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

Donglai Dai and Dhabaleswar K. Panda

November 1998

This paper will be appearing in the Special Issue on Cache Memory and Related Problems,
IEEE Transactions on Computers, February 1999.

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

Donglai Dai and Dhabaleswar K. Panda

Abstract— **Modern high performance networks being used for scalable distributed shared memory (DSM) systems support multiple paths to increase bandwidth and/or reduce contention. Such networks violate the constraint of pairwise in-order message delivery 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 SGI Origin series, is to employ an intelligent cache coherence protocol which detects and resolves all race conditions caused by out-of-order (OoO) events. The second strategy, used in the HAL Mercury series, is to use a sophisticated network interface (NI) which detects and remedies every OoO event 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 propose a new strategy that uses *block correlated FIFO channels*. This new strategy detects all potential race conditions and prevents them from occurring. It allows the use of a simple cache coherence protocol and an inexpensive NI. We also present an efficient implementation of this strategy based on current technology. Detailed simulations are performed using benchmark applications to evaluate the performance of our new strategy. The results indicate that compared to the existing strategies, our new strategy always provides either the best or close to the best overall performance. This study also 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, network interface, directory-based protocols, and cache coherence.**

I. Introduction

Scalable distributed shared memory (DSM) systems, represented by the cache coherent non-uniform memory access (CC-NUMA) systems, are emerging as the trend in building parallel systems because they provide the much desired programmability. Using a directory based cache coherence protocol, CC-NUMA systems automatically replicate remote data in the private caches of a processing node when needed. Examples of such systems include Stanford DASH/FLASH [1], MIT Alewife [2], Convex Exemplar [3], Sequent NUMA-Q [4], SGI Origin [5], and HAL Mercury [6].

The inherent processor-to-memory communication in CC-NUMA systems results in frequent exchange of short messages. Communication latency is a pronounced component of the penalty for a remote memory access in such systems [7], [8]. The interconnection networks in most CC-NUMA systems employ modern switches/routers like SGI SPIDER [9] and HAL PRC [6] which use performance-enhancing mechanisms such as cut-through switching, multiple virtual channels, and efficient buffering extensively. As a result, multiple paths exist between any given pair of nodes, either implemented physically [10] or supported logically via virtual channels [6], [9]. Multiple paths alleviate network congestion, increase throughput, and reduce average message latency. One side effect of using multiple paths 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 [5], many directory-based cache coherence protocols [11] designed for DSM systems require pairwise in-order arrival.

To exploit the advantages of a multiple-path network in a DSM system, architects currently use two alternative strategies. The first one, used in 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 drawback of this strategy is the high complexity in the design, verification, and implementation of the resulting coherence protocol [6]. The second strategy, used in HAL Mercury, is to enhance the network interface with reordering capability to ensure that all pairwise messages seen by the 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.

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. It allows the use of both an in-order (FIFO) cache coherence protocol and a simple network interface. We also present an efficient implementation of this strategy based on current technology. To quantitatively evaluate the performance of our proposed strategy, we have performed simulation experiments using practical system configurations and benchmark applications. For most applications, the results show that the new strategy is less than 1% slower than the SGI Origin approach and up to 40% *faster* than the HAL Mercury approach. This demonstrates that DSM systems using our proposed strategy can provide very competitive performance at a much lower cost.

The authors are with Dept. of Computer and Information Science, The Ohio State University, Columbus, OH 43210-1277, E-mail: {dai,panda}@cis.ohio-state.edu.

The rest of this paper is organized as follows. Section II reviews issues relevant to communication in DSM systems. Sections III and IV describe the intelligent cache coherence protocol strategy and the total FIFO channel strategy, respectively. In Section V, we propose the block correlated FIFO channel strategy and present an efficient implementation. Section VI describes our methodology for performance evaluation and discusses the results, and finally in Section VII we draw the conclusions.

II. Issues in CC-NUMA System Design

In this section, we first overview the architecture of a typical CC-NUMA system and then discuss several key issues relevant to incorporating multiple-path networks in such systems.

System Model: A typical CC-NUMA 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 contains a cache coherence controller and separate interfaces for the processor module, the memory/directory module, and the network. It also contains logic for deadlock prevention and arbitration between the interfaces. The cache coherence controller observes and resolves all cache misses and synchronization operations. The network interface (NI) is responsible for sending/receiving messages and it provides the desired communication abstraction to the remaining part of a node. The abstraction of sending, transferring, and receiving a message is often called a *network transaction* [13].

Complexity of Cache Coherence Protocol: The cache coherence protocol of a CC-NUMA system ensures that a *read* to any memory location always gets the content from the latest *write* which the reading processor is aware of. Various cache coherence protocols differ on the types of states and messages being used. The complexity of a protocol increases with the number of state types and message types. Although fast high coverage techniques exist [12], complete coverage verification methods rely on exhausting the reachability of finite state machines or their equivalents. Until now, verification of a highly complex coherence protocol remains a monumental task for computer architects [15]. Complex protocols also introduce larger overhead on the critical path, leading to longer occupancy for each state transition in the protocols.

OoO Arrival and OoO Event: As mentioned earlier, modern high-performance networks support multiple virtual channels [6], [9] or multiple physical routes [10] between a pair of nodes to increase bandwidth and/or reduce congestion. When virtual channels are supported, a path can be viewed as a chain of virtual channels from the source node to the destination node. The latency of a message varies depending on the length of the message, the length of the used path, and congestion along the path being used. Therefore, it is possible that between a source-destination pair, a message using one path reaches its destination sooner than a previously sent message using another path. Such a bypassing scenario is commonly known as an *out-of-order (OoO) arrival* of messages [13]. A message may bypass more than one messages. For each of the bypassed messages, exactly one *out-of-order (OoO) event* occurs.

Example Race Condition: In a CC-NUMA system, many types of race conditions may be caused by OoO events. Figure 1 shows one example. Node N_1 sends a write-miss request ($Rxq(1)$) to block B_1 whose home is node N_2 and whose current owner is node N_3. Once the request Rxq arrives at N_2, a flush request ($Fsh(2)$) is sent by N_2 to N_3. Assume right before the arrival of Fsh at N_3, the (dirty) copy of B_1 is displaced and sent back ($Wbk(3a)$) to N_2. Once Fsh reaches N_3, a reply ($Frp(3b)$) is sent by N_3 back to N_2 as the acknowledgment. Either message Wbk or Frp may reach N_2 first (i.e., a race condition) in a network allowing OoO arrivals. This uncertainty becomes a problem for many cache coherence protocols because after the Frp arrives, reply $Rxp(4)$ normally takes a copy from the main memory at N_2 which may or may not have been updated by the Wbk. Such a (coherence-sensitive) race condition must be detected and resolved properly.

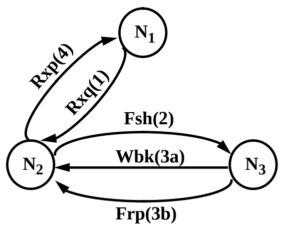

Fig. 1. An example race condition caused by an OoO event.

For resolving race conditions, in the following three sections, we examine two existing strategies and their drawbacks, and propose a new efficient strategy.

III. Intelligent Cache Coherence Protocol (I-CC) Strategy

The basic idea behind the I-CC strategy is to build enough intelligence into the cache coherence controller so that it can detect and revolve all race conditions. Typically, the network in the target system is logically separated into request and response networks. Within each logical network, complete freedom is granted in routing a message from its source to destination. Figure 2(a) illustrates the network connecting two arbitrary nodes in such a system. Four parallel paths are shown, with two paths in each of the two logical networks. This strategy can use simple and efficient network interfaces with a selection function which decides whether the request or the response network should be used for transferring a particular message.

In this strategy, the coherence protocol must detect a race condition on-the-fly. The popular approach is to denote 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 'spoiling' message arrives before the expected 'authoritative' message

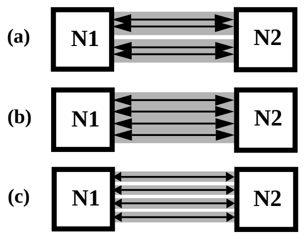

Fig. 2. The networks between two nodes in a system using: (a) the I-CC, (b) the T-FIFO, and (c) the C-FIFO strategies.

(e.g., message 3b bypasses 3a in Fig. 1), the spoiler (message 3b) can be buffered or NAKed. Other actions like reverting to a simpler protocol, or combining transactions can also be taken to resolve it depending on different performance optimization goals [13]. For a more detailed description on such cache coherence protocols, readers can refer to [1], [5], [13]. It is to be noted that OoO events across memory blocks do not cause any race condition by virtue of synchronization mechanisms (e.g., release consistency [1], [13]) used in the CC-NUMA systems.

In order to get a rough idea on the complexity of an intelligent cache coherence protocol, let us examine the protocol used in SGI Origin [5] as an example. As reported in [13], there are 15 requests (including invalidations and interventions) and 39 responses, 7 directory states[1], and more than 7 cache states used in this protocol[2]. Compared to cache coherence protocols that are used in the next two strategies, this complexity is significantly higher. This is the main drawback of the I-CC strategy.

Overall, CC-NUMA systems using the I-CC strategy are expected to deliver high performance because the benefits of the multiple-path network are aggressively exploited. Unless race conditions have actually occurred, forward progress in applications can always be made.

IV. Total FIFO Channel (T-FIFO) Strategy

The philosophy of the T-FIFO strategy [6] is to build a powerful network interface (NI) to shield the effect of OoO arrivals from the rest of the node. The network can use any path for transferring a message from its source to destination. The in-order message arrival property, ensured by the NI, effectively eliminates all race conditions for cache coherence protocols, allowing many simple and efficient (in-order based) cache coherence protocols (such as those discussed in [11], [13], [15]) to be used. Figure 2(b) illustrates an example network with four parallel paths connecting two arbitrary nodes in a system using the T-FIFO strategy. For a race condition as shown in Fig. 1, this strategy ensures that the cache coherence protocol always sees message 3a before message 3b regardless of which one has arrived at the NI of node N_2 first.

The T-FIFO strategy demands two key functions from the NI: a) to detect all occurrences of OoO arrivals and b) to remedy every such occurrence into a set of in-order arrivals. A representative design is to use a sliding window protocol enhanced with reordering capability. Figure 3 shows the pseudo-code description of such an NI. The code is self-explaining. Detailed description on the NI design can be found in [14]. The main enhancement contains (pairwise) sequence number manipulation for OoO arrival detection and buffer management for restoring message order.

```
Sending:                          Receiving:
  gen-seqno(msg) /* r */            receive(msg)
  construct(msg)                    if (out-of-seq(msg)) /* r */
  send(msg)                           reorder-defer(msg) /* 2r */
                                    else { dispatch(msg)
                                      reorder-dispatch()} /* f(r) */
```

Fig. 3. Pseudo-code description of the NI used in the total FIFO channel strategy. 'Dispatch' means delivering 'msg' to the cache coherence controller. The comments show the time delays for reordering operations.

Typically, OoO arrivals occur rarely in a system. Not every OoO arrival causes a race condition. However, the overhead for detecting OoO arrivals slows down every network transaction. This is the main drawback of the T-FIFO strategy.

V. Block Correlated FIFO Channel (C-FIFO) Strategy

In this section, we propose a new strategy which uses simpler cache coherence controllers and inexpensive NIs. We first develop several key concepts used in this strategy and then present an efficient implementation.

A. Eliminating Race Conditions

A careful examination can reveal the fact that in a DSM system, the services of memory operations targeting to the same memory block must be serialized. For convenience of discussion, let us define two memory operations to be *block correlated memory operations* if their target memory blocks are the same. We can similarly define *block correlated messages* and *block correlated network transactions.* A race condition is a scenario in which the serialization order of two block correlated memory operations may be observed differently by the involved nodes because of OoO events. Such an OoO event can be defined as a *block correlated OoO event.* All race conditions are caused by block correlated OoO events. Because each memory operation is serviced by a chain of network transactions, to eliminate all race conditions, a necessary and sufficient condition is that all block correlated network transactions must maintain the in-order property while non-correlated network transactions can proceed arbitrarily[3]. The important performance implications of this condition will become clear in Section VI.

B. Block Correlated FIFO Channel (BCFC)

A *block correlated FIFO channel (BCFC)* can be defined as the abstraction of a FIFO channel used for transfer-

[1]Including one state for efficient page migration.
[2]Many optimizations targeted towards better performance have been incorporated into the SGI Origin coherence protocol. Otherwise, these numbers may be slightly smaller.
[3]When the order between non-correlated network transactions is critical, it will be ensured by the synchronization mechanism of the DSM system.

ring messages relating to a particular memory block. Every memory block has a distinct block correlated FIFO channel (BCFC) associated with it between each pair of nodes. The number of BCFCs between two given nodes is equal to the total number of blocks in the shared address space. Such BCFCs put no restriction on the arrival order of non-block correlated messages because they travel in separate BCFCs. Therefore, BCFCs provide a perfect mechanism for enforcing the necessary and sufficient condition for eliminating race conditions discussed in the previous subsection.

C. The Strategy and Implementation

We propose to build DSM systems using simple (inorder based) cache coherence protocols and block correlated FIFO channels (BCFCs). We call such a strategy the *block correlated FIFO channel (C-FIFO) strategy.* This strategy prevents a race condition, such as the one shown in Fig. 1, from ever occurring because messages 3a and 3b are block correlated and thus message 3b will never bypass message 3a.

At a first glance, it seems impractical to implement BCFCs in a system because of the large number of BCFCs required. Most current generation networks, such as those used in IBM SP [10], HAL Mercury [6], and SGI Origin [5], support only a small number of (physical or logical) parallel paths (FIFO channels). However, if we map (or collapse) multiple BCFCs onto a single path, the path can be viewed as *block correlated* to a particular *set* of memory blocks. Using this mapping idea, a simple implementation of the C-FIFO strategy becomes realizable. For efficiency purposes, the mapping from BCFCs to parallel paths should incur a minimum overhead. An ideal mapping function is the ‘modulo’ operation on the block address associated with each BCFC. Since the number of parallel paths is typically small in a system, selecting a path for transferring a message based on a few least significant bits (wires) of the block address associated with the message can be done easily. Figure 2(c) illustrates four block correlated paths (i.e., four logical networks) connecting two nodes in a system using the C-FIFO strategy.

It is clear that the C-FIFO strategy can use cache coherence protocols as simple as those used in the T-FIFO strategy and NIs as efficient as those used in the I-CC strategy. Figure 4 shows a qualitative comparison of these strategies. In the next section, we compare and discuss the performance of these three strategies, quantitatively.

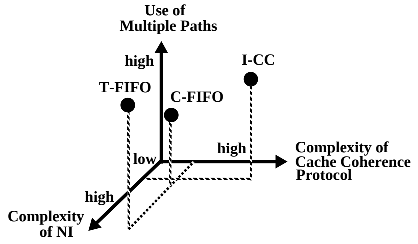

Fig. 4. A qualitative comparison of alternative strategies for incorporating multiple-path networks in DSM systems along three fundamental design axes. The ‘Use of Multiple Paths’ axis indicates the flexibility for transferring a given message.

VI. Performance Evaluation

This section presents our simulation-based performance evaluation methodology, simulation results, and discussions.

A. Methodology

The hardware cache coherent multiprocessor we simulated had a generic DSM architecture similar to the FLASH system [1], supporting release consistency. Each processor was modeled as a 300 MHz single-issue superscaler, supported by a 8 KB direct-mapped write-through L1 D-cache (32 bytes per line), a perfect I-cache, and a 128 KB 2-way associative write-back L2 cache (256 bytes per line). Coalescing write buffers were provided for both L1 and L2 caches (8 entries each). A read miss in L1 cache and L1 write buffer stalled the processor. The memory module was assumed to support multiple read/write ports, with a 66 ns response time. A directory cache was assumed to eliminate any directory access stall. The bandwidth between node controller and each module (e.g., processor and memory modules) was assumed to be 1.6 GB/sec.

The system had 64 processing nodes connected via a 5D hypercube with 2 nodes per switch (like SGI Origin [5]). The network used the dimensional order (from low to high) wormhole routing scheme [7], [8], [13] and supported 2 virtual channels. Each virtual channel was assumed to be 64 bits wide and have an input buffer of 256 bytes and an output buffer of 32 bytes. The network switch was assumed to operate at 100 MHz, with 20 ns, 10 ns, and 10 ns for link synchronization, routing, and crossing the crossbar, respectively. The link propagation was assumed to be 10 ns.

Two compatible cache coherence protocols originally presented and verified in [15] were used, with extensions for release consistency. The non-FIFO coherence protocol was used for the I-CC strategy, while the FIFO coherence protocol was used for the T-FIFO and the C-FIFO strategies. The occupancy on cache coherence controller for each network transaction was assumed to be mostly 46.2 ns (14 processor cycles), with additional 16.5 ns (5 processor cycles) for each invalidation message. The equal occupancy assumption might favor the I-CC strategy because of the higher complexity of its coherence protocol. However, based on our simulation results, this bias does not affect the overall conclusions of this study.

The basic network interface (NI), as used in the I-CC strategy, was assumed to take 40 ns for sending or receiving a message. For the C-FIFO strategy, extra 10 ns was assumed for selecting a path. For the T-FIFO strategy, the reordering capability was modeled as shown in Fig. 3. For simplicity, a delay of r was assumed for performing a load-modify-store operation on a sequence number and $2r$ for depositing or removing an OoO message at the NI.

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 value of r, which was assumed to be 100 ns, 200 ns, and 300 ns, respectively, corresponding to aggressive, intermediate, and conservative implementations. In the I-CC configuration, one virtual channel was used for transferring requests and the other for responses. In the C-FIFO configuration, one virtual channel was used for transferring messages related to even addressed blocks and the other for odd ones. In the T-FIFO-1, T-FIFO-2, and T-FIFO-3 configurations, both virtual channels were used with no distinction.

We used six applications — FFT (64K points), MP3D (50K particles), Radix (1M keys, 1K radix), Barnes (8K particles, 4 steps), LU (512 by 512 matrix), and Water (512 molecules, 4 steps) — in our simulation evaluations. All are real applications or challenging computational kernels ported from the Stanford SPLASH/SPLASH2 benchmark suite. These applications were compiled using the optimization level equivalent to O2 of *gcc*.

B. *Results and Discussions*

In this section, we evaluate the performance of DSM systems using the I-CC, T-FIFO, and C-FIFO strategies. We study the overall execution times of applications, the characteristics of network transactions, the characteristics of block correlated OoO arrivals and OoO events, and the impact of several key system parameters.

B.1 Overall Results

The overall execution times of benchmark applications on different system configurations are shown in Fig. 5. All times are normalized to that of the I-CC configuration. The times are further broken down into four components: the CPU computation busy time (Busy), the memory read waiting time (Read), the memory write waiting time (Write), and the synchronization waiting time (Sync). 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 from 0% (in MP3D) to 2.7% (in Radix). For most applications, this difference is less than 1%. With the simplicity at the NI level and at the cache coherence controller level, such a performance makes the C-FIFO strategy very attractive.

For all applications except Radix, the performance trend of the five configurations is the same. Namely, the configurations using either the I-CC strategy or the C-FIFO strategy outperform those using the T-FIFO strategy. The performance of the I-CC configuration (strategy) and that of the C-FIFO configuration (strategy) are very comparable. Among the configurations using total FIFO channels, as expected, the performance decreases for all applications as the reordering overhead (r) increases.

From the timing breakdowns, it can be observed that the CPU computation busy time remains almost constant across all configurations in every application. This is expected because the configurations (thus the strategies) target to reduce various waiting times due to communication,

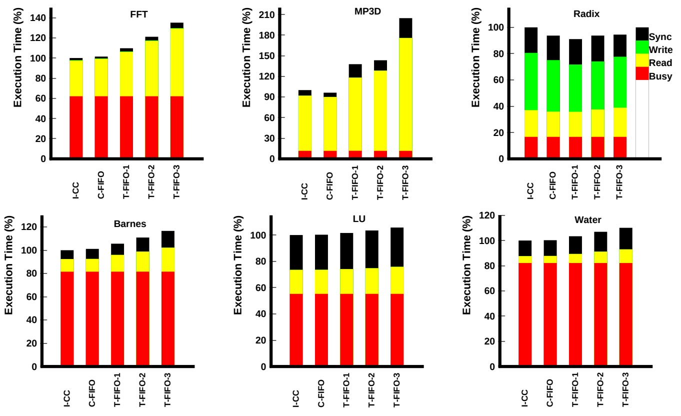

Fig. 5. The overall execution times of benchmark applications on different configurations.

not the computation busy time. Two facts can be easily observed from the breakdowns. First, the computation to communication ratios are within typical ranges for each individual application, consistent with results reported by other research [1], [2]. Second, for all applications except Radix, the write waiting time is negligible. This correlates to earlier research on release consistency [13].

B.2 Characteristics of Network Transactions

Figure 6 shows the average latency of network transaction on the five evaluated configurations during the execution of applications. For most applications, the latency is significantly higher in the T-FIFO strategy than those in the I-CC or C-FIFO strategy. This shows that a noticeable portion of the overhead incurred by the message reordering at the network interface (NI) lies on the critical path of every network transaction, especially the uncontented network transactions.

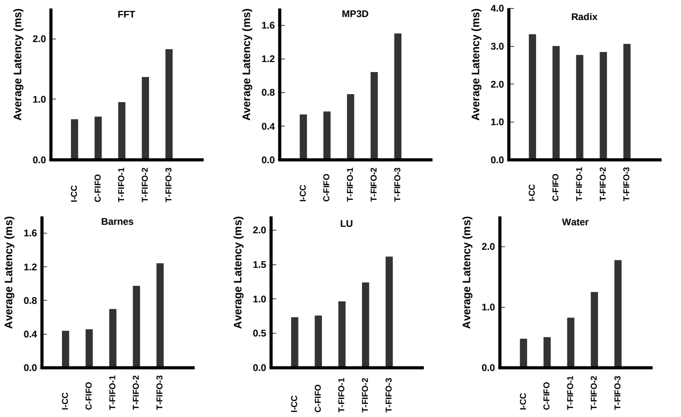

Fig. 6. The average latency of a network transaction on different configurations.

It can be observed that a strong correlation exists between Figs. 5 and 6. This is caused by two reasons: a) the computation remains almost same across configurations; and b) the total number of network transactions changes marginally across configurations, as shown in Table I. This shows that the average latency of a network transaction has a strong impact on the overall performance of a CC-NUMA system. Any new technique for reducing this metric can potentially improve the overall system performance significantly.

TABLE I
Total number of network transactions (in thousands).

	FFT	MP3D	Radix	Barnes	LU	Water
I-CC	294	5,344	5,862	1,585	521	612
C-FIFO	290	5,091	5,857	1,572	511	612
T-FIFO-1	292	5,169	5,882	1,604	496	610
T-FIFO-2	292	4,823	5,890	1,560	484	610
T-FIFO-3	292	4,741	5,870	1,566	477	610

Now, let us examine the performance results of Radix in Fig. 5. As shown in the figure, the C-FIFO configuration still provides the second best performance. However, the I-CC configuration is surprisingly outperformed by all other configurations. This phenomenon can be explained by 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 temporarily congestion in the network. In the T-FIFO-1, T-FIFO-2, and T-FIFO-3 configurations, the two virtual channels were used equivalently for transferring any messages. To its contrary, in the I-CC configuration, one virtual channel was dedicated to transferring request messages, the other to transferring response messages. Due to the imbalance between the request and response 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 usage of the virtual channels and thus network bandwidth is improved to a certain extent depending on the temporal distribution of the block addresses. This result indicates that the T-FIFO and C-FIFO strategies can adjust better than the I-CC strategy when severe network congestion occurs.

B.3 Characteristics of OoO Arrivals and OoO Events

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 OoO events in the most aggressive T-FIFO-1 configuration. The T-FIFO-2 and T-FIFO-3 configurations cause worse penalties. Table II shows the rate of OoO arrivals between two nodes (row A), the average number of OoO events generated per OoO arrival (row B), the average rate of block correlated OoO events per OoO event (row C), and the average number of block correlated OoO events per message arrival (row D). It can be observed that the average number of block correlated OoO events per message arrival (i.e., the erroneous outcome of a race condition) is a very small number, in the order of 10^{-2} or less. Our experimental data for the T-FIFO-2 and T-FIFO-3 configurations also showed that this number was even smaller. Such a small value indicates that the overhead incurred on every message transmission at the network interface (NI) in the T-FIFO strategy overkills system performance.

TABLE II
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
A	1.9e-2	2.0e-2	2.6e-2	7.6e-3	3.5e-2	1.0e-2
B	1.0002	1.0006	1.0129	1.0000	1.0000	1.0002
C	0.66	1.0e-3	5.6e-2	6.5e-2	0.12	1.0e-4
D	1.3e-2	1.9e-5	1.5e-3	5.0e-4	4.4e-3	1.0e-6

The above evaluations were based on a specific set of implementations. To ensure that the conclusions are not limited to certain implementations, we also studied the impact of several key design parameters relevant to communication. In the next subsection, we present results on the impact of L2 cache line size. Other results such as the impact of L2 cache size, network topology, and bisection bandwidth can be found in [14].

B.4 Impact of Smaller L2 Cache Line Size

It is well known that varying the cache line size of a given cache has the bath-tub effect on the overall execution time. A smaller L2 cache line reduces the average latency of network transaction and alleviates false sharing between nodes. On the other hand, the amortized cost per network transaction is higher. The increased misses at the L2 cache generate more network transactions. Overall, the burstiness of network traffic is smoothened. Figure 7 shows the execution time breakdowns of the I-CC, the C-FIFO, and the T-FIFO-2 configurations with a L2 cache line of 128 bytes. Compared to the corresponding results with a line size of 256 bytes, the overall performance improves in all three configurations for each application. Indirectly, this trend can be observed from Figs. 5 and 7 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. 5 to Fig. 7. It is to be noted that both SGI Origin [5] and HAL Mercury [6] 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 the C-FIFO configuration over the T-FIFO configuration, increases. This is because the reordering overhead becomes more prominent under reduced overall execution time.

VII. Conclusion

In this paper, we have proposed a new, block correlated FIFO channel (C-FIFO) strategy for incorporating multiple-path networks in scalable DSM systems. This new strategy combines the advantages and avoids the drawbacks of two existing strategies, i.e., the intelligent cache coherence protocol (I-CC) strategy and the total FIFO channel (T-FIFO) strategy. An efficient implementation of this new strategy using current technology has also been presented. Detailed performance evaluations demonstrate that for most applications, our proposed C-FIFO strategy outperforms the T-FIFO strategy by a factor of up to 40%

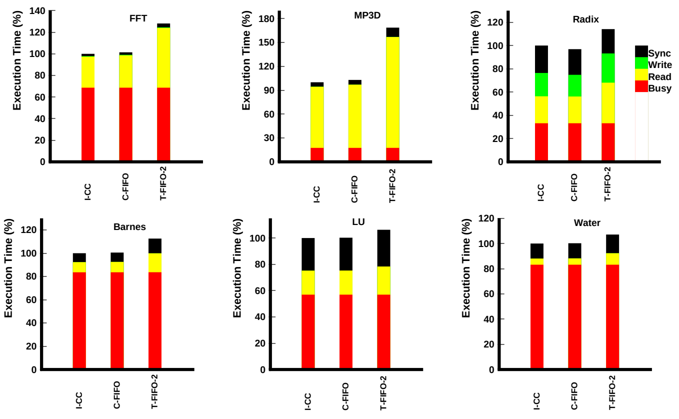

Fig. 7. Impact of smaller cache line (128 bytes) on the overall execution times of benchmark applications on three configurations.

and performs almost equal to the I-CC strategy at a much lower cost.

This study shows that not all network transactions in DSM systems are equally important at a given time. The effective latency of network transactions which can contribute to forward progress in applications is crucial for overall system performance. With the simplicity at the cache coherence controller level and at the network interface level, the C-FIFO strategy offers a significant cost-performance advantage over the existing strategies. Current and future generation DSM systems can therefore benefit significantly by using this strategy.

Acknowledgments

This research is supported in part by NSF Career Award MIP-9502294 and NSF Grant CCR-9704512. The authors would like to thank Prof. Jose Duato and Dr. Rajeev Sivaram for constructive discussions and to the anonymous reviewers for suggestions on presentation. Some related publications are available on the PAC web page `http://www.cis.ohio-state.edu/~panda`.

References

[1] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy, "The Stanford FLASH Multiprocessor", in *Proceedings of the International Symposium on Computer Architecture*, 1994, pp. 302–313.

[2] A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung, "The MIT Alewife Machine: Architecture and Performance", in *International Symposium on Computer Architecture*, 1995, pp. 2–13.

[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*, February 1997, pp. 81–86.

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

[5] 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)*, June 1997, pp. 241–251.

[6] W.-D. Webber, S. Gold, P. Helland, T. Shimizu, T. Wicki, and W. Wilcke, "The mercury interconnect architecture: A cost-effective infrastructure for high-performance servers", in *Proceedings of International Symposium on Computer Architecture (ISCA-24)*, 1997, pp. 98–107.

[7] 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, pp. 454–461.

[8] D. Dai and D. K. Panda, "How Can We Design Better Networks for DSM Systems?", in *Proceedings of the 1997 Parallel Computer Routing and Communication Workshop (PCRCW'97), Lecture Notes in Computer Science #1417*, Atlanta, GA, June 1997, pp. 171–184.

[9] M. Galles, "Spider: A High-Speed Network Interconnect", *IEEE Micro*, pp. 34–39, January/February 1997.

[10] 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*, 1996, pp. 258–267.

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

[12] 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.

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

[14] D. Dai and D. K. Panda, "Exploiting the Benefits of Multiple-Path Network in DSM Systems: Architectural Alternatives and Performance Evaluation", Tech. Rep. OSU-CISRC-8/98-TR33, The Ohio State University, August 1998.

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

Donglai Dai received a BS degree in computer science from Xian Jiaotong University, Xian, China, in 1985, and an MS degree in computer science from The Ohio State University in 1994. He is currently a PhD candidate at OSU. His research interests include the design, modeling, and evaluation of high-performance computing systems, with special focuses on distributed shared memory (DSM) systems and network-based computing systems. He has worked on a variety of projects, including kernel firmware for the LISP_M1 prototype and the EXTENT parallel compiler. He is a member of the IEEE Computer Society.

Dhabaleswar K. Panda received the B.Tech degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1984, the M.E. in electrical and communication engineering from the Indian Institute of Science, Bangalore, India, in 1986, and the Ph.D. in computer engineering from the University of Southern California, USA, in 1991. He is an Associate Professor in the Department of Computer and Information Science, The Ohio State University, Columbus, USA. His research interests include parallel computer architecture, wormhole-routing, interprocessor communication, collective communication, network-based computing, and high-performance computing. He has published over 60 papers in major journals and international conferences related to these research areas.

Dr. Panda has served on Program Committees and Organizing Committees of several parallel processing conferences. He was a Program Co-Chair of the CANPC '97 and CANPC '98 workshops and a Co-Guest-Editor for two special issue volumes of Journal of Parallel and Distributed Computing on "Workstation Clusters and Network-based Computing". Currently, he is serving as a Vice-Chair of the ICPP '98 conference, an Associate Editor of IEEE Transactions on Parallel and Distributed Computing, an IEEE Distinguished Visitor Speaker, and an IEEE Chapters Tutorials Program Speaker. Dr. Panda is a 1995 recipient of the NSF Faculty Early CAREER Development Award and a 1997 recipient of the Lumley Research Award at the Ohio State University. He is a member of the IEEE Computer Society and the Association for Computing Machinery.