Exploiting the Benefits of Multiple-Path Network in DSM Systems: Architectural Alternatives and Performance Evaluation

Donglai Dai and Dhabaleswar K. Panda

Technical Report
OSU-CISRC-8/98-TR33

Auguest 1998

Exploiting the Benefits of Multiple-Path Network in DSM Systems: Architectural Alternatives and Performance Evaluation

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

Abstract:

Modern high performance networks being proposed for scalable distributed shared memory (DSM) systems support multiple paths (virtual or physical) to increase bandwidth and/or reduce contention. Such networks violate the constraint of pairwise in-order message delivery (FIFO channels) implicitly required by many existing directory-based cache coherence protocols. To solve this problem, two alternative strategies are currently used by computer architects. The first strategy, used in the design of SGI Origin series, is to employ an intelligent cache coherence protocol which directly detects and resolves all coherence race conditions caused by out-of-order (OoO) events in the networks. The second strategy, proposed in the design of HAL S1 series, is to use a sophisticated network interface (NI) which detects and remedies every occurrence of out-of-order (OoO) events between each pair before the messages are fed to the cache coherence controllers. Both strategies involve complicated hardware logic, either at the cache coherence controller level or at the NI level.

In this paper, we focus on the interplay between network interface support and cache coherence protocol in scalable DSM systems with multiple-path networks. We propose a new strategy that use *block correlated FIFO channels*. This strategy detects all potential coherence-sensitive (pairwise) race conditions and prevents them from occurring in the networks. This new strategy allows the use of a simple cache coherence protocol and an inexpensive network interface support. Simulation experiments are performed using benchmark applications to evaluate the performance of our new strategy in comparison with the existing two strategies. Our results indicate that DSM systems using our new strategy always provide either the best or very close to the best overall performance. This study provides valuable insights into the design tradeoffs in incorporating modern networks into DSM systems.

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

Contents

1 Introduction

Scalable shared memory multiprocessor machines, commonly known as distributed shared memory (DSM) systems, are emerging as the trend in building parallel systems because they provide much desired programmability [14]. Such systems make the conversion from sequential programs to their parallel equivalents practical for a large variety of applications. The most popular class of scalable shared memory multiprocessors employs the cache coherent nonuniform memory access (CC-NUMA) architecture. CC-NUMA systems incorporate private caches in each processing node, automatically replicate remote data when needed, and mostly use a flat directory based write-invalidate cache coherence protocol to ensure that accesses to shared data by any processor always get the latest copy. Examples of such systems include research prototypes (Stanford DASH [18], MIT Alewife [1], and Stanford FLASH [15]), and commercial systems (Convex Exemplar [3], Sequent NUMA-Q [19], SGI Origin [16], and HAL S1 [31]).

The inherent processor-to-memory communication of CC-NUMA systems results in frequent exchange of short messages. Communication latency is a pronounced component of the penalty of a remote memory access miss. Therefore, most CC-NUMA systems use (specially designed) high performance switches/routers for constructing the underlying interconnects. For example, SGI Origin uses SGI SPIDER routers [10] with wormhole-switching and HAL S1 uses HAL Mercury switches [31, 20] with packet-switching. Even though the detailed designs differ from one another, these high performance switches/routers all rely on the same set of mechanisms such as cut-through switching, multiple virtual channels [7], and efficient internal buffering [29] to increase network throughput and reduce latency.

Multiple paths exist between a pair of nodes in almost all modern high-performance networks [8]. These paths can be either implemented physically [2] or supported logically via virtual channels [7]. They are used to alleviate congestion, increase throughput, and reduce average message latency. Multiple paths are supported to different degrees by the SGI SPIDER router, the HAL Mercury switch, and the IBM SP switch [2]. One side effect of using multiple paths between a pair of nodes is that messages from a source may arrive at a destination in different orders. This is known as the pairwise out-of-order (OoO) message arrival problem. Although exceptions exist [17, 16], many directory-based cache coherence protocols [27, 30, 4] designed for DSM systems require pairwise in-order arrival to make the design and verification easier [28].

To exploit the advantages of a multiple-path network in a DSM system, architects currently use one of two alternative strategies. The first strategy, represented by SGI Origin, is to enhance the cache coherence protocol with more intelligence so that it can detect and resolve all critical out-of-order (OoO) message arrivals. The main concern of this strategy is the high complexity in the design, verification, and implementation of the resulting cache coherence protocol [31]. The second strategy, represented by HAL S1, is to enhance the network interface with reordering capability to ensure that all pairwise messages seen by the cache coherence protocol are in-order (total FIFO channel). The drawback of this strategy is a noticeable increase in both complexity and overhead (delay) at the network interface. The extra overhead may offset some performance advantages of the multiple paths. It is a real challenge to design a DSM system which can obtain most advantages of the multiple-path networks without considerably increasing the complexity of the cache coherence protocol and increasing the complexity/overhead (delay) of the network interface.

In this paper, we take on such a challenge and propose a new strategy for exploiting the benefits of multiple-path networks in a DSM system using *block correlated FIFO channels.* This new strategy detects all potential coherence sensitive (pairwise) race conditions and prevents them from occurring in the network. It allows the use of both an in-order cache coherence protocol and a simple network interface. We compare and contrast the strengths and weaknesses of the two previously known strategies and show how the new strategy combines their advantages and avoids their drawbacks under current-day technology. Thus, the proposed strategy can offer better performance at low cost.

To quantitatively compare the performance of the three strategies using practical system configurations, we performed simulation experiments using benchmark applications. For most applications, our results show that the proposed new strategy is typically less than 1% slower than the SGI Origin approach and up to 40% *faster* than the HAL S1 approach. These results indicate that DSM systems using our new strategy can provide very competitive overall performance.

The main contributions of this paper can be summarized as follows:

- A new strategy is proposed to obtain maximum performance benefit of multiple-path networks while maintaining low complexity in the cache coherence protocol and network interface. A new concept — *block correlated FIFO channels* — is introduced to satisfy the desired order property for a simple cache coherence protocol and allow multiple parallel paths/routes to be used in the network for achieving high throughput and low latency. An algorithm and a simple implementation of this new strategy are designed and presented in detail.

- A thin reordering protocol is designed for the total FIFO channel strategy. An algorithm and an implementation with high performance and reasonable complexity are discussed and presented in detail. Even through this strategy was described in [31], no detailed information on the reordering mechanism was presented.

- The results confirm the intuition that the performance of intelligent cache coherence protocol strategy is superior to that of total FIFO channel strategy. To the best of our knowledge, this is the first study to compare the performance of DSM systems using these two strategies under fairly equal conditions.

- This study sheds new lights on the impact of network and network interface on the overall DSM performance.

The rest of this paper is organized as follows. Section 2 reviews issues relevant to communication in DSM systems. This section also overviews the alternative strategies for incorporating multiple-path networks. Section 3 describes the intelligent cache coherence protocol strategy. It explains the basic idea, functionality of the network interface, and the complexity of the cache coherence protocol being used. Section 4 describes the total FIFO channel strategy. It explains the basic idea and presents the design and implementation of a thin sliding window protocol for message reordering in detail. In Section 5, we provide the motivation for a new strategy, introduce the concept of block correlated FIFO channels, and present an efficient yet low cost implementation of this concept. Section 6 describes our methodology for evaluating the performance of the various strategies discussed in this paper. Simulation results and insightful discussions are presented in Section 7 and Section 8 presents our conclusions.

2 Issues in CC-NUMA System Design

In this section, we briefly review some basic issues relevant to communication in the design of DSM systems. We overview the alternative strategies and issues for incorporating multiple-path networks in DSM systems.

2.1 Cache Coherence Protocol and Network Transaction

A CC-NUMA system provides a global memory address space to all processing nodes. The *home node* of a memory block is a node in whose main memory the block is allocated. When a memory operation is issued to a location and the issuing node (which is not the home node) does not have a valid copy in its private cache, a request message is sent via the network to the home node. Eventually, a response message comes back to the issuing node with a copy of the memory block containing the content of the desired location. The entire communication process of constructing, sending, receiving, and dispatching a message is commonly abstracted as a single *network transaction* [6]. Thus, a simple request-response cycle contains two network transactions. At any time, information is maintained among a set of nodes about the operating states of copies of each memory block. A node in this set either has cached the block or is the home node of the block. A cache coherence protocol is a protocol guaranteeing that a *read* to any memory location always gets a copy of the content from the latest *write* which the reading processor is aware of. In a CC-NUMA system, such a consistent view of memory is ensured on the basis of each memory block. Various cache coherence protocols differ mainly on the types of block states used, the types of network transactions used, and the rules for state transitions.

The complexity of a cache coherence protocol is a very important issue in the design of a DSM system. A complex coherence protocol requires a large amount of logic to be implemented and takes a lot of effort to design, verify, and test the logic. Until now, verification of a highly complex coherence protocol remains a monumental task for computer architects [17]. Although fast high coverage techniques exist [9], various complete coverage verification methods rely on exhausting the reachability of finite state machines or their equivalents. The required amount of computation time and storage increase exponentially as the number of states of cache and directory and the number of types of network transactions increase [22]. Also, complex protocols typically introduce larger overhead (delay) on the critical path leading to longer occupancy on key logic circuits for each state transition in the protocols. This has a detrimental impact on the overall performance.

2.2 Cache-based and Memory-based System

The cache coherence protocol of a CC-NUMA system is strongly influenced by the directory organization of the system. Depending on where the directory information is stored, CC-NUMA systems can be divided into to categories: (a) cache-based systems and (b) memory-based systems [6]. In cache-based systems, the information about current cached copies of a block is scattered along with the copies forming a distributed linked list. The home node of the block contains a pointer to the head of the list. IEEE SCI standard [26] and most earlier commercial systems (e.g.,., Convex Exemplar [3] and Sequent NUMA-Q [19]) fall into this category. Alternatively, in memory-based systems, the information about current cached copies of a block is completely maintained at the home node. Typically, a variation of the presence bit-vector directory organization is used. Most research prototypes (Stanford DASH/FLASH [18, 15], MIT Alewife [1]) and many current-day commercial systems (SGI Origin [16], HAL S1 [31]) fall into this category. The storage overhead of directory used to be a major concern in designing memory-based systems. Nowadays, practical techniques like limited pointers and directory caches have been developed to reduce this overhead considerably. Due to the inherent efficiency allowed by the associated cache coherence protocol [6], memory-based systems are emerging as the trend in building CC-NUMA systems. This paper therefore focuses on the design of this second category of CC-NUMA systems.

2.3 Request-Response Deadlock

Another important issue in the design of a DSM system is the method adopted to deal with the potential request-response deadlock [18, 6]. Such a deadlock occurs when the input buffer of a node is filled up by messages/transactions that can not be progressively processed by the cache coherence protocol. Thus, no new messages/transactions can be received and processed. Two main solutions are commonly used to solve this problem: to provide enough buffering [6] or to provide (logically) separate networks [6].

The first solution is viable because there is an upper bound on the number of outstanding memory operations per node. Depending on the cache coherence protocol being used, the total number of network transactions may vary but is still bounded. Therefore, the amount of input buffer space per node for the maximum number of possible incoming messages/transactions is bounded. The buffer space can be located either at the network interface or in (local) main memory. The shortcoming of this solution is that it is too expensive in practice. DSM systems using this solution normally also employ time-out and retransmission to reduce the amount of buffer space. The MIT Alewife [1] and the HAL S1 [31] use this solution. We assume this solution for two of the three strategies studied in this paper.

The second solution uses separate request-response networks. This creates no deadlock if the protocol is strictly of request-response type [6]. In such a protocol, a request transaction generates a response or no transaction, and a response generates no transaction. A node can start one or more request-response cycles only when it serves its own outstanding memory operations. However, such a protocol normally produces a long chain of network transactions to complete a memory operation. For better performance, practical coherence protocols use various forwarding techniques. These protocols detect potential deadlock situations and avoid the deadlocks by resorting to NACKs or reverting to a strict request-response protocol [6] temporarily. Such a solution is used in Stanford DASH [18], FLASH [15], and SGI Origin [16]. This solution is assumed in one of the three strategies discussed in this paper.

2.4 Multiple-Path Network and Out-of-Order(OoO) arrivals

As mentioned earlier, modern high-performance networks support multiple virtual channels [10, 20] or multiple physical routes [2] between a pair of nodes to increase bandwidth and/or reduce congestion. When virtual channels are supported in a network, a path can be viewed as a legitimate sequence of virtual channels from the source to the destination node. The latency of a message varies depending on the length of the message, length of the used path, and the congestion along the hops of the used path. Therefore, there is a possibility that a message ($m2$) using one path reaches its destination sooner than a previously injected message ($m1$) using another path between the same source-destination pair, as shown in Fig. 1. Such a bypassing scenario is commonly known as an *out-of-order (OoO) arrival* of messages. A message may bypass more than one previously injected messages. One *out-of-order (OoO) event* is generated by an out-of-order (OoO) arrival between the bypassing message and each of the bypassed messages. The concepts of out-of-order (OoO) arrival and out-of-order (OoO) event are very important for designing cache coherence protocols. As we will see later, they are treated differently in each of the strategies discussed in this paper.

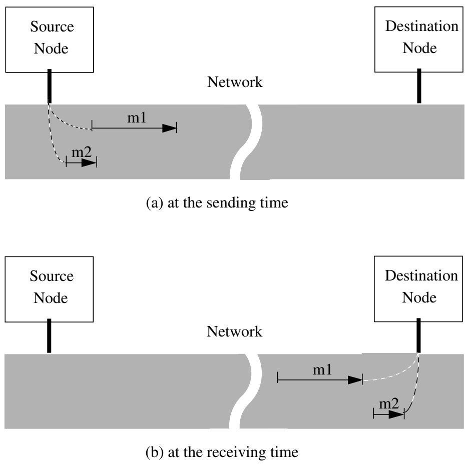

Figure 1: An example of out-of-order (OoO) arrival of message between a source-destination pair.

2.5 A Generic Architecture

A generic CC-NUMA architecture is shown in Fig. 2. The entire system consists of a number of processing nodes connected together using a scalable network. Each node has a processor, its private cache, a portion of global shared memory, and a sophisticated node controller. The node controller is the heart of a CC-NUMA system. Its main components include a cache coherence controller and separate interfaces for the processor module, memory/directory module, and network. It also contains logic for deadlock prevention and arbitration between the interfaces. The cache coherence controller observes all cache misses, synchronization events and uncached operations. It keeps track of the stable or transient information of ongoing memory operations, invalidates or retrieves cache blocks, completes incoming network transactions, and initiates outgoing network transactions. The network interface is mainly responsible for constructing, sending, receiving, and dispatching the actual messages. It also does all necessary work to provide the desired communication abstraction from the underlying network to the remaining part of a node.

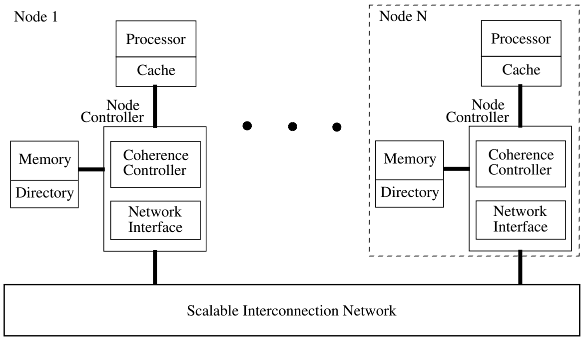

Figure 2: A generic CC-NUMA architecture.

2.6 Alternative Strategies for Incorporating Multiple-Path Networks

The main characteristics of a CC-NUMA system are primarily dependent on the strategy used to design its node controller, especially its cache coherence controller and/or network interface. Therefore, this paper focuses on the basic differences in the node controller between alternative strategies for obtaining the benefits of multiple paths in networks. Three different strategies are considered: the intelligent cache coherence protocol (I-CC) strategy, the total FIFO channel (T-FIFO) strategy, and the block correlated FIFO channel (C-FIFO) strategy. The I-CC strategy uses an enhanced cache coherence protocol with more intelligence to detect and resolve the critical out-of-order messages. The T-FIFO strategy uses an enhanced network interface to ensure that all pairwise messages seen by the cache coherence controller still possess the in-order property. The C-FIFO strategy partitions messages into groups such that no coherence sensitive race conditions exist across groups and race conditions within each group are prevented from occurring via the network. Table 1 summarizes how the complexity of the node controller (cache coherence controller and network interface) varies across the three strategies. More details about each of these strategies are presented in the next three sections.

Table 1: Overall comparison of the complexity of node controller under alternative strategies for incorporating multiple-path networks.

	Complexity	
Strategy	Coherence Controller	Network Interface
I-CC	High	Low
T-FIFO	Low	High
C-FIFO	Low	Low

3 Intelligent Cache Coherence Protocol (I-CC) Strategy

In this section, we overview the I-CC strategy [16] for incorporating multiple-path networks. We explain the basic idea and overview the network interface being used under this strategy. Finally, we focus on the critical factors in the design of its cache coherence protocol.

3.1 The Basic Idea

The philosophy behind the I-CC strategy is to build enough intelligence into the cache coherence controller so that it can detect and revolve all the race conditions caused by the actual occurrences of OoO events. The strategy uses extremely simple network interfaces. Typically, the overall network of the system is logically separated into request and response networks. Complete freedom is granted in routing a message from a source to its destination within each logical network. Figure 3 illustrates the network connecting arbitrary processing nodes in such a system. Four parallel paths between the nodes are shown where the request and response (logical) networks have two paths each.

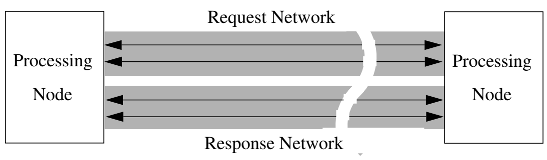

Figure 3: The request or response logical networks between a source-destination pair in a system using the I-CC strategy.

3.2 The Network Interface

As mentioned earlier, the network interface used in systems employing the I-CC strategy is quite simple. A description of the network interface functionality in a pseudo-code format is listed in Fig. 4. Constructing and dispatching functions mostly involve simple format changes between intra-nodal data and inter-nodal one. The selection function decides whether the request or the response network should be used in transferring a message based on its type. A simple network interface like this can be implemented very efficiently by using the pipelining technique. As it will be shown later, such a low overhead network interface can be very beneficial to the overall system performance.

Sending:
construct(msg)
net=select(msg)
send(msg, net)

Receiving:
receive(msg, net)
error-check(msg)
dispatch(msg)

Figure 4: Pseudo-code description of the network interface functionality in a CC-NUMA system employing the intelligent cache coherence protocol strategy.

3.3 The Cache Coherence Protocol

In this section, we discuss the most important and relevant factors in the design of an intelligent cache coherence protocol for the I-CC strategy. For a more detailed description on such cache coherence protocols, readers can refer to [16, 18, 17, 15].

3.3.1 Required Intelligence

In order to appreciate the intelligence of a cache coherence protocol required by the I-CC strategy, let us first examine an example of race condition as shown in Fig. 5. 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 current owner is node N_3. Once the request Rxq arrives at node N_2, a flush request ($Fsh(2)$) is issued by N_2 to 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 an in-order arrival 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 allows arbitrary message arrival order, message Frp can arrive at N_2 either before or after Wbk. This uncertainty becomes a race condition that must be detected and resolved properly.

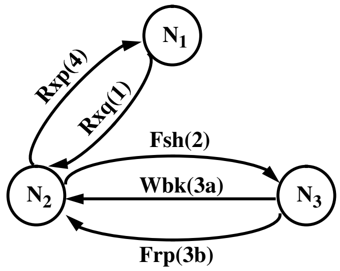

Figure 5: An example of race conditions faced by cache coherence protocols when out-of-order (OoO) arrival of messages is allowed in the underlying network.

There are many other types of race conditions exist in a CC-NUMA system using the I-CC strategy. The coherence protocol must first detect all of them dynamically. The popular approach is to remember the relevant history of operations on each memory block using different directory/cache states. By checking the current state and the type of message, all race conditions can be detected successfully. Once a race condition is detected, actions like buffering, issuing a NACK, reverting to a simpler protocol, or combining transactions can be taken to resolve it depending on different performance optimization goals.

It should be noted that race conditions are detected based on each memory block individually. When OoO events between messages relating to different memory blocks occur, they are not detected by the above coherence protocols. Fortunately, the OoO events across memory blocks do not raise any coherence race condition by virtue of synchronization mechanisms used in the CC-NUMA systems. This aspect together with the synchronization mechanisms are discussed in more detail in Section 5.

3.3.2 Complexity

It is difficult to estimate the precise complexity of the I-CC coherence protocols because of the wide variation in the functions and optimizations supported by them. In order to get a rough idea of the range of complexity of these intelligent cache coherence protocols, let us take the SGI Origin [16] coherence protocol as an example. In this protocol [6], overall, the number of types of network transaction is 15 requests (including invalidations and interventions) and 39 responses. The number[1] of directory states is 7. The number of cache states is no less than 7. It should be noted that many optimizations targeted towards better performance have been incorporated into the SGI Origin coherence protocol. Without these optimizations, the above numbers may vary slightly. Nevertheless, it is clear that the intelligence required by the I-CC strategy introduces significant extra complexity into the coherence protocol.

Overall, CC-NUMA systems using the I-CC strategy are expected to deliver high performance because the benefits of the multiple-path network are exploited intensively in the normal transactions (the most common cases) and transferred to applications. Little or no progress is made in the coherence protocol process only in the undesirable scenario where race conditions (the infrequent cases) have actually occurred. The main drawback of the I-CC strategy, however, is the high complexity of the coherence protocol. As we will see later, the other two strategies discussed in this paper allow the use of a much simpler coherence protocol.

4 Total FIFO Channel (T-FIFO) Strategy

In this section, we study the critical details of a total (pairwise) FIFO channel strategy for incorporating multiple-path networks. We first describe the basic idea and the cache coherence protocol used in this strategy. We then focus on an efficient design for the network interface.

4.1 Basic Idea and Cache Coherence Protocol

The philosophy of the T-FIFO strategy [31] is to build powerful network interfaces to screen all messages and shield the effect of any out-of-order (OoO) arrival from the rest of the node. This strategy uses a simple cache coherence protocol assuming an in-order arrival network. Typically, sufficient input buffer space is provided at each node to break deadlocks as discussed in Section 2.3. The network which transfers both request and response messages in an undistinguished manner has the maximum freedom in routing a message from a source to its destination. The enhanced network interface with its reordering capability can detect all pairwise OoO arrivals and remedy them. It ensures that all messages seen by the cache coherence protocol controller satisfy the pairwise in-order arrival property regardless of the actual order in which these messages arrive. Figure 6 illustrates an example network connecting arbitrary processing nodes in such a system. Four parallel paths between the nodes in the uniform network are shown. Compared to Fig. 3, it is to be noted that no distinction is made between request and response networks.

The cache coherence protocol used in this strategy can assume the in-order arrival property

[1] Including one state for efficient page migration.

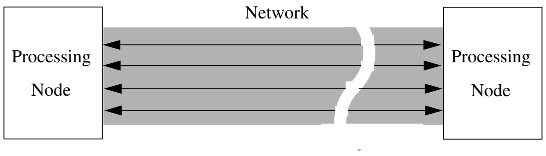

Figure 6: The uniform network between a source-destination pair in a system using the T-FIFO strategy.

between pairwise messages safely. This in-order arrival property effectively prevents almost all difficult race conditions from occurring. Most existing cache coherence protocols can be used by this strategy [27, 30, 4, 5, 23, 22]. Since these coherence protocols are well understood, no further discussions are provided.

4.2 The Network Interface

CC-NUMA systems using the T-FIFO strategy demand two basic functions from the network interface: (a) to detect the occurrences of pairwise out-of-order (OoO) arrivals and (b) to remedy every such occurrence into a set of in-order arrivals for the remaining part of the node. Various techniques exist to accomplish both requirements. A simple sliding window protocol coupled with a reordering buffer is the typical choice of a designer. A description of the enhanced network interface in a pseudo-code format is listed in Fig. 7. Compared to Fig. 4, this new network interface becomes more complex, especially at the receiving side. The main enhancement consists of two parts: (a) manipulation of pairwise sequence numbers and (b) manipulation of the messages in the reordering buffer. In the following subsections, we study the implementation of each of these two parts with an emphasis towards their impact on performance.

```
Sending:
  gen-seqno(msg) /* r */
  construct(msg)
  send(msg)

Receiving:
  receive(msg)
  error-check(msg)
  /* seqno-check returns two values */
  (inorder,src)=seqno-check(msg) /* r */
  if (inorder==FALSE)
    reorder-defer(msg) /* 2r */
  else {
    dispatch(msg)
    reorder-dispatch(src) /* f(r) */
  }
```

Figure 7: Pseudo-code description of the network interface in a CC-NUMA system employing the total FIFO channel strategy. The comments show the time delay model of relevant operations. This delay model is used in Sections 6 and 7 for performance evaluation.

4.2.1 Generating and Checking Pairwise Sequence Number

The sliding window protocol uses a bounded sequence number associated with each message transmitted in the network. Each pair of source and destination nodes in the network uses an independent series of sequence numbers. A message gets its sequence number at the source node and carries this number as part of its header to the destination. The network interface of each node has two sets of counters keeping the pairwise sequence numbers, one for sending and the other for receiving. Each counter in the two sets corresponds to a potential source or destination. All counters are initialized properly to the same value before any message transmission takes place in the network. A i'th sending sequence number counter is incremented at the time when a message at the node for node i is generated. Correspondingly, the i'th receiving sequence number counter is incremented at the time when a message is delivered to the coherence controller from node i. Any occurrence of out-of-order messages is detected when the value in the receiving sequence number counter of a destination is less than the sequence number contained in the received message.

In a high performance network interface, multiple messages can be generated, injected, received, and delivered at the same time. Thus, overlapped accesses to different sequence number counters must be allowed. On the other hand, the atomicity of a load-modify-store operation on each counter must also be enforced for the above sliding window protocol to work properly. Therefore, interlock signals among simultaneous counter accesses are necessary. For an aggressive design, an access to a counter stalls only when there exists an ongoing previous access to the same counter. Accesses to different counters do not affect one another.

4.2.2 Organization and Manipulation of Reordering Buffer

Once the occurrence of an out-of-order message is detected at the destination, the message can be either dropped, NACKed, or buffered. Both the dropping and NACKing techniques require message retransmission and are suitable only for lightly loaded networks. In the context of high performance DSM systems, the buffering technique is more attractive. The basic idea behind buffering is to defer the delivery of the message to the coherence controller until all its predecessors have been received and delivered. Such a buffer is called reordering buffer. When a reordering buffer is used, a check must be done to see if the next expected message is already in the buffer each time a message is delivered to the node controller. This checking must be done quickly, otherwise it may slow down the message receiving process significantly.

A naive design is to use separate buffer entries for each potential source so that out-of-order messages between one pair only increases the checking time after a message between that pair is delivered. However, this design requires a large buffer space and is expensive to implement in a scalable DSM system. A better alternative is to use a pairwise flag or counter indicating if out-of-order messages between the pair exist in the buffer. The buffer space can be shared by all the sources. The flag/counter can be kept and accessed together with the receiving sequence number counter of each pair. Checking the reordering buffer can be done in two steps. At the time that a message is delivered to the coherence controller, the flag is checked together with the manipulation on the receiving sequence number counter. If the flag is 'off', the checking of the reordering buffer completes. If the flag is 'on', a traversal of the reordering buffer takes place to potentially retrieve the next expected message between the pair.

The reordering buffer need not be physically separated from the normal message receiving buffer. A linked list can be used to save the time spent on message copying between the two. Entering, exiting, or traversing the reordering buffer must now be made mutually exclusive. Under extremely rare condition that the reordering buffer is full, an out-of-order (OoO) message can either be dropped and retransmitted later or be placed in uncached local main memory (as an extension to the reordering buffer). Since there are no or very few messages in a reordering buffer at any given time, the above design works well. Our experimental data shows that one or two entries in a reordering buffer are sufficient to gain most of the benefits of such a buffer for almost all applications.

With the above reordering buffer, a network transaction using the thin sliding window protocol (Fig. 7) can be described as follows. At the beginning, the network interface of the source obtains a (unique) pairwise sequence number (taking r cycles) and constructs a new message containing the sequence number as a part. The message is then sent out via the network. When the message is received by the network interface at the destination, the receiving network interface compares the sequence number contained with the expected sequence number between the source-destination pair (taking r cycles). If the two numbers are equal, then the message can be dispatched to the remaining part of the destination node. Otherwise, an out-of-order message arrival occurs and the message is deferred by entering the reordering buffer (taking $2r$ cycles). At any time, when a message is dispatched, the corresponding pairwise flag is checked for any potentially deferred message between the pair in the reordering buffer. If the flag is 'on', a search in the reordering buffer takes place. Each time, a comparison on the sequence number takes r cycles. An exit from the reordering buffer takes $2r$ cycles.

From the above description, it is clear that in the T-FIFO strategy, the overhead introduced by the message reordering slows down every network transaction even if no out-of-order (OoO) message arrival actually occurs. However, OoO message arrival arises only occasionally in multiple-path networks using current generation technology. In other words, in-order arrivals are much more common than out-of-order (OoO) arrivals. Therefore, the T-FIFO strategy penalizes the most common cases for the sake of special cases, violating the fundamental principle of "making the common case fast" [13] for designing high-performance computer systems.

5 Block Correlated FIFO Channel (C-FIFO) Strategy

From the discussion in the previous two sections, it is clear that both existing strategies for incorporating multiple-path networks have severe drawbacks. There is a need for a new strategy which has low complexity at both the coherence controller level and the network interface level. In this section, we present such a strategy. We develop this strategy from the basic goal of a CC-NUMA system, describe its key concepts, and presents a simple design at the coherence protocol level and network interface level using current-day technology.

5.1 Basic Requirement for Consistency

Implications of Synchronization Mechanism: The ultimate goal of a node controller is to serve a maximum number of outstanding memory operations in the minimum amount of time. To serve an outstanding memory operation, a chain of network transactions need to be completed successfully according to the cache coherence protocol being used. Fortunately, the services for most simultaneous/concurrent outstanding memory operations can proceed in an overlapped manner without causing any consistency problem. This is a direct result of the synchronization mechanism used in current generation DSM systems. Such a mechanism ensures that services for multiple outstanding memory operations from the same processing node can proceed in an overlapped manner. However, outstanding memory operations from different processing nodes may still cause consistency problem, because of the asynchronous nature of the nodes.

A Necessary and Sufficient Condition for Consistency: For convenience of discussion, we define two memory operations to be *(block) correlated* memory operations if their target memory blocks are the same. Similarly, we can define (block) correlated network transactions and (block) correlated messages to be network transactions or messages that relate to the same memory block. A necessary and sufficient condition for consistency is that if two memory operations are block correlated, their services (the chains of network transactions) must be serialized to the same order observed by every involved node (more specifically, every involved node's cache coherence controller). A race condition indicates that there is a possibility that the serialization order between two memory operations or their network transactions is observed differently by the involved nodes. Such a race condition may allow an out-of-order (OoO) event to occur between pairwise correlated messages. If nothing is done about this event, the cache coherence controllers of the source and destination observe different order of the serialization. Such an OoO event is defined as a block correlated OoO event. All critical race conditions are associated with correlated OoO events.

Race Conditions in the I-CC and T-FIFO Strategies: In the I-CC strategy, race conditions are allowed in the coherence protocol. The I-CC strategy depends on the coherence protocol to detect block correlated OoO events and resolve them. However, in the T-FIFO strategy, the network interface has no knowledge about the operation history of a memory block. The only knowledge the network interface has about network transactions is pairwise order. Therefore, it assumes that any two network transactions are potentially block correlated and takes conservative actions to force the in-order arrival property among all pairwise network transactions in order to prevent any possible block correlated OoO events from occurring.

Insight for Improvement: The necessary and sufficient condition for consistency can be translated into the following two consistency rules: (1) all correlated network transactions need to maintain the in-order property; and (2) non-correlated network transactions can proceed in any overlapped manner. The I-CC strategy uses both rules, but to enforce rule 1 using the coherence protocol, making the protocol highly complex. On the other hand, the T-FIFO strategy does not exploit the potential benefits of rule 2 in order to use a significantly simpler coherence protocol than the I-CC strategy. We now propose a new strategy which enforces both rules while using the same coherence protocol as the T-FIFO strategy.

5.2 Parallel FIFO Paths

A FIFO path is a path along which no message is allowed to bypass another message. Most current generation networks support FIFO paths (physical or logical). For example, the IBM SP [2] network has multiple physical routes between every pair of nodes, each of which is a FIFO path. On the other hand, in networks based on switches like HAL Mercury [31], virtual channels are used to provide parallel FIFO paths between nodes. Finally, in networks using the SGI SPIDER switches, the effect of FIFO channels can be obtained by simply setting one bit 'off' (which is the normal case in SGI Origin [16]) in the header of each message. Figure 8 illustrates a network connecting arbitrary processing nodes using parallel FIFO paths. Four parallel paths are shown between the nodes.

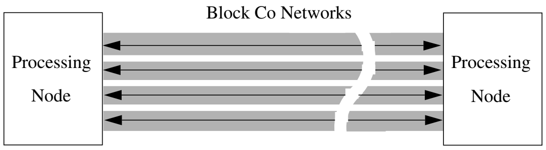

Figure 8: The separate logical networks between a source-destination pair in a system using the C-FIFO strategy.

A *block correlated FIFO channel (BCFC)* is an abstraction of a FIFO channel used for transferring block correlated messages between a source-destination pair. Multiple BCFC may be mapped to a single FIFO path. Every set of block correlated messages between a source-destination pair has a distinct BCFC. Such a BCFC puts no restrictions on the order of network messages which are not block correlated between the source-destination pair because they are in separate channels. Therefore, block correlated FIFO channels are a perfect match for enforcing the two rules for consistency previously discussed in subsection 5.1.

5.3 Simple Network Interface

Conceptually, the number of block correlated FIFO channels (BCFC) between a given pair of nodes is equal to the total number of blocks in the shared address space. The complexity of an implementation of BCFC using the sliding window protocol as shown in Fig. 6 will be too high because of the explosive number of counters required. However, by using the parallel correlated FIFO channels described in previous subsection, the implementation of BCFC becomes a simple mapping problem from the set of BCFC to the set of parallel FIFO path. The criteria for a good implementation include the following two: (1) the mapping should incur low overhead every time it is used and (2) the mapping should require low hardware complexity. A good mapping function is the remainder (modulo) arithmetic function. Since the number of parallel FIFO paths between a source-destination pair is normally quite small in a practical system, a hard-wired remainder operation on a few least significant bits of the block address contained in the message header can be efficiently done. If the number of parallel FIFO paths between a pair is a power of two, those least significant bits can be used directly to select a virtual path. Therefore, the resulting network interface for this C-FIFO strategy is very similar to one used by the I-CC strategy as shown in Fig. 3. The above mapping function is mostly as simple and efficient as the selection function.

It is now clear that this new C-FIFO strategy uses a cache coherence protocol as simple as the one used in the T-FIFO strategy and a network interface as simple and efficient as the one used in the I-CC strategy. From the qualitative discussions in this and previous two sections, we can expect that the C-FIFO strategy can provide us with designs of more cost-performance DSM systems than the two existing strategies.

6 Evaluation Methodology and System Parameters

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

6.1 Node Organization

The hardware cache coherent multiprocessor we simulated had a generic DSM architecture similar to the FLASH machine [15]. Each node contained a processor module, an interleaved memory and its associated directory module, and an integrated node controller or hub. The processor module contained one processor and two-level caches. The processor was modeled as a single-issue microprocessor. The instruction latencies and issue rules were modeled based on the DLX design [13]. The processor was supported by a 8 Kbytes direct-mapped write-through L1 data cache with a line size of 32 bytes and a perfect instruction cache. The operating rate was assumed to be 300 MHz. A coalescing write buffer with 8 entries, each of the size of L1 cache lines, was provided between L1 and L2 caches to reduce processor stalls caused by writes. A read miss in both the L1 cache and the L1 write buffer stalled the processor until the desired data was returned from the lower levels of the memory hierarchy. The processor and L1 cache connected to a 128 Kbytes 2-way set associative write-back and write-allocate L2 cache with a line size of 256 bytes via a 8-byte width data-path. The interleaved memory and directory module was assumed to support multiple read/write ports. On a memory access, the critical word of the memory block was available in 20 processor cycles (66 ns). The successive words in the block followed in a pipelined fashion. A directory cache was assumed to eliminate the stalls in the node controller caused by directory accesses.

In the node controller, the connections to the processor module and memory/directory module were assumed to be 16 bytes wide and were assumed to operate at 100 MHz. A coalescing transaction buffer of 8 entries was provided for resolving outstanding local or remote memory operations. Each entry was able to hold one L2 cache line and merge multiple L1 cache writes targeting the same L2 line. The machine supported the release consistency memory model [15, 16]. The synchronization protocol assumed in the system was the queue-based spin-on-local-copy protocol, similar to the one used in DASH [18]. The coherence controller and network interface within the node controller is described in detail in subsection 6.3.

6.2 Network Organization

The system consisted of 64 processing nodes. Two processing nodes were separately connected to one network switch (somewhat like SGI Origin [16] [2]) to avoid the scenario in which the intra-cluster connection becomes a bandwidth bottleneck. The network switches were interconnected in a topology of k-ary n-cube (default was a 5D hypercube). The network switches used the dimensional order wormhole routing scheme and supported multiple virtual channels with a flit size of 64 bits (8 bytes). Each virtual channel had an input buffer of 32 flits (256 bytes) and an output buffer of 4 flits (32 bytes). Adaptivity between different virtual channels existed but could be turned off. Internally, each switch used the DAMQ input buffering scheme to eliminate the head-of-line (HOL) blocking problem [29, 10]. The switching pipeline was modeled based on SGI SPIDER switch [10], i.e., 2 switch cycles for link synchronization, 1 cycle for routing, and 1 cycle for moving a flit across the crossbar. The switch was assumed to operate at 100 MHz. The distance between switches was assumed to be short. Thus the link propagation time was assumed to be as 1 switch cycle (10 ns).

6.3 Coherence Controller and Network Interface

In order to evaluate the merits of different strategies in a fair manner, it is necessary that the coherence protocols used must support compatible block states and set of network transactions (except for those caused by out-of-order (OoO) message arrivals). In other words, any performance optimizations on the protocols must be applied equally. For this reason, instead of using the FLASH's extensively optimized and fine tuned cache coherence protocol, we used two compatible (FIFO and non-FIFO) cache coherence protocols originally presented and verified in [22]. These two protocols assume a full-map bit-vector directory organization and have been designed using the optimization style of intervention forwarding [6]. We disabled the prefetching part from the FIFO coherence protocol because it was not supported in the non-FIFO coherence protocol. On the other hand, we extended both protocols to support the release consistency memory model [11]. The extended non-FIFO coherence protocol was used in the system configuration following the I-CC strategy, while the extended FIFO coherence protocol was used in the configurations using either the T-FIFO strategy or our C-FIFO strategy. The occupancy on the coherence controller for each type of network transaction was assumed to be equal (mostly 14 processor cycles or 46.6 ns) in all configurations to eliminate any artifacts on performance caused by different implementations. For invalidation intervention, 5 additional cycles were assumed for each invalidation message sent. The equal occupancy assumption might favor the I-CC strategy because of the higher complexity of the non-FIFO protocol. However, if any bias were caused as a result, it should make our performance evaluation more conservative and increase the confidence of the overall conclusions of this study.

On a raw network interface, as was used in I-CC strategy, 4 memory bus cycles (40 ns) were assumed for preparing an outgoing message (header) and 4 cycles for error-checking and dispatching on an incoming message. For C-FIFO strategy, 1 more cycle was assumed for selecting virtual channel based on the block address. For T-FIFO strategy, the reordering capability was modeled as described earlier in section 4.2. For simplicity, it was assumed that

[2] In SGI Origin, two processing modules are connected to the same hub to share its high cost [6].

the same number of memory bus cycles (denoted as r) were needed for performing a load-modify-store operation on a sequence number counter each time. In other words, functions like getting a new sequence number, checking the expected sequence number, and comparing the sequence number of a message in the reordering buffer took r cycles each. It was also assumed that $2 * r$ cycles were needed for either entering or exiting reordering buffer.

Table 2 summarizes the default memory hierarchy parameters, coherence controller occupancy delays, network parameters, and network interface parameters used in our simulations.

Table 2: Default system parameters used in simulation.

Memory Hierarchy Parameters	
Processor frequency	300MHz
L1 cache access	1 proc cycle
L1 cache line size	32 bytes
L1 cache set associativity	1
L1 cache size per node	8 Kbytes
L1 coalescing write buffers	8
Data-path width between L1 and L2	8 bytes
L1 cache line fill time	6 proc cycles
L2 cache access	2 proc cycle
L2 cache line size (L)	256 bytes
L2 cache set associativity	2
L2 cache size per node	128 Kbytes
L2 coalescing write buffers	8
Memory bus frequency	100MHz
Memory response delay	20 proc cycle
Memory bus width (W)	16 bytes per mem cycle
L2 cache line fill time	20 proc cycles + L/W mem cycles
Memory block access time	20 proc cycles + L/W mem cycles
Coherence Controller Occupancy	
Directory check	7 proc cycles
Directory check and update	14 proc cycles
Invalidation request	5 proc cycles each
Cache intervention	3 proc cycles
Network Interface Parameters	
Outgoing constructing	4 mem cycles
Incoming dispatching	4 mem cycles
Reordering overhead parameter	r mem cycles
Control message size	16 bytes
Data message size	16+L bytes
Network Parameters	
Network frequency	100MHz
Channel width / Flit size	8 bytes
Link Propagation	1 net cycle
Router switch delay	1 net cycle
Routing time	1 net cycles
Link synchronization time	2 net cycles
Physical network	1
Virtual channels	2

6.4 Configurations Evaluated and Measurements

In the simulation experiments, we considered five configurations: one using the I-CC strategy, one using the C-FIFO strategy, and three using the T-FIFO strategy (denoted as T-FIFO-1, T-FIFO-2, and T-FIFO-3). The T-FIFO-1, T-FIFO-2, and T-FIFO-3 configurations differed only in the assumed value of r. This value was 10, 20, and 30, respectively, corresponding to aggressive, intermediate, and conservative implementations. In all configurations, two virtual channels were used. In the I-CC configuration, one virtual channel was used for transferring request messages and the other for transferring response messages. In the C-FIFO configuration, one virtual channel was used for transferring both request and response messages related to blocks which have an even directory ID and the other was used for transferring both request and response messages related to blocks which have an odd directory ID. In the T-FIFO-1, T-FIFO-2, and T-FIFO-3 configurations, both virtual channels were used equivalently and adaptivity between the two were allowed to reduce network congestion.

We present our simulation results in two sets: the overall execution time and average latency of network transaction. These measurements help us to correlate the behavior of the underlying network with the overall DSM system performance and to gain important insights. The overall execution time is broken down into four components: the CPU computation busy time (B), 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 I-CC configuration for each application. The latencies of network transaction are presented in absolute time (microseconds). The presented latencies are the average of the measured raw communication latencies over all types of network transactions. The raw communication latency was globally clocked from the instance when a message generation is triggered at the source coherence controller till the instance when that message is removed from the waiting queue at the destination coherence controller. It includes the time spent at the network interface of both ends and the time spent inside the network with various types of contention occurring in between. In addition to the timing results, we also measured the frequency of pairwise out-of-order (OoO) arrivals/events and the fraction of block correlated pairwise out-of-order (OoO) arrivals/events among the pairwise out-of-order (OoO) arrivals/events in the T-FIFO configurations. All these measurements together with related discussions are reported in Section 7.

6.5 Applications

We used six applications — FFT, MP3D, Radix, Barnes, LU, and Water — in our simulation evaluations. All are real applications or challenging computational kernels ported from the Stanford SPLASH/SPLASH2 [25, 32] benchmark suite.

FFT is the complex 1-D Fast Fourier Transformation computation kernel. It represents a wide variety of applications in the field of digital signal processing. The n complex input points are organized as a $\sqrt{n} \times \sqrt{n}$ matrices which is partitioned into submatrices and stored contiguously for each processor to improve data locality. The communication, occurring in matrix transpose steps, has an all-to-all regular pattern.

MP3D is a rarefied fluid flow dynamics simulation using the Monte Carlo method. It represents a range of applications involving the force and movement of high speed objects in an extremely low density environment. Two large arrays of structures are used to store the information of molecules (the objects) and cells. The algorithm partitions the computation load on the basis of each molecule and assigns it statically to each processor. The amount of computation in MP3D may change slightly depending the underlying machines because there are a number of harmless data race conditions in the code. The communication in MP3D is largely unstructured.

Radix is the integer radix sort kernel, a major task in data base applications. It iteratively creates local histograms by permuting keys and by merging them into a global histogram. The permutation phase requires, depending on the key values, all-to-all irregular communication (in the worst case).

Barnes is an application representative of the class of hierarchical N-body methods, used in astrophysics, electrostatics, and plasma physics, among others. In this application, a globally shared space-motion-mass tree (its primary data structure) is continually reconstructed and repartitioned at every time-step. Particles are assigned to each processor and accessed through pointer arrays. The communication pattern in Barnes is hierarchical and irregular.

LU is the computation kernel of lower and upper matrix decomposition. It represents a wide range of dense linear algebra applications. The 2D $n \times n$ input matrix of double precision numbers is converted into a 4D $N \times N \times B \times B$ matrix for purposes of reducing communication and load balancing. The elements assigned to the same processor are allocated contiguously to improve spatial locality. The communication pattern is one-to-many and largely regular.

Water is an N-body molecular dynamics application. It evaluates forces and potentials in a system of water molecules over a period of time. The main data structure is an array of the molecules in the system. It adopts a predictor-corrector method ($O(n^2)$ algorithm) to compute the forces and potentials in each time-step. Irregular communication occurs when the cell list is updated which in turn is caused by the movement of molecules into and out of cells.

Table 3 summarizes 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* before running them on the simulator testbed.

Table 3: Applications and the used input sizes.

Application	Description	Problem Size
FFT	Fast Fourier Transform	$64K$ complex points
MP3D	Low density fluid flow simulation	$50,000$ particles
Radix	Integer radix sort	$1M$ keys, $1K$ radix, max $1M$
Barnes	Hierarchical N-body simulation	$8K$ particles, $\theta = 1.0$, 4 time steps
LU	Blocked LU decomposition	512×512 doubles, 8×8 blocks
Water	Molecular dynamics simulation	512 molecules, 4 time steps

7 Results and Discussions

In this section we first evaluate the overall performance of DSM systems using three different strategies: I-CC, T-FIFO, and C-FIFO. Next, we study the impact of several key system parameters on the performance of systems using different strategies.

7.1 Overall Results

We first studied the overall execution time of each benchmark application on different configurations. The results are shown in Fig. 9. It can be observed that the C-FIFO configuration always delivers either the best or very close to the best performance among the five evaluated configurations. The actual performance difference between the C-FIFO configuration and the best one varies across applications, with a minimum of 0% seen in MP3D to a maximum of 2.7% in Radix. For most applications, this difference is less than 1%. Considering the significant reduction in complexity of the network interface and/or coherence protocol, this performance result makes C-FIFO strategy very attractive.

From the timing breakdowns, it can be observed that the CPU computation time remained almost constant across all 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. Different designs or implementations of the node controller target to reduce various waiting times due to communication but not the computation times. The breakdowns are very helpful in understanding how the communication to computation ratio changes across different configurations for all applications. Two facts can be easily observed from the breakdowns of this study. First, the computation to communication ratios are within typical ranges for each individual application [12], consistent with results reported by other research [15, 12, 24, 1, 21] Second, for all applications except Radix the write waiting time is negligible. This correlates to earlier research on release consistency models [11].

For all applications except Radix, the performance trend of the five evaluated configurations is the same. Namely, the configurations using an intelligent coherence protocol or block correlated FIFO channels perform better than the configurations using total FIFO channels. Across applications, the configuration delivering the best performance changes between the two strategies: I-CC and C-FIFO. However, the difference between these two strategies is relative small. Among the configurations using total FIFO channels, as expected, the performance decreases as the overhead of message reordering (r) increases for all applications. The difference between the best and the worst performance of the five evaluated configurations varies widely across applications ranging from 5.7% in LU to 104.7% in MP3D. This shows that the design and implementation of a high performance node controller is critical in a scalable DSM system.

The main reason for the relative poor performance in configurations using total FIFO channels is that a noticeable portion of the overhead incurred by the message reordering at the network interface lies on the critical (uncontented) path of every network transaction. For most applications, this causes a significant increase to the latency of network transaction on average. This can be seen from the curves of average latency of a network transaction, shown in Fig. 10, on the five evaluated configurations during the execution of applications.

A strong correlation between the overall execution time of an application and the average

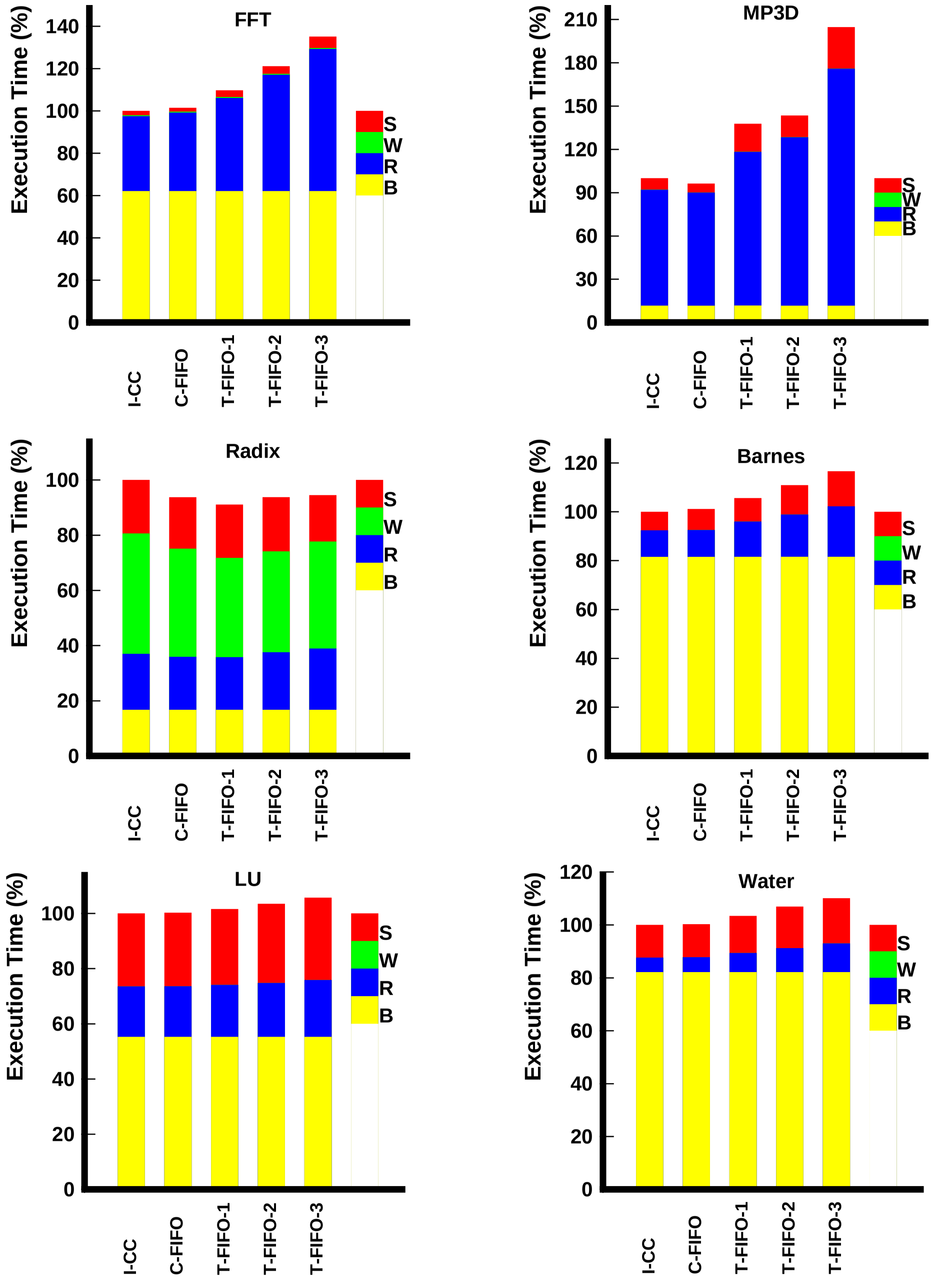

Figure 9: The overall execution times of benchmark applications on different configurations.

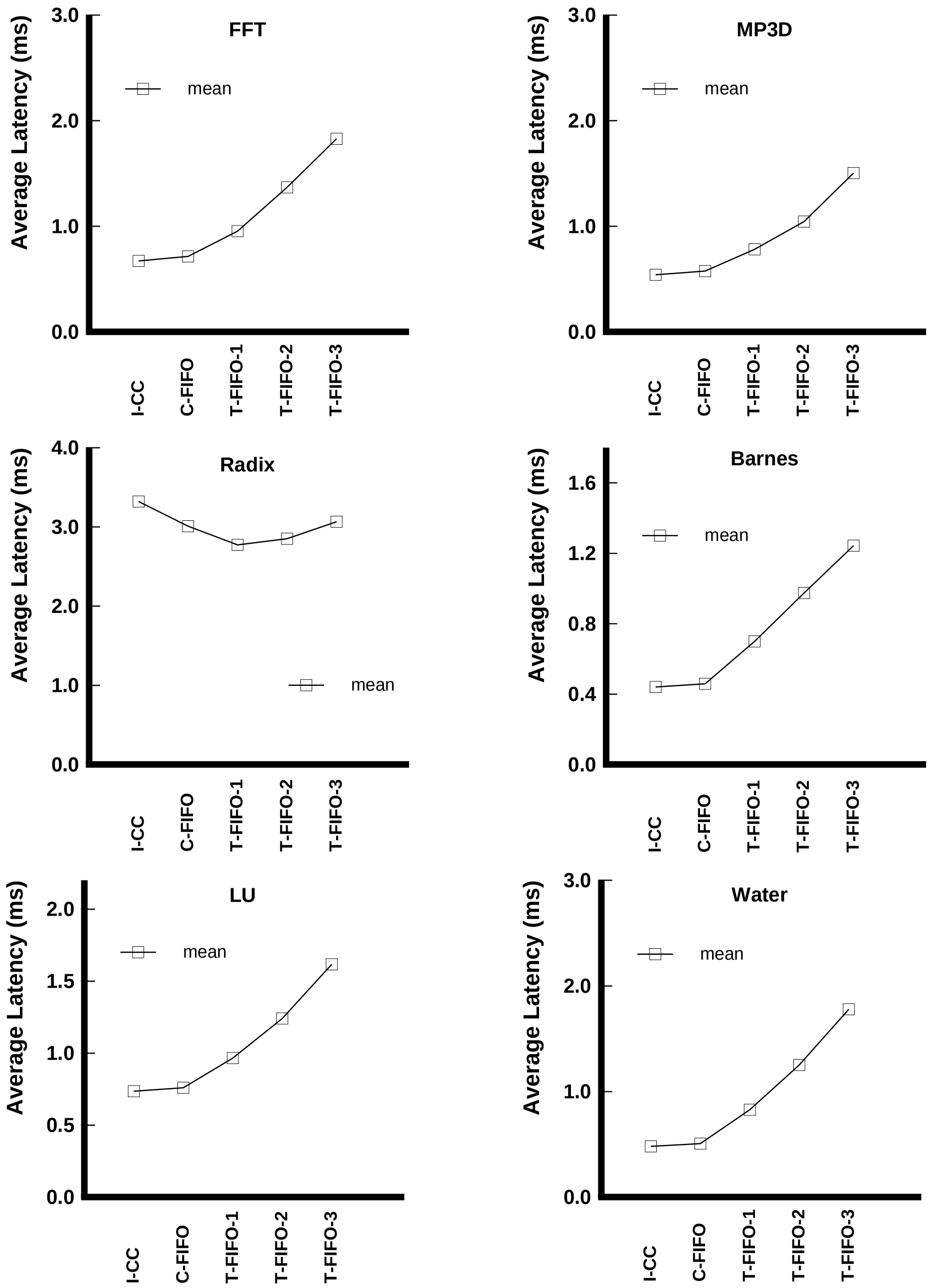

Figure 10: The average latency of a network transaction on different configurations.

latency of network transaction of a system exists because of two reasons: (a) the computation remains almost same across configurations, as pointed out above; and (b) the total number of network transactions changes marginally across configurations. The actual numbers of network transactions in the five evaluated configurations for the benchmark applications are shown in Table 4. It can be observed that the changes in the count are very small compared to the corresponding changes in the average latency of network transaction (shown in Fig. 10) for any application. This indicates that average latency of a network transaction is a very important metric for the overall performance of a CC-NUMA system. Any new techniques for reducing this metric can potentially improve the overall system performance significantly.

Table 4: Total number of network transactions.

	FFT	MP3D	Radix	Barnes	LU	Water
I-CC	294,089	5,344,046	5,862,551	1,585,346	521,460	612,401
C-FIFO	290,764	5,091,898	5,857,961	1,572,074	511,527	612,143
T-FIFO-1	292,648	5,169,787	5,882,002	1,604,268	496,503	610,822
T-FIFO-2	292,545	4,823,979	5,890,881	1,560,806	484,856	610,434
T-FIFO-3	292,928	4,741,001	5,870,935	1,566,683	477,415	610,990

Now, let us understand the performance results of different configurations for Radix. As shown in Fig. 9, the C-FIFO configuration still provides the second best performance. However, the I-CC configuration is surprisingly outperformed by all other configurations. As for other applications, a similar trend occurs to the average latencies of network transaction, as shown in Fig. 10. A further investigation reveals that this phenomenon can be related to the usage of the parallel virtual channels in different configurations.

In Radix, especially at the permutation phase when the local histograms are merged into the global histogram in the earlier iterations, multiple writers and false sharing generates bursty heavy network traffic. This causes the network in a configuration to be temporarily congested. In the T-FIFO-1, T-FIFO-2, and T-FIFO-3 configurations, the two virtual channels were used equivalently to transfer any messages. To its contrary, in the I-CC configuration, one of the two virtual channels was dedicated to transfer request type of messages, the other to response type. Due to the imbalance between the request and response type traffic, the virtual channels and thus network bandwidth in the latter configuration are not used as effectively as those in the former when the network is congested. However, in the C-FIFO configuration, the use of the virtual channels and thus network bandwidth is improved to a certain extent depending on the distribution of the block addresses related to live messages. Therefore, under the same traffic condition, the increase in the latency of network transaction caused by network congestion differs among the configurations. In the set of experiments for Radix, this difference of increase happened to be more prominent than the difference caused by the reordering overhead as shown in Fig. 10. The result indicates that the C-FIFO strategy can potentially adjust better than the I-CC strategy when severe network congestion occurs.

To gain more insights into the severeness of penalty on the average latency of network transaction exerted by total FIFO channels, we examined the characteristics of pairwise out-of-order (OoO) arrivals and events in the most aggressive T-FIFO-1 configuration. The T-FIFO-2 and T-FIFO-3 configurations cause worse penalties. Table 5 shows the average percentage of

OoO arrivals among all messages between the same pair, the average number of OoO events generated per OoO arrival, the average percentage of correlated OoO events among all pairwise OoO events, and the average correlated OoO events generated per message between a pair from the same set of experiments. It can be observed that the average correlated OoO events generated per message between a pair is a very small number in the order of 10^{-2} or less. Our experimental data (not included) for the T-FIFO-2 and T-FIFO-3 configurations show that this number is even smaller. Such a decrease can be explained by the increase in the minimum interval between two successive messages between the same pair of nodes, thus, less chances for a message to bypass its predecessor. Therefore, it is reasonable to believe that the average correlated OoO events generated per message between a pair of nodes is small (likely in the order of 10^{-2}) for the current technology. Such a small range indicates that the overhead incurred on every message transmission at the network interface in the T-FIFO strategy over-kills system performance, as discussed in section 4.2.

Table 5: Summary of out-of-order (OoO) messages per pair of processing nodes in a total FIFO channel system (T-FIFO-1 configuration).

	FFT	MP3D	Radix	Barnes	LU	Water
OoO Arrival Percent	1.91%	1.95%	2.65%	0.76%	3.49%	1.03%
OoO Events/OoO Arrival	1.0002	1.0006	1.0129	1.0000	1.0000	1.0002
Co OoO Event Percent	66.65%	0.10%	5.65%	6.55%	12.55%	0.01%
Co OoO Events/Msg	1.27e-2	1.95e-5	1.50e-3	4.98e-4	4.38e-3	1.03e-6

In order to see that our main conclusion about the three design strategies is valid for a wide range of system implementations, we also performed performance comparisons by varying a number key design parameters related to L2 cache and network topology. In the following subsections, we focus on the trend of overall system performance of different strategies and present the results from three configurations: I-CC, C-FIFO, and T-FIFO-2. We selected T-FIFO-2 because it is considered the most practical implementation of the T-FIFO strategy for current technology.

7.2 Impact of Cache Parameters

In this section, we study the impact of L2 cache line size and L2 cache size on the performance trend among the strategies.

7.2.1 Impact of Smaller Cache Line Size

We first studied the case where L2 caches use a different line size. It is well known that varying the cache line size has the bath-tub effect on the overall execution time. A smaller L2 cache line reduces the performance cost of a network transaction. In addition, it has the benefit in alleviating the damaging impact of false sharing. On the other hand, the amortized cost per network transaction is higher. The increased L2 cache misses can generate more network transactions. Overall, the burstiness of network traffic is smoothened. Figure 11 shows the execution time breakdowns of the three configurations with a L2 cache line of 128 bytes size. Compared to the corresponding results with a line size of 256 bytes (Fig. 9), the overall performance improves in all three configurations for each application. This conclusion is reached based on two facts: (a) the absolute CPU computation busy times barely changed for each application in all our experiments; and (b) the relative percentages of the CPU computation busy times increase from Fig. 9 to Fig. 11. For some communication stressed applications, MP3D and Radix, the improvement is quite large (up to 49.6%). This correlates to the fact that both SGI Origin [16] and HAL S1 [31] systems use a L2 cache line of 128 bytes. Interestingly, with this cache line size, the performance gap among different configurations, especially the improvement of C-FIFO over T-FIFO-2, increases. This is because the performance of the communication subsystem becomes more prominent under reduced overall execution time.

Also shown in Fig. 11 is a noticeable change in the performance trend for Radix, where the I-CC configuration outperforms T-FIFO-2; and C-FIFO performs the best. This is a direct result of reducing the severeness of temporary network congestion. Now, the overall performances of the three configurations show a consistent trend for all applications. This indicates that the I-CC strategy always performs better than the T-FIFO strategy if severe network congestion can be avoided.

Overall, the results again confirm that the C-FIFO strategy is very promising in reducing hardware complexity and increasing overall performance of scalable DSM systems using current generation technology.

7.2.2 Impact of Larger Cache Size

Let us now examine the case where each processing node has a larger L2 cache. A larger L2 cache helps to reduce capacity misses and thus potentially decreases the communication to computation ratio. Figure 12 shows the results of the three configurations with 256 Kbytes L2 caches (twice of the size used in experiments shown in Fig. 9). As expected, the overall performance improves in all three configurations for each application as the cache size increases. However, the performance trend remains unchanged (compared to Fig. 9) among all three strategies while the performance gap grows. This again attributes to the more prominent role of the communication subsystem on the overall performance.

7.3 Impact of Network Topology

We also studied the impact of interconnection network topology on the performance improvement. We first examined the case where more processing nodes are connected to the same switch. Then, we examined the case where network bisection bandwidth decreases.

7.3.1 Impact of Larger Cluster Size

In general, a larger cluster increases the amount of intra-switch communication and reduces inter-switch communication. Figure 13 shows the results from systems consisting of 16 switches connected in a 4D hypercube topology with each switch connecting to 4 processing nodes (thus a total of 64 processing nodes). It is to be noted that each switch was a 8-port bidirectional crossbar. The link bandwidth was maintained as the same as the default system. Due to the

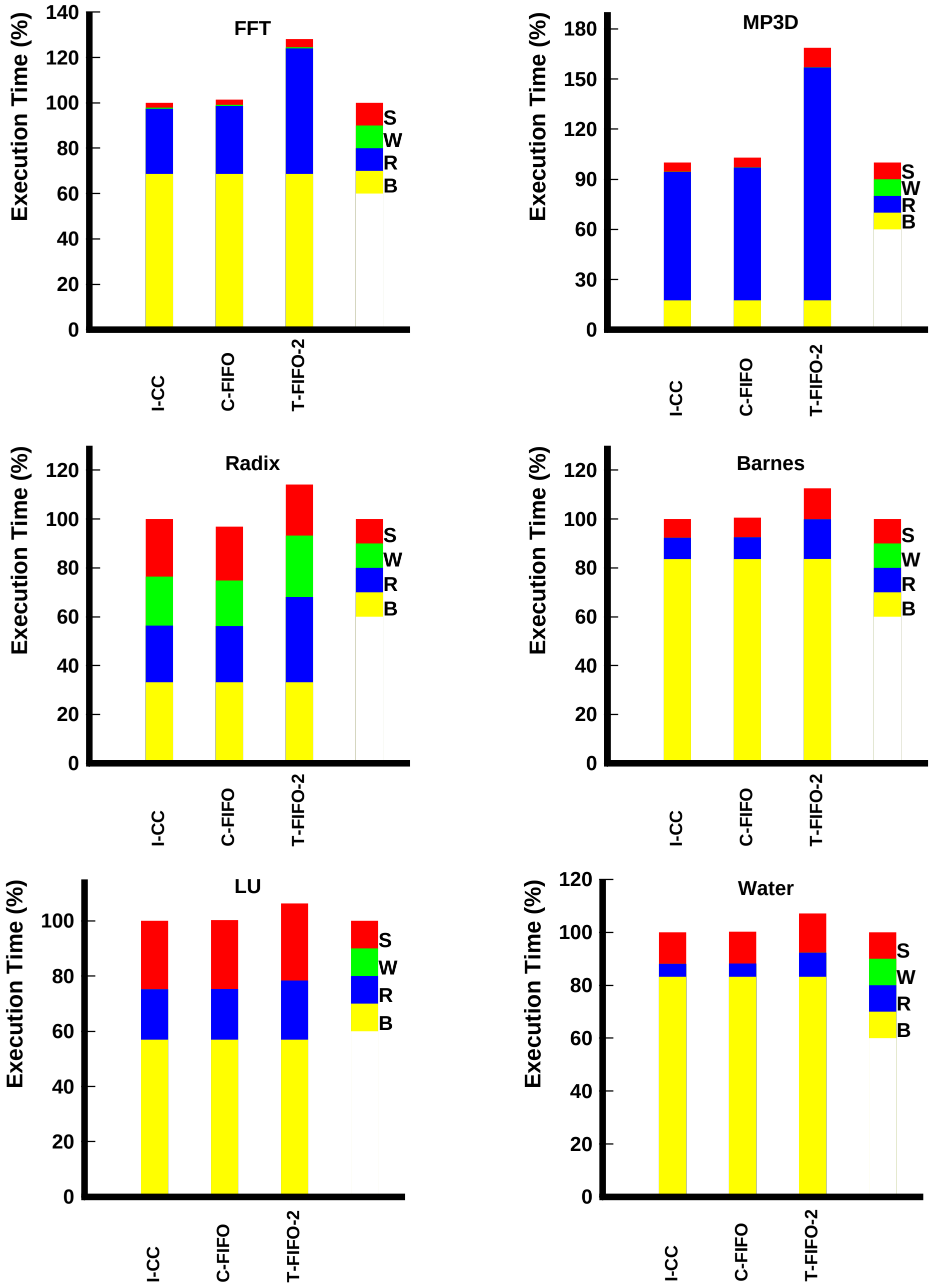

Figure 11: Impact of smaller cache line (128 bytes) on the overall execution times of benchmark applications on three configurations.

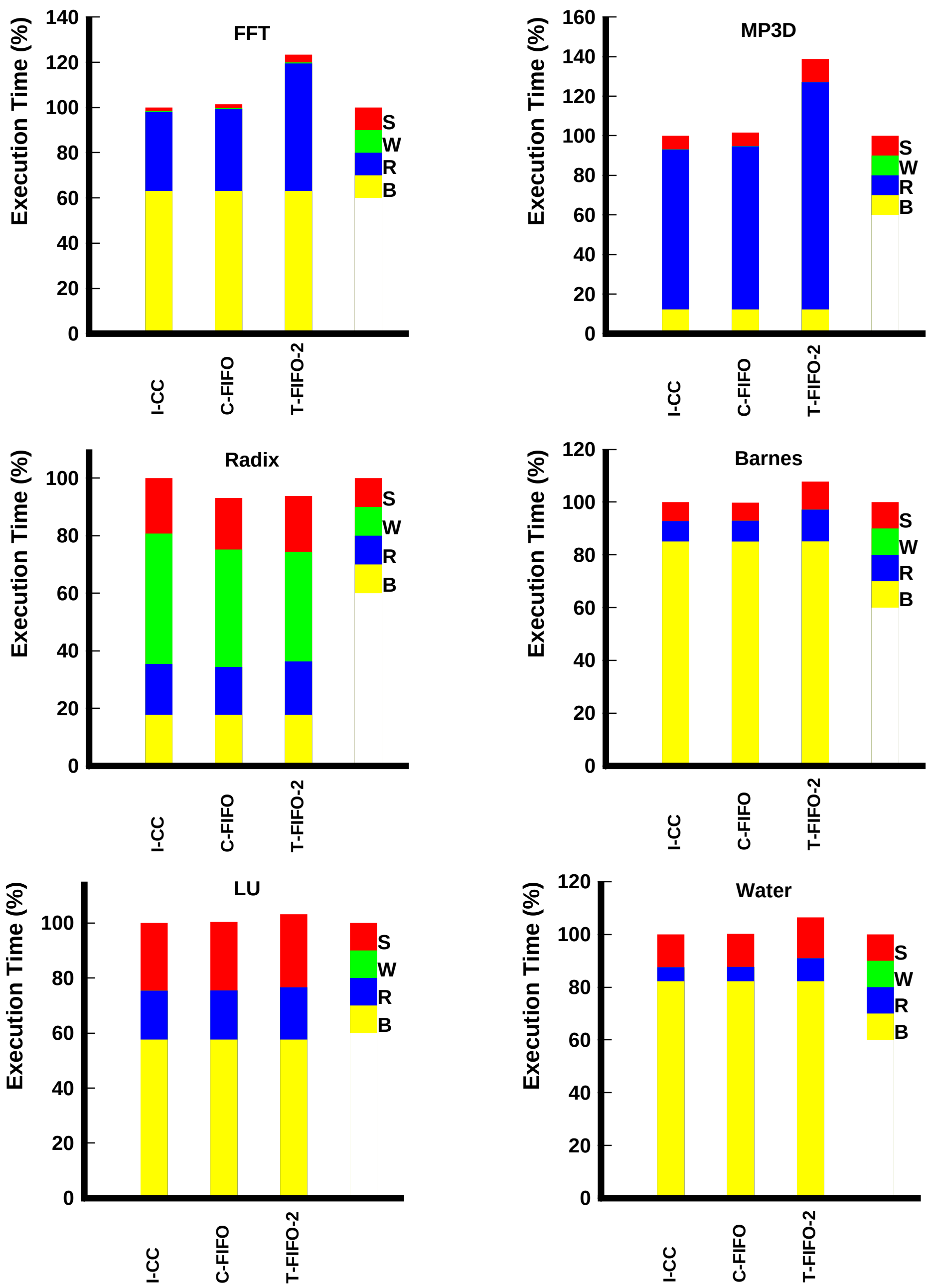

Figure 12: Impact of larger cache size per node (256K) on the overall execution times of benchmark applications on three configurations.

larger cluster size, the total number of switches decreased. As a result, the actual bisection bandwidth of the network also decreased. It can be observed that the overall performance of each application decreases (compared to Fig. 9) in all three configurations as cluster size increases. Interestingly, the performance gap among the configurations changes differently, depending on the application. This gap increased in MP3D; barely changed in Radix, Barnes, LU, and Water; and decreased in FFT. However, the basic trend of performance among the three strategies remained unchanged from Fig. 9.

7.3.2 Impact of Smaller Network Bisection Bandwidth

Under the constraint of fixed number of nodes, as the dimensionality of a k-ary n-cube network decreases, the bisection bandwidth decreases, the diameter increases, and therefore the severeness of network congestion increases. The main difference between the experiments in this section and the previous section is that the amount of intra-switch and inter-switch communication remains unchanged. Figures 14 and 15 show the results from systems containing 32 switches connected by a 3D and 2D mesh, respectively. In all experiments, each switch connects to 2 processing nodes, the same as in our base configuration. It is to be noted that the number of bidirectional ports supported by a switch changed as the connectivity decreased. It can be observed that the overall performance of each application decreases significantly in all three configurations as topology changes from 5D (Fig. 9) to 3D (Fig. 14) to 2D (Fig. 15). As the same time, the performance gap among the configurations increases for Radix and decreases for all other applications. Both the increase and decrease of the gap can be attributed to the growing prominence of congestion in the network, as explained in subsection 7.1. Nevertheless, our conclusion about the overall performance of the three strategies still remains unchanged.

8 Conclusion

In this paper, we have compared and contrasted the strengths and weaknesses of alternative strategies for incorporating multiple-path networks in scalable distributed shared memory (DSM) systems. Figure 16 provides a qualitative comparison of these strategies along three fundamental design axes. We first presented two previously known strategies, the intelligent cache coherence protocol (I-CC) strategy and the total FIFO channel (T-FIFO) strategy. For the T-FIFO strategy, we have presented an efficient implementation of a sliding window protocol network interface. Next, we presented a new strategy, C-FIFO, using block correlated FIFO channels. The new strategy has been described in detail and it has been shown that it can combine the advantages of the two previously known strategies and avoid their respective drawbacks. A low complexity implementation of this strategy using current technology has been presented. Table 6 summarizes the overall comparison of these strategies.

We performed extensive simulations to measure the performance of these three strategies using six benchmark applications. The impact of cache parameters (such as cache size and cache line size) and network parameters (such as cluster size and network dimensionality) on the performance of these six benchmarks under each of the three strategies was evaluated. We find that the proposed C-FIFO strategy outperforms the T-FIFO strategy (used by the HAL S1) for most cases and can improve performance by a factor of up to 40% for practical

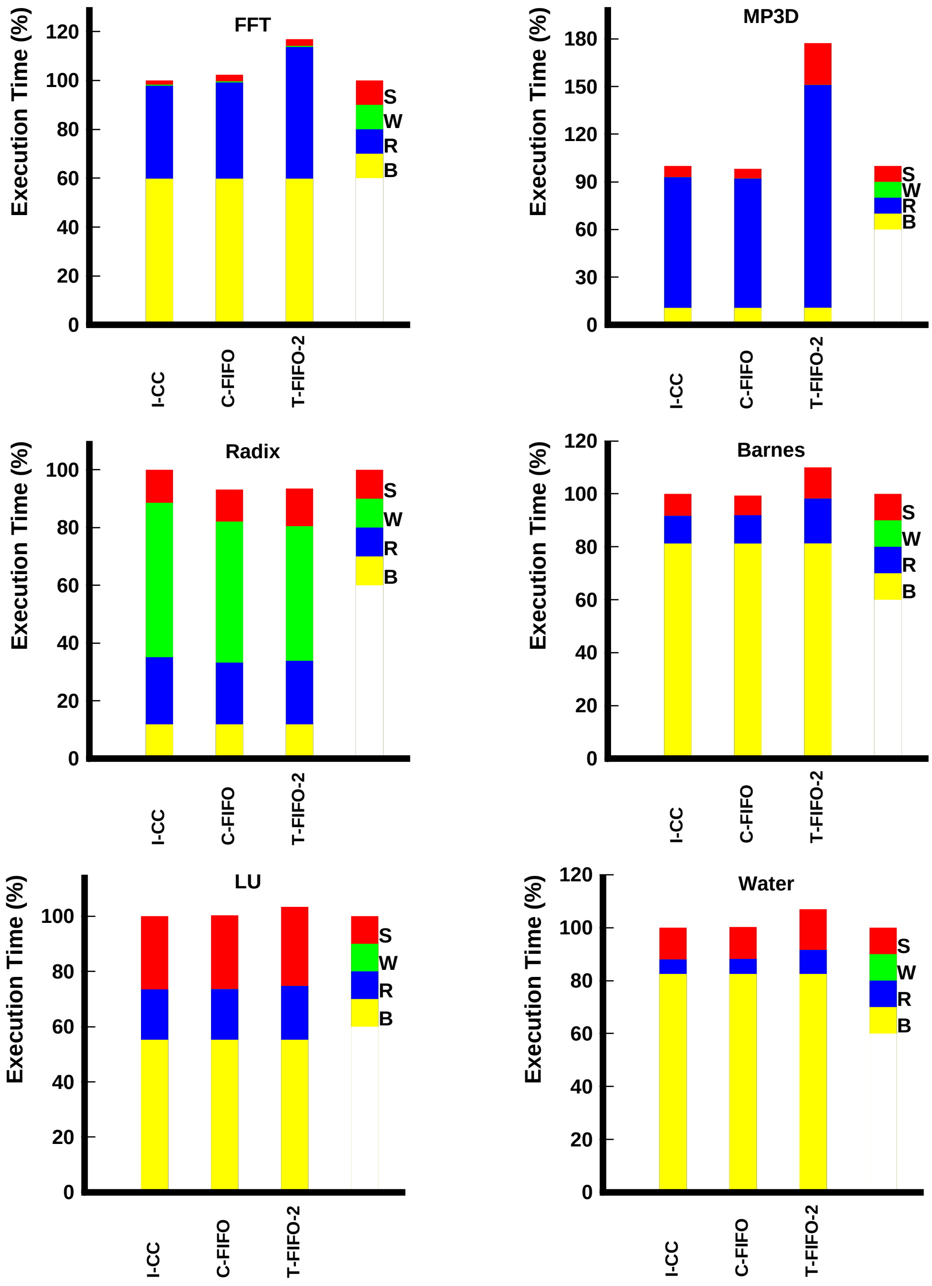

Figure 13: Impact of larger cluster size (4 processing nodes per switch) on the overall execution times of benchmark applications on three configurations.

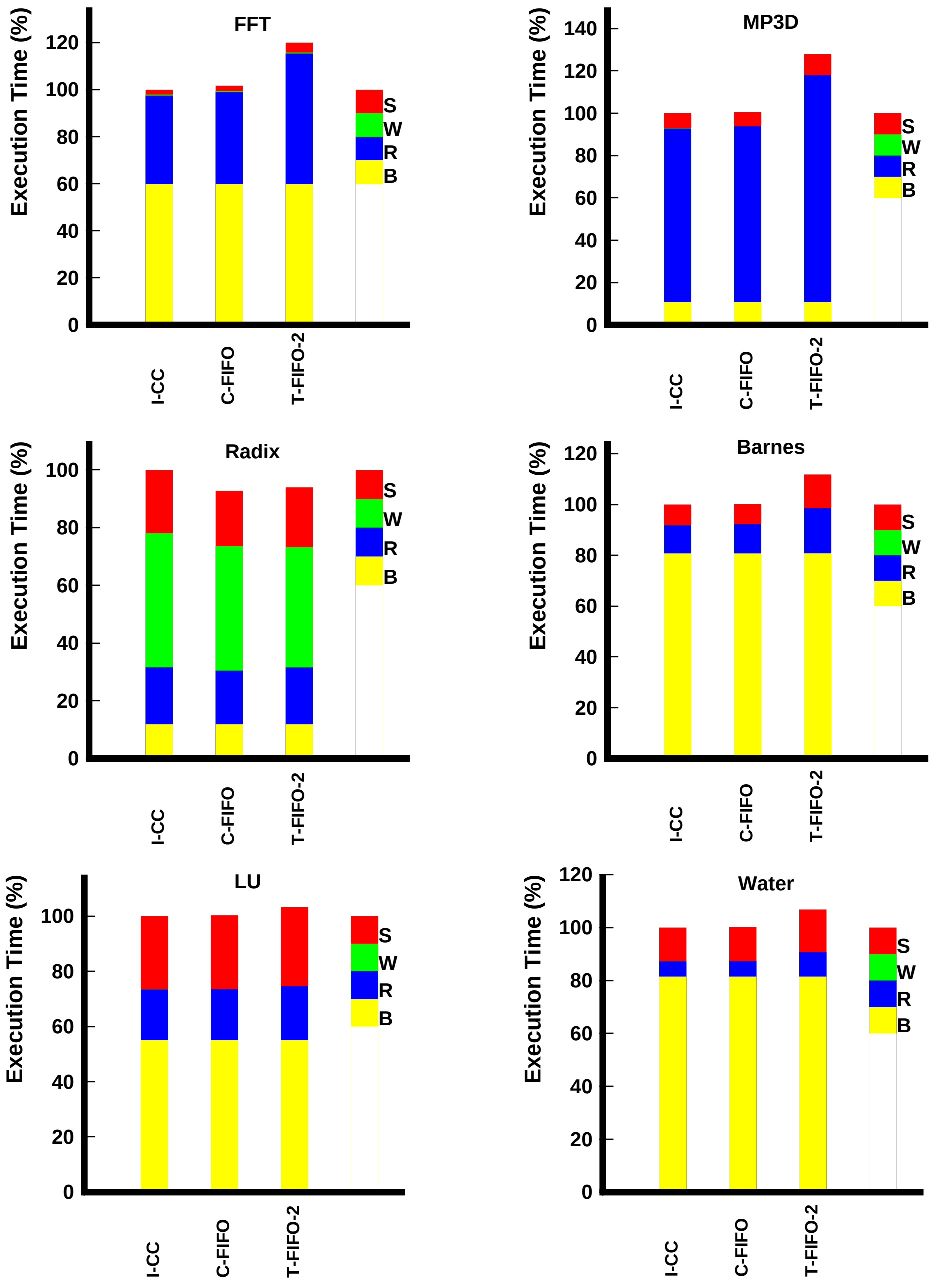

Figure 14: The overall execution times of benchmark applications on three configurations. Switches are connected using a 3D mesh topology.

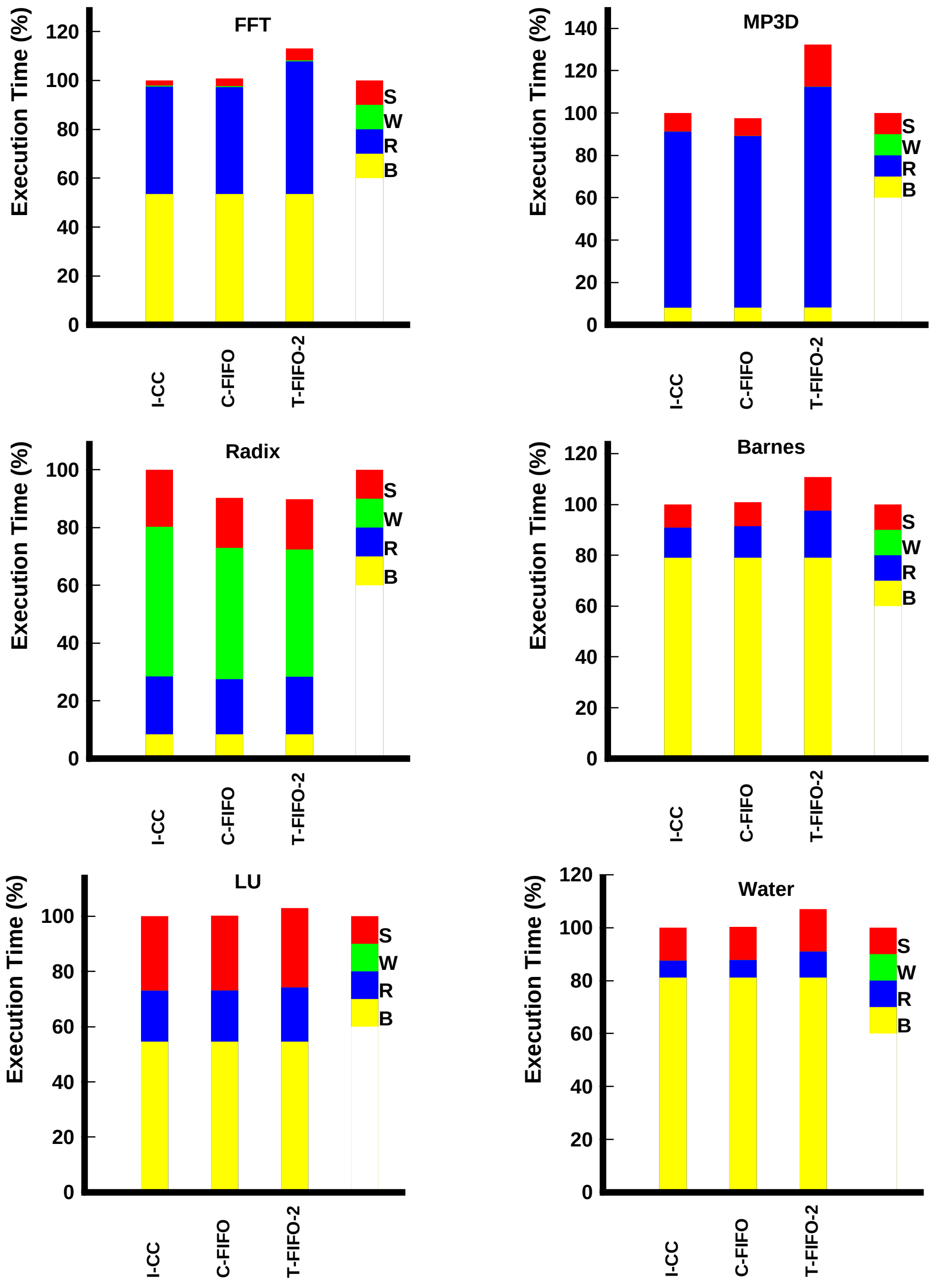

Figure 15: The overall execution times of benchmark applications on three configurations. Switches are connected using a 2D mesh topology.

Table 6: Overall comparison of alternative strategies for incorporating multiple-path netowrks in DSM systems.

Strategy	Complexity		Network Adaptivity	System Performance
	Coherence Controller	Network Interface		
I-CC	High	Low	Medium High	Mostly High
T-FIFO	Low	High	High	Mostly Low
C-FIFO	Low	Low	Medium Low	High

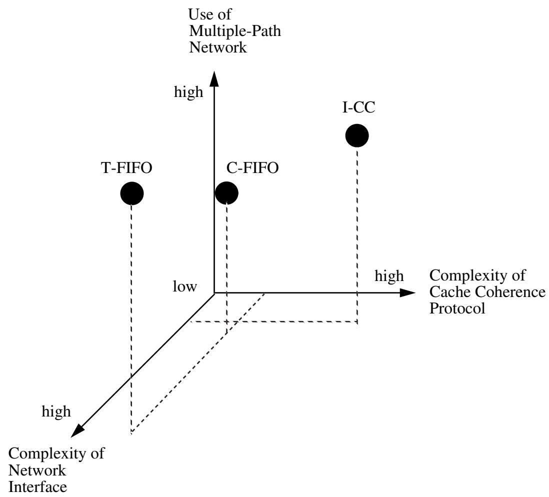

Figure 16: A qualitative comparison of alternative strategies for incorporating multiple-path networks in DSM systems along three fundamental design axes.

system configurations. Furthermore, in most cases, the C-FIFO strategy performs as well as the I-CC strategy (used by the SGI Origin) and is typically slower by less than a factor of 1%. Taking into account the significant simplicity of the C-FIFO strategy, we conclude that it offers a significant cost-performance advantage over the existing (I-CC and T-FIFO) strategies. Current and future generation DSM systems can therefore benefit significantly by using this strategy.

References

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

[2] Y. Aydogan, C. B. Stunkel, C. Aykanat, and B. Abali. Adaptive Source Routing in Multistage Interconnection Networks. In *Proceedings of the International Parallel Processing Symposium*, pages 258–267, 1996.

[3] T. Brewer and G. Astfalk. The Evolution of the HP/Convex Exemplar. In *Proceedings of COMPCON Spring'97: Forty-Second IEEE Computer Society International Conference*, pages 81–86, February 1997.

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

[5] D. Chaiken and A. Agarwal. Software-Extended Coherent Shared Memory: Performance and Cost. In *International Symposium on Computer Architecture*, pages 314–324, 1994.

[6] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware-Software Approach.* Morgan Kaufmann, March 1998. To appear.

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

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

[9] A. T. Eiriksson and K. L. McMillan. Using formal verification/analysis methods on the critical path in system design: A case study. In *Proceedings of Computer Aided Verification Conference*, Liege Belgium, 1995. Springer Verlag.

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

[11] K. Gharachorloo et al. Memory Consistency and Event Ordering in Scalable Shared-memory Multiprocessors. In *17th International Symposium on Computer Architecture*, pages 15–26, May 1990.

[12] M. Heinrich et al. The performance impact of flexibility in the stanford flash multiprocessor. In *The sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, San Jose, CA, October 1994.

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

[14] K. Kennedy et al. A Nationwide Parallel Computing Environment. *Communication of the ACM*, 40(11):63–72, 1997.

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

[16] 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.

[17] D. Lenoski et al. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In *Proceedings of the 17th Annual Symposium on Computer Architecture*, pages 148–159, May 1990.

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

[19] T. Lovett and R. Clapp. STiNG: A CC-NUMA computer system for the commercial marketplace. In *International Symposium on Computer Architecture*, pages 308–317, 1996.

[20] A. Mu et al. A 285 MHz 6-port Plesiochronous Router Chip with Non-Blocking Cross-Bar Switch. In *1996 Symposium on VLSI Circuits: Digest of Technical Papers*, pages 136–137, 1996.

[21] V. S. Pai, P. Ranganathan, and S. V. Adve. The Impact of Instruction-Level Parallelism on Multiprocessor Performance and Simulation Methodology. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 72–83, February 1997.

[22] Fong Pong. Symbolic State Model: A New Approach for the Verification of Cache Coherence Protocols. PhD Thesis, University of Southern California, August 1995.

[23] 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.

[24] S. K. Reinhardt, J. R. Larus, and D. A. Wood. Tempest and Typhoon: User-Level Shared Memory. In *Proceedings of the International Symposium on Computer Architecture*, pages 325–337, April 1994.

[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] IEEE Computer Society. IEEE Standard for Scalable Coherence Interface (SCI). *IEEE Standard 1596-1992*, 20, August 1993.

[27] Per Stenstrom. A survey of cache coherence schemes for multiprocessors. *IEEE Computer*, 23(6):12–24, June 1990.

[28] C. Stunkel. Challenges in the Design of Contemporary Routers. In *Proceedings of Parallel Computer Routing and Communication Workshop (PCRCW)*, June 1997.

[29] Yuval Tamir and Gregory L. Frazier. Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches. *IEEE Transactions on Computers*, 41(6):725–737, June 1996.

[30] M. Tomasevic and V. Milutinovic. *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions.* IEEE Computer Society Press, 1993.

[31] 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.

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