, the overall execution times were reduced for those configurations which used two or more virtual lanes in the reply network. Specifically, configurations with more virtual channels in the reply network, like (1,2,3,3), (1,3,4,4), and (1,4,5,5), provided the best or close to the best performance in their respective group. The maximum reduction in the execution time of each application ranged from 3.2% in Barnes to 60.3% in Radix. The main reason behind the small performance improvement in Barnes is that the CPU computation time is very dominant (close to 80% in our base configuration) in the overall execution time. Similarly, the high performance improvement in Radix can be contributed to the dominant memory access waiting time (about 70% in the base configuration). Therefore, the maximum performance improvements of 26.8% and 13.4% in FFT (45% computation time) and LU (60% computation time) respectively are more representative.

The results also show that configuration (1,2,3,3) is an interesting design alternative from cost-performance point of view. It gains close to the maximum performance improvement by using just one more virtual channel in the reply network than the minimum required in DSM systems.

Figure 9 shows the average message latencies in the request and reply networks from the same set of experiments. It can be observed that the average message latency in the request network decreased initially and flattened out (or increased slightly) as more virtual channels were moved from the request network to the reply network within each group. On the other hand, the average message latency in the reply network kept decreasing towards a certain value as it obtained more virtual channels.

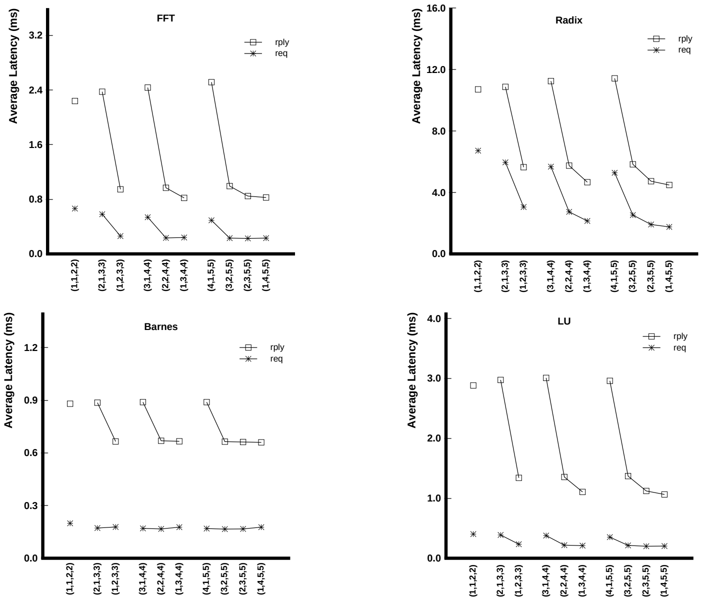

Figure 9: The effect of virtual channel partitioning on the average message latencies in the request and reply networks.

Another observation is that the knee points of all the curves appear at the positions when there are two virtual lanes in the reply network. It correlates to our finding that configuration (1,2,3,3) is the most promising alternative from a cost-performance perspective.

In the next three subsections, we examine how and to what extent the performance benefit of multiple virtual channels change when varying (one at a time) the values of three critical system parameters.

5.2.3 Impact of Cache Line Size

First, we studied the impact of cache line size on the benefit of virtual channels. Modern DSM systems tend to use large cache line size (typical 128 or 256 Bytes) to save the cost of main memory directories and cache tags and exploit the spatial locality of memory references. We selected four configurations (1,1,2,2), (1,2,3,3), (1,3,4,4), and (1,4,5,5) and performed simulations by varying cache line size from 64 to 128 and to 256 bytes. Figures 10 and 11 show the results of the application execution times and the average message latencies. All the execution times are normalized with respect to the configuration (1,1,2,2) with a line size of 128 bytes (the left most bar in the middle group) for each application. As expected, more virtual channels in the reply network are found to be beneficial. This trend becomes stronger as the cache line size grows.

Specifically, maximum performance improvement increases from 23.4% to 32.7% in FFT, from 51.7% to 62.3% in Radix, from 1.6% to 9.4% in Barnes, from 13.3% (with a line of 128 bytes) to 18.6% in LU. According to a study done in [6], the execution time of a DSM application shows a bathtub behavior as the cache line size changes with a pollution point around 64 bytes. Thus, it is reasonable to expect similar or larger performance gain when some other cache line size is used in a real system.

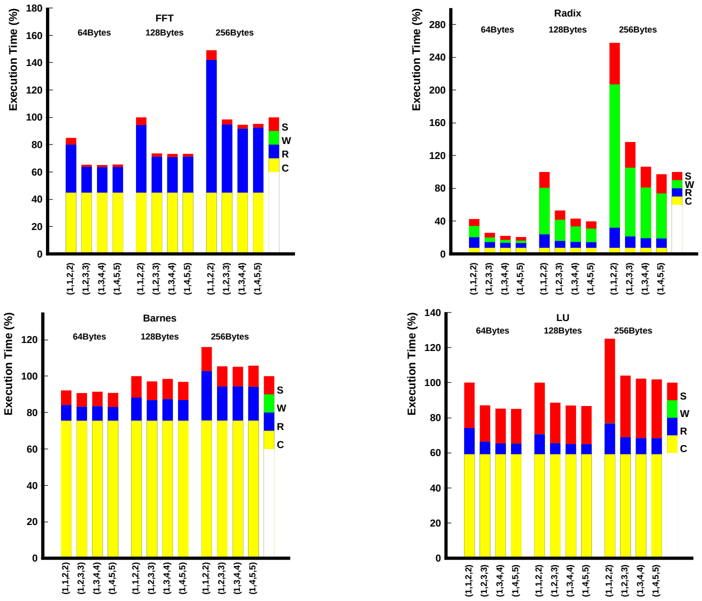

Figure 10: Changing trend of the effect of virtual channels on the overall execution times under different cache line sizes.

It can again be observed that the configuration (1,2,3,3) remains attractive for a wide range of cache line sizes. The difference between the performance of this configuration and that of the optimal configuration using more number of virtual channels grows slowly as the cache line size increases. This is shown quite clearly from the changes in the sharpness with which the curves turn at the knee points in Fig. 11.

5.2.4 Impact of Routing Delay

Next, we studied the impact of routing decision delay on the benefit of virtual channels. Some researchers believe that the network speed can not remain unchanged as more number of virtual channels are supported. In an earlier study [2], a performance model of 30% slowdown in network cycle time for adding each virtual channel has been proposed. However, a more careful analysis on the problem reveals that the slowdown is mainly caused by the routing delay of the header flit at a router. As the network technology advances and better pipelining techniques are developed, several commercial routers [17, 24] supporting a moderate number of virtual channels have been designed successfully without noticeably increasing the network cycle time and routing delay. However, for an in-depth investigation, we selected two configurations (1,3,4,4) and (1,4,5,5) and performed simulations by varying routing delay from 3 network cycles to 4

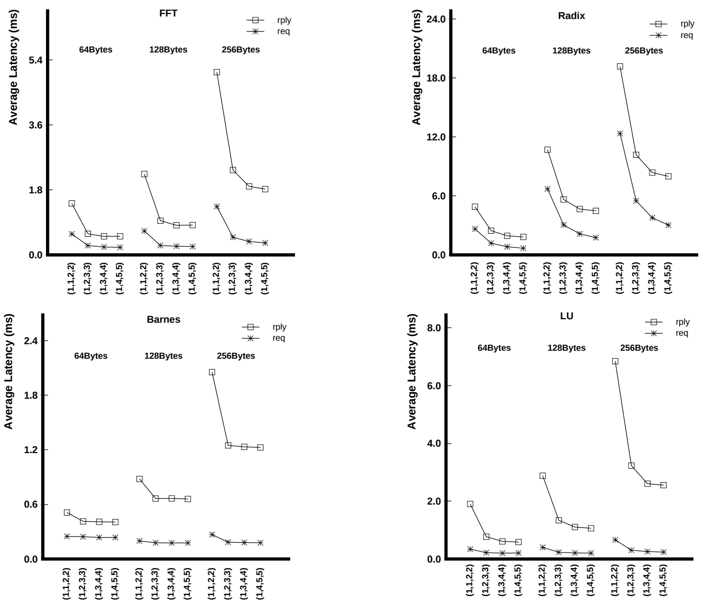

Figure 11: Changing trend of the effect of virtual channels on the average message latencies under different cache line sizes.

network cycles (33% slowdown) and using the same network cycle time. We did not select (1,2,3,3) configuration because there are good reasons to believe that no noticeable increase in routing delay is needed to increase the number of virtual channels from 2 to 3. Figures 12 and 13 show the results of the application execution times and the average message latencies, respectively. It can be observed that such an increase in the routing delay causes marginally increase in overall execution time (less than 2.8% in all our experiments). This indicates that the performance of DSM systems is not so sensitive to reasonably increased routing delay. However, the performance is quite sensitive to the overall network cycle time (or link cycle time) increase as reported in [5, 6, 27].

5.2.5 Impact of Topology

Finally, we studied the impact of network topology. We focused on the k-ary n-cube topologies. Recently, there are some renewed interests of using higher dimensional k-ary n-cube topologies in DSM systems because of many different reasons [10, 20]. Thus, we selected three configurations (1,1,2,2), (1,2,3,3), and (1,3,4,4) and performed simulations by varying dimensionality from 2 to 3 to 6 and keeping the total number of processing nodes fixed at 64. Figures 14 and 15 show the results of the application execution times and the average message latencies, respectively. It can be observed that the performance benefit of using virtual channels reduces as the dimensionality increases. Specifically, the performance improvement decreases from 26.8% to 3.5% in FFT, from 57.0% to 32.8% in Radix, from 2.9% to no gain in Barnes, and from 13.1% to 4.2% in LU. This is not surprising because the network bisection bandwidth increases as the dimensionality increases (under constant link width). The actual physical bisection bandwidth increases from 6.4GB/s to 12.8GB/s to 25.6GB/s. Thus, less contention occurs in the network and virtual

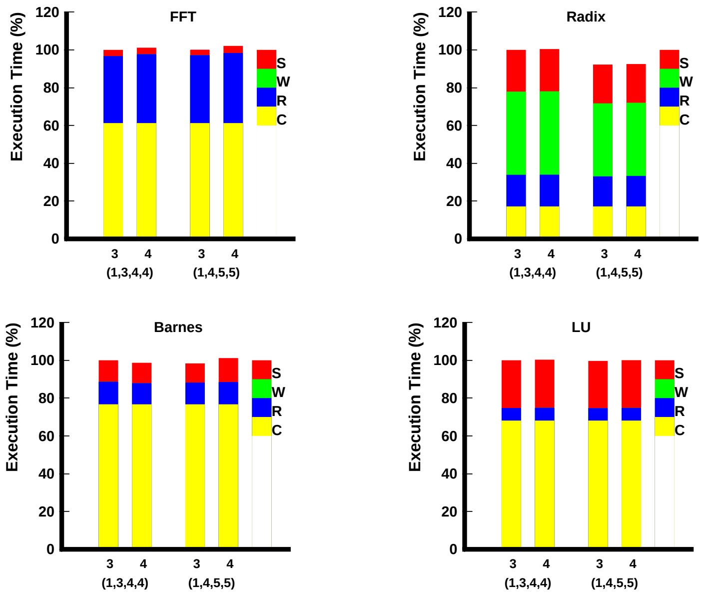

Figure 12: Impact of switch routing delay on the overall execution times.

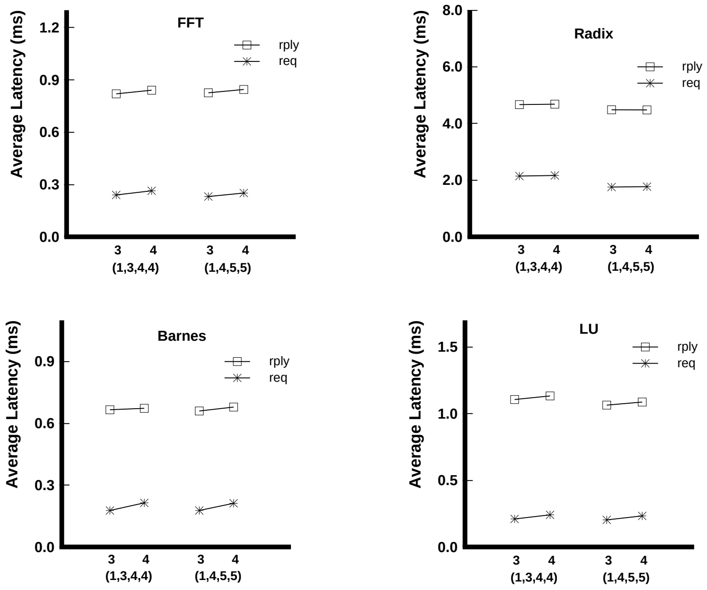

Figure 13: Impact of switch routing delay on the average message latencies.

channels become less useful. This indicates that the benefit of contention reduction obtained by using more virtual channels can also be obtained by using more physical channels at a higher cost. This is confirmed in Fig. 15 which shows that the reductions in average message latencies decrease significantly from 2D to 3D to 6D in all the applications.

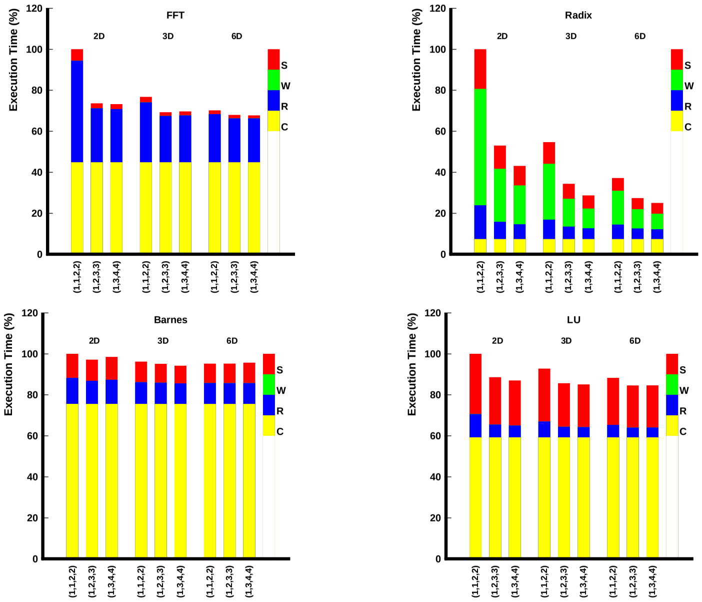

Figure 14: Impact of network topology on the overall execution times.

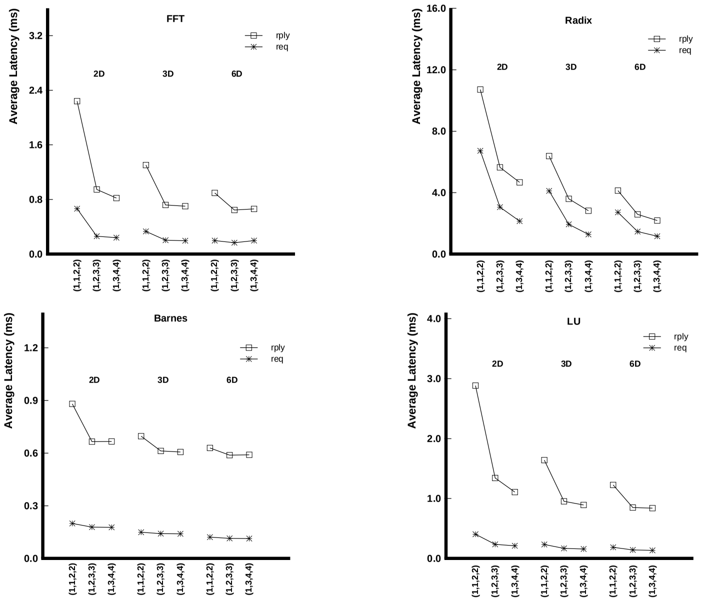

Figure 15: Impact of network topology on the average message latencies.

6 Related Work

Using multiple virtual channels with various buffer sizes in DSM systems was first studied in [19]. However, with more realistic assumptions on existing technology, study [27] has suggested that it may be unjustified to use virtual channel mechanism in DSM systems with respect to cost-performance viewpoint. But, as we have shown in this study, incorporating virtual channel mechanism into DSM systems in an efficient manner can provide a significant performance improvement. The performance impact of network contention under various designs of cache and memory module, processor speed, and different network components has been reported in [6]. A set of guidelines have been proposed in [5] for designing better networks for DSM systems with two virtual networks (with one virtual channel each) and one injection and consumption channel at the network interface. Impact of multiple physical injection/consumption channels has been the focus of study [3]. However, this study has been done with respect to uniform traffic distribution in distributed memory systems. Traffic non-uniformities in the forward and reverse multistage networks in the Cedar system have been investigated in [26]. Strategies for improving the performance of dance-hall vector multiprocessors using MIN networks have been proposed in this study.

7 Conclusion

In this paper we have presented two simple techniques for exploiting the promising performance benefit of the virtual channel mechanism in distributed shared memory systems. Virtual injection and consumption channels can be implemented at very little additional cost to the existing designs of switches and network interfaces. Unbalanced virtual channel partitioning between the request and reply networks requires some design changes within the routers without any increase in cost. Important issues in designing non-FIFO coherence protocols for effectively using multiple virtual channels in distributed shared memory systems have also been highlighted. The effects of the virtual channel mechanism with both techniques have been evaluated through simulations using a set of representative benchmark applications.

Our results indicate that network configuration consisting of one request virtual lane, two reply virtual lanes, three virtual lanes per injection/consumption channel is the most cost-effective design alternative. Overall, we have shown that with an appropriate cache coherence protocol, the virtual channel mechanism can reduce the execution time of a DSM application significantly. In addition, we have shown that the performance benefit of using virtual channels increases with a larger cache line size, is not so sensitive to changes in routing delay, and decreases in a higher dimensional k-ary n-cube topology under constant physical link bandwidth constraint. This study demonstrates that the virtual channel mechanism can significantly improve the performance of a distributed shared memory system. It also provides valuable understandings on the tradeoffs for designing efficient DSM systems using virtual channels.

The techniques proposed in this paper are applicable to any distributed shared memory system using the virtual channel mechanism. It is not limited to k-ary n-cube topologies. This study has focused on the adaptivity between virtual lanes using dimensional order routing. We are currently studying the effective means of using other adaptive routing algorithms, block multiplexing among virtual channels, and priority messages in DSM systems.

Additional Information: A number of related papers and technical reports are available electronically through the home page of *Parallel Architecture and Communication* (PAC) research group. The URL is `http://www.cis.ohio-state.edu/~panda/pac.html`. Alternatively, the papers can be obtained by anonymous ftp from: `ftp.cis.ohio-state.edu, cd pub/communication`.

References

[1] K. V. Anjan and T. M. Pinkston. An Efficient Fully Adaptive Deadlock Recovery Scheme: DISHA. In *International Symposium on Computer Architecture*, pages 201–210, 1995.

[2] K. Aoyama and A. A. Chien. The Cost of Adaptivity and Virtual Lanes in a Wormhole Router. *Journal of VLSI Design*, 2(4):315–333, 1995.

[3] D. Basak and D. K. Panda. Alleviating Consumption Channel Bottleneck in Wormhole-Routed k-ary n-cube Systems. *IEEE Transactions on Parallel and Distributed Systems*, 1997. To appear.

[4] R. V. Boppana and S. Chalasani. A Comparison of Adaptive Wormhole Routing Algorithms. In *International Symposium on Computer Architecture*, pages 351–360, 1993.

[5] D. Dai and D. K. Panda. How Can We Design Better Networks for DSM Systems? In *Proceedings of the 1997 Parallel Computing, Routing, and Communication Workshop (PCRCW'97)*, Atlanta, GA, June 1997.

[6] D. Dai and D. K. Panda. How Much Does Network Contention Affect Distributed Shared Memory Performance? In *Proceedings of the International Conference on Parallel Processing*, Chicago, IL, Aug 1997.

[7] W. J. Dally. Virtual Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, pages 194–205, Mar 1992.

[8] 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.

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

[10] Jose Duato and M. P. Malumbres. Optimal Topology for Distributed Shared Memory Multiprocessors: Hypercubes Again? In *Proceedings of the Euro-Par96*, page to appear, 1996.

[11] A. Agarwal et al. The MIT Alewife Machine: Architecture and Performance. In *International Symposium on Computer Architecture*, pages 2–13, 1995.

[12] D. Lenoski et al. The Stanford DASH Multiprocessor. *IEEE Computer*, pages 63–79, March 1992.

[13] J. Kuskin et al. The Stanford FLASH Multiprocessor. In *Proceedings of the International Symposium on Computer Architecture*, pages 302–313, 1994.

[14] S. C. Woo et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *International Symposium on Computer Architecture*, pages 24–36, 1995.

[15] S. K. Reinhardt et al. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 48–60, May 1993.

[16] W.-D. Webber et al. The mercury interconnect architecture: A cost-effective infrastructure for high-performance servers. In *Proceedings of International Symposium on Computer Architecture (ISCA-24)*, pages 98–107, 1997.

[17] Mike Galles. Spider: A High-Speed Network Interconnect. *IEEE Micro*, pages 34–39, January/February 1997.

[18] J. L. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[19] Akhilesh Kumar and Laxmi N. Bhuyan. Evaluating Virtual Channels for Cache-Coherent Shared-Memory Multiprocessors. In *Proceedings of the 10th ACM International Conference on Supercomputing*, May 1996.

[20] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA highly Scalable Server. In *Proceedings of the 24th IEEE/ACM Annual International Symposium on Computer Architecture (ISCA-24)*, pages 241–251, June 1997.

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

[22] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of the Supercomputing*, 1995.

[23] D. K. Panda, D. Basak, D. Dai, R. Kesavan, R. Sivaram, M. Banikazemi, and V. Moorthy. Simulation of Modern Parallel Systems: A CSIM-based approach. In *Proceedings of the 1997 Winter Simulation Conference (WSC'97)*, December 1997.

[24] S. L. Scott and G. M. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Proceedings of the Symposium on High Performance Interconnects (Hot Interconnects 4)*, pages 147–156, August 1996.

[25] J. P. Singh, W. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. *Computer Architecture News*, 20(1):5–44, 1992.

[26] J. Torrellas and Z. Zhang. The Performance of the Cedar Multistage Switching Network. *IEEE Transaction on Parallel and Distributed Systems*, pages 321–336, April 1997.

[27] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das. Performance benefits of virtual channels and adaptive routing: An application-driven study. In *Proceedings of ACM International Conference on Supercomputing (ISC'97)*, July 1997.