Proceedings of the IASTED International Conference
Parallel and Distributed Computing and Systems
November 3-6, 1999 in Cambridge Massachusetts, USA

A Cache Coherence Protocol for the Bidirectional Ring Based Multiprocessor*

Hitoshi Oi and N. Ranganathan
Dept. of Computer Science and Engineering,
University of South Florida,
Tampa, FL 33620.
Email: {oi,ranganat}@csee.usf.edu

Abstract

In this paper, a new cache protocol for ring based shared memory multiprocessors is discussed and analyzed. The proposed protocol uses multicasting of rings to reduce the message traversal length. The simulation results show that the proposed protocol with a bidirectional ring improved the system performance by 8% to 30% as compared to Barroso's protocol using a unidirectional ring. Assuming the bidirectional ring structure in both cases, the proposed protocol yields up to a 13% performance improvement over Barroso's protocol.

Keywords: Cache coherence protocol, shared memory multiprocessor, bidirectional ring network.

1 Introduction

Distributed shared memory (DSM) multiprocessors provide the convenience of globally shared memory space over the physically distributed memory modules. This convenience is realized by sending messages over the interconnection network that connects processing elements (PE). As a result, the design of the interconnection network has a significant impact on the performance of DSM multiprocessors.

Among various interconnection networks that have been used for multiprocessor systems, the ring networks have the advantages of (1) fixed node degree (modular expandability), (2) simple network interface structure (fast operation speed) and (3) low wiring complexity (fast transmission speed).

Shared memory multiprocessors using ring networks include KSR-1 of Kendall Square Research [3], NUMAchine of University of Toronto [2], and Scalable Coherent Interface (SCI) defined by the IEEE P1596 standard [4]. These systems are based on unidirectional rings; hence all the messages have to traverse all the way through the ring even if the destination is within the local neighborhood. The use of bidirectional rings can significantly reduce the message traversal and hence leads to a higher performance [5, 6]. In this paper, we propose a cache coherence protocol for bidirectional ring based multiprocessors, and evaluate its performance by comparing it to those of the existing protocols for ring multiprocessors. This paper is organized as follows: Section 2 describes the proposed protocol in detail; In Section 3, we discuss the advantages of the proposed protocol by comparing it to other protocols qualitatively; In Section 4, the performance of the proposed protocol is evaluated by execution-driven simulations; Some conclusions are provided in Section 5.

2 Proposed Coherence Protocol for Bidirectional Ring

The diagram in Figure 1 shows the possible states and transitions of a cache block in the proposed protocol. There are three basic states, *Invalid* (I), *Shared* (S) and *Exclusive* (E), and four transient states, *Read Pending* (RP), *Read Failed* (RF), *Write Pending* (WP) and *Write Failed* (WF). A cache block in I state does not exists in the cache because it has not been accessed or it has been invalidated by other PE. A cache block in S state can be accessed for reads, but not for writes. There may be more than one cached copies of a memory block in S state in a system. A cache block in E state can be accessed for both reads and writes. There exists only one cached copy of a memory block in E state in a system. Four transient states (RP, RF, WP, WF) are used to solve conflicts between concurrent accesses and will be explained further in Section 2.2.

In a DSM multiprocessor, each PE is assigned a part of global memory space in the unit of memory blocks and maintains directory entries for the memory blocks. Each memory block can have the following four states as shown in Figure 2: *Uncached* (UC), *Cached Clean* (CC), *Cached Dirty* (CD) and *Forward Pending* (FP). When no cached copy of the memory block exists (i. e. only the main memory has the data block) the memory block is in UC state. When

*This research is supported in part by a National Science Foundation Grant No. MIPS 9522265.

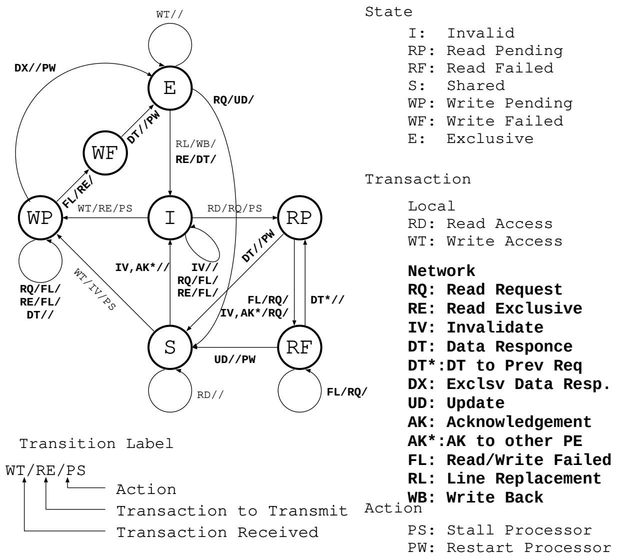

Figure 1: Cache Block State Transition Diagram

one or more unmodified cached copies exist, the block is in CC state. When a PE tries to modify its cached copy of the block, the state of the memory block moves to CD state. In this case, the content of the block in the main memory is invalid. The directory entry for the block records which PE has the modified copy of the block. A read access to a modified memory block is forwarded to the owner node. The memory block will be over written by the data sent from the owner node, and until this transaction is completed, the memory block is in FP state.

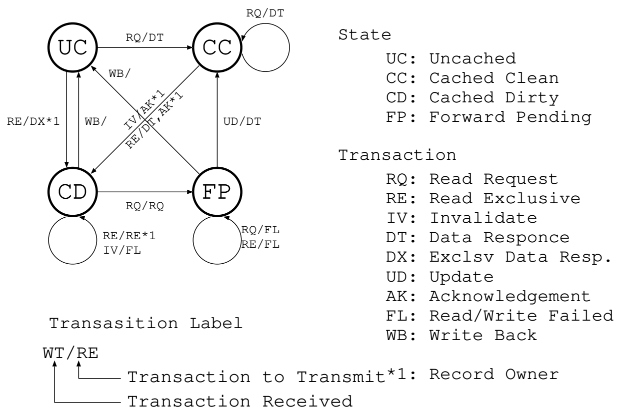

Figure 2: Memory Block State Transition Diagram

2.1 Basic Coherence Actions

Read Miss: On a read miss, the cache block changes its state to RP immediately and sends a RQ message to the block's home node. If the memory block is in either UC or CC state, the home node sends back a DT message to the requesting node and the state of the memory block becomes (or remains in) CC. Upon receipt of the DT message, the requesting node changes the state of the corresponding cache block to S. Successive read accesses to the same cache block result in cache hits. If the memory block is in CD, the home node forwards the request to the owner node of the dirty block, and it changes the memory block state to FP immediately. Upon receipt of the forwarded request from the home node, the owner node replies with a UD message and it changes its cache block state to S. This UD message is multicast and is received at both the home node and the requesting node. Upon receipt of the UD message, the home node changes its memory block state from FP to CC, and the requesting node changes its cache state from RP to S (*Shared*). Transactions in this mode of read miss are explained in comparison with other protocols in Section 3.

Write Miss: In this context, the term "write miss" stands for a write access to a block not existing in the cache. Thus, a copy of the accessed block needs to be brought into the cache and cached copies (if exist) in other PE's need to be invalidated. The requesting node sends an RE message to the home node, and it changes the cache block state from I to WP. The RE message is handled as a broadcast message at the network interface of each node on the path from the requester to the home; the RE message is taken into the cache controller and is also forwarded to the next node. If the memory block is in CC state, the home node sends a DT message to the requesting node as well as an AK message to the next node. Note that the DT message is sent through the shorter path to reduce the latency. However, the AK message has to traverse the longer path to the requesting node to invalidate the nodes that the RE message did not pass by. If the memory block is in CD state, the RE message is forwarded to the current owner of the block, and the owner of the memory block is updated with the requesting node. The current owner of the block sends a DX message to the requester, and changes the state of the block to I. When the requesting node receives DX message, it changes the state of the corresponding cache block to E and successive accesses (both reads and writes) to the same address result in cache hits. Each node through which the IV or AK (to the requesting node) passes changes the state of the corresponding cache block (if it exists) to I.

Invalidation: This transaction is initiated when a PE tries to modify a cache block in S state. The requesting node sends an IV message to the home, and it changes the state of the cache block to WP. At the home node, the state of the memory block is changed from CC to CD, and an AK message is returned to the requesting node through the path that the IV message did not traverse to reach the home node. Again, the IV and AK messages are handled as broadcast messages at the network interface of the nodes through which they pass: the state of the corresponding cache block at each node is changed from S to I. Upon the receipt of the AK message, the requesting node changes the state of the cache block to E. Successive accesses (both reads and writes) to the same address result in cache hits.

In the proposed protocol, sequential consistency is still provided because the write atomicity requirement [7], consisting of (1) serialization of writes to the same location and (2) invisibility of write until completion of invalidation of all copies, is observed.

Replacement: If a cache block in E state is selected for a replacement, it needs to be written back to the main memory. The node where the replacement occurs sends a WB message to the home node, and it changes the state of the cache block to I. When the home node receives the WB message, it changes the state of the memory block from CD to UC. Note that a cache block in S state is replaced without sending a WB message.

There is a possibility of a race due to a write back of a modified block. The home node does not know that the (previous) owner no longer has the modified data unless the home node receives the WB message. Therefore, a RQ or RE from another processor is forwarded to the previous owner node. When the RQ/RE message is received by the previous owner node, it sends an FL message to the requesting node. Upon the receipt of the FL message, the node that sent RQ/RE re-issues a RQ/RE message.

2.2 Conflict Resolution

Unlike a shared bus multiprocessor, multiple transactions to the same memory location can be traversing through the ring network at the same time. In this section, the proposed protocol's solution for the conflicts between concurrent accesses using the transient cache block states (RP, RF, WP and WF) and the memory block state (FP) will be described.

Read-Read Conflict: This combination of access conflict occurs when the target memory block is dirty (CD state). A RQ message arrives at the home node, and it is forwarded to the owner node. The home node changes the state of the memory block to FP. A Read-Read conflict occurs when the second RQ message arrives at the home node before the modified block is written back to the memory. The memory controller sends an FL message to the read-requesting node, and let it re-issue RQ message. An option of optimization could be to record the latter RQ message and reply with a DT message when the write back is completed. This option involves an additional cost of a more complex memory controller.

Read-Write Conflict: There are two possibilities for the conflict of this combination. The first possibility is that a write request (IV or RE) arrives at the home node of the requested block first, and then a read request (RQ) arrives at the home node. At the time of the RQ's arrival, the home node only knows that the ownership of the block is given to the writing node, and it does not know whether the invalidation of all the copies of the block has been finished or not. Thus the RQ message is forwarded to the writing node. The problem arises if the invalidation has not been completed when the forwarded RQ arrives at the writing node. The proposed protocol solves this conflict by sending an FL message to the reading node. Upon receipt of the FL message, the reading node changes the state of the block to RF and re-issues a RQ message. When the second RQ reaches the writing node, the write transaction must have been completed (if it is not completed, although very unlikely, re-issue of FL and RQ is repeated). Hence, the second RQ is handled the same as a read request to a dirty block. Another possibility is that the RQ arrives at the home node first and then an IV (or RE) arrives. Thus, the reading node may receive the AK* (acknowledgment to other PE's write) message before the DT that is in response to the previous RQ. Upon the receipt of the AK* message, the reading node changes the state of the cache block from RP to RF, and it re-issues a RQ message to the home node. When the reading node receives the DT message to the first read request, it changes the state from RF to RP. The second RQ message is just handled as a read request to a dirty node, and the corresponding DT message completes the second read request.

Write-Write Conflict: This conflict occurs when two or more nodes that have shared copies of the same memory block try to modify the data at the same time. These multiple write requests are serialized at the home node, and the write request that has arrived at the home node first is the winner; the node that issued the first write request will receive an AK message and it will proceed. Other nodes that tried to write to the same memory block will receive FL messages and have to re-issue RE messages. Note that the losers of these competing writes had cached copies of the block but these copies are considered to be invalidated by the IV or AK message issued by the winner of the conflicting write requests.

3 Comparison With Existing Protocols

Barroso and Dubois proposed three cache coherence protocols for ring based shared memory multiprocessors [1]. The proposed protocol is based on their full-map directory protocol. NUMAchine is also a ring based multiprocessor developed at University of Toronto [2]. The proposed protocol has mainly three advantages over these protocols for ring-based multiprocessors: (1) shorter read miss latency, (2) shorter write miss/invalidation latency and (3) less hardware overhead for directory entries. Below these advantages are explained by comparing the proposed protocol to other protocols, in terms of message traversals and hardware requirement.

The advantage of the proposed protocol over other protocols in terms of hardware overhead is as follows. Other protocols assumed a full-map directory entry to keep track of which nodes have copies of a memory block. In the proposed protocol, the multicast capability of the ring network is fully exploited and hence a full-map directory is eliminated. A directory entry in the proposed protocol needs to maintain the state of the memory block and the binary encoded ID of the owner node when the block is in CD state. Thus, the hardware overhead for directory entries can be reduced by $N/\lg N$, where N is the number of processing elements in the system.

Compared to Barroso's and NUMAchine's protocols, the proposed protocol shortens the message traversal length as follows. On the read miss to a modified block, a read request is forwarded to the node having the modified copy in its cache (owner node). The owner node changes the state of the cache block from E (*Exclusive*) to S (*Shared*) and sends a data message back to the home node. In Barroso's and NUMAchine protocols, the home node updates the memory block with the data sent from the (previous) owner, and then sends the data to the requesting node. In the proposed protocol, a multicast message is used if the requesting node is on the path from the owner node to the home node. This multicast message is received at both the requesting node and the home node. Consequently, the read miss latency is shorter and also the number of links traversed is reduced. When the home node is on the path from the owner node to the requesting node, the proposed protocol also uses a multicast message. This does not save the number of links traversed but can avoid the delay of the network interface and the memory state machine at the home node.

On a write access to a cache block with S (*Shared*) state in Barroso's and NUMAchine protocols, the requesting node first sends an invalidation request to the home node. Note that this invalidation request does not actually invalidate the nodes on the path from the requesting node to the home node. The home node sends an invalidation message that traverses the entire ring. In Barroso's protocol, when the invalidation returns to the home node, an acknowledgment is sent to the requesting node. Therefore, two entire ring traversals are needed (for a unidirectional ring). In NUMAchine protocol, the requesting node considers the write transaction is complete when the invalidation reaches at the requesting node. However, the invalidation still needs to traverse back to the home node. Thus, the total message traversal length is from the requesting node to the home node plus one entire ring. In the proposed protocol, the requesting node sends an invalidation message that actually invalidates the nodes on the path to the home node. The home node sends back to the requesting node an acknowledgment, that passes through the nodes that were not invalidated by the invalidation message. Thus, only one entire ring traversal is needed in the proposed protocol.

We use Barroso and Dubois' protocol and NUMAchine's protocol with slight modification in the later section for performance comparison.

4 Performance Evaluation

In this section, we evaluate the performance of the proposed protocol by comparing it to the existing protocols through execution-driven simulations. First, simulation environment, including system parameters, assumptions, and benchmark programs are described. Then, simulation results are presented.

4.1 Simulation Environment

For performance evaluation, we have developed an execution-driven simulator for a 32-processor system using Augmint multiprocessor toolkit [8]. Each processing node consists of an L1 cache, an L2 cache, a part of globally shared memory, and an interface to the ring network. The system parameters used in the simulations are shown in Table 1. The clock speed of the ring network is five times slower than that of the processor, and the link between adjacent nodes is four-stage pipelined (thus, the latency between adjacent nodes is 20 processor clock cycles). Slotted ring structure is assumed in both ring networks. The latency of the network interface that connects L2 cache and memory to the ring network is 10 processor clock cycles. In our simulation, we have chosen relatively slow timing parameters for ring network to emphasize the effect of network traversal and congestion. In addition, we assume that all the instruction accesses are L1 hit, and that private data accesses that have missed at L1 are L2 hits. Within four categories of cache misses (cold, capacity, conflict, and coherence), capacity miss and conflict miss are highly affected by the size and the degree of associativity of the cache. Also, the problem size of each SPLASH 2 benchmark program is intended to be small enough to simulate in a reasonable time [9]. Moreover, each benchmark program has different working set sizes. Therefore, rather than using some specific L2 cache size, we have decided to use an infinite size, full associative L2 cache.

To ensure equal bandwidths, the number of packets for the same type of message is doubled on the bidirectional

L1 Cache	Latency: 1, Size: 8KB, Block Size: 32B, Direct Mapped
L2 Cache	Latency: 8, Size: Infinite, Block Size: 64B, Full-Assoc.
Memory	Latency 30
Ring	Clock: 5, Node-to-node latency: 20, L2/Memory ↔ Ring latency: 10, 32 Nodes, Data link: 32bit/node

Table 1: System parameters. Timing parameters are represented by processor clock cycles.

Application	Sharing	Communication†
Lu	Repl	(see text)
Ocean	MigRW	NN
Radix	MigRW/MigR	ATA
Volrend	Repl/MigR	ATA
Water-N^2	Repl	ATA

†NN: Nearest-neighbor, ATA: All-to-all

Table 2: Benchmark programs and their sharing and communication patterns.

ring. A request message (a message without data) can be transmitted in a single packet on the unidirectional ring, while on the bidirectional ring it is divided into two packets. A data message is divided into sixteen packets on the unidirectional ring while it is divided into 32 packets on the bidirectional ring. Multiple packets of a message are transmitted in (possibly) non-consecutive slots and re-assembled at the destination node.

We use a set of parallel applications in Table 2 that have various sharing and communication patterns. These applications are from SPLASH 2 benchmark suits [9] and we use their default problem sizes. The sharing pattern of each program is also shown in the Table 2 using the classification in [10]. *Repl* is a sharing pattern in which a data structure is accessed by several processors in a read-mostly manner. Another sharing pattern is called migratory sharing, in which a data object is largely accessed by a single processor at a time. The migratory sharing is further divided into read-mostly (*MigR*) and read-write (*MigRW*).

The third column in Table 2 shows the communication patterns of the benchmark programs. *Ocean* has a nearest-neighbor communication pattern (NN in Table 2). 27% of remote misses are destined to the adjacent nodes. *Lu* has a very unique communication pattern. It has peaks at nodes whose link distances are multiple of four. All other applications exhibit more or less flat (all-to-all) communication patterns (ATA). The home node location of a memory block is decided by the page-level first-touch policy, where the page size is 4KByte. No attempt to optimize communication distance was made.

4.2 Simulation Results

Execution time of each application normalized to that of Barroso's protocol with unidirectional ring (referred to as "base case" of comparison hereafter) is shown in Figure 3.

The fraction of access time within execution time is relatively low for Lu. Although the bidirectional ring reduced the average read latency by 20%, the difference in execution time is small. Assuming the same protocol, bidirectional ring was about 9% faster than unidirectional ring. This difference in performance mainly came from the reduction of read access time. The differences among protocols assuming the same ring structure were small ($\leq 2\%$).

In *Ocean*, the proposed protocol was most effective. It had a very strong nearest-neighbor communication pattern, which was beneficial for a bidirectional ring. Also, 79% of read misses were to modified memory blocks, and 25% of those misses were optimized by the proposed protocol (as described in Section 3). Compared to the base case, the proposed protocol with bidirectional ring was 21% faster. When a bidirectional ring was used for all the protocols, the proposed protocol was faster than Barroso's and NUMAchine protocols by 13% and 8%, respectively. The performance difference between unidirectional and bidirectional rings was mainly due to the reduction of read access time.

Radix had very high remote miss rate (90% of L2 misses were remote), which emphasized the differences of interconnection networks. Especially, the average latency of write access was reduced by 45% using a bidirectional ring. Thus, the bidirectional ring was 26 to 28% faster than the unidirectional ring. On the other hand, only 10% of read misses were optimized by the proposed protocol. Also, 79% of write misses were to blocks owned by other L2 cache. In this mode of write miss, all the protocols operated in the same way. As a result, differences among protocols with the same ring were small ($\leq 3\%$).

Volrend has the all-to-all communication pattern. 88% of read misses and 66% of write misses were to clean memory blocks. Also, the fraction of read/write access time within execution time is the lowest among benchmark programs. Thus, the network traffic is also low and the differences among protocols as well as among rings used were small. The proposed protocol was only 8% faster than the base case and the differences among protocols with the same ring were less than 3%.

In *Water*, the fraction of write miss was relatively high (40%), which was advantageous for protocols with write optimization (NUMAchine and the proposed protocol). For read accesses, half of the misses were to modified memory blocks, and 42% of such misses were optimized by the proposed protocol with a bidirectional ring. The proposed protocol with a bidirectional ring was 15% faster than the base case. When all the protocols used a bidirectional ring, the proposed protocol was 7% and 4% faster than Barroso's and NUMAchine protocols, respectively.

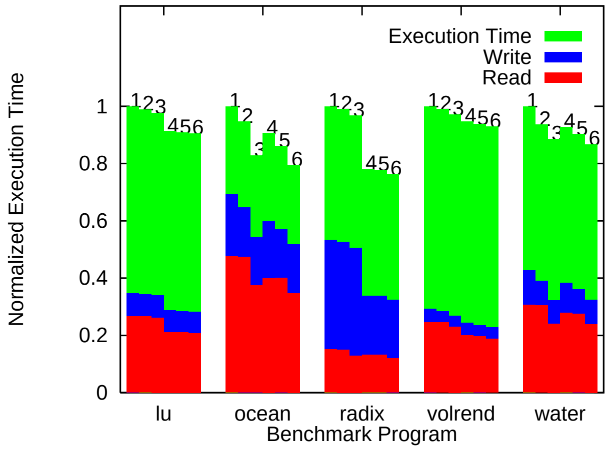

Figure 3: Normalized execution time. The label on each bar indicates a protocol/ring combination: 1: Barroso's protocol with a unidirectional ring (comparison base), 2: NUMAchine with a unidirectional ring, 3: proposed protocol with a unidirectional ring, 4: Barroso's protocol with a bidirectional ring, 5: NUMAchine with a bidirectional ring, 6: proposed protocol with a bidirectional ring.

5 Conclusions

In this paper, we proposed a cache coherence protocol for a bidirectional ring-based multiprocessor. In addition to the use of a bidirectional ring, the proposed protocol has the advantages of shorter ring traversal by exploiting the multicast capability of the ring network and less hardware overhead for memory directory entries. The results of execution-driven simulations showed that the proposed protocol with a bidirectional ring was up to 29% faster than the combination of Barroso's protocol with a unidirectional ring.

The topics of further investigation include incorporation of the latency hiding techniques such as prefetching [11] and relaxed consistency models [7]; an adaptive page migration scheme based on the access frequency and the number of links; extension of the proposed protocol for multilevel ring network; and the effect of system parameters (ring speed, cache size/speed, etc) on the performance.

References

[1] L. A. Barroso and M. Dubois, *Cache Coherence On A Slotted Ring*, in Proceedings of International Conference on Parallel Processing, vol. 1, 1230–1237, August 1991.

[2] Z. Vranesic et al, *The NUMAchine Multiprocessor*, Technical Report, Department of Electrical and Computer Engineering, Department of Computer Science, University of Toronto, June 1995.

[3] Kendall Square Research Corporation, *Technical Summary*, 1992.

[4] D. B. Gustavason, *Scalable Coherent Interface and Related Standards Projects*, IEEE MICRO, Vol. 12, No. 1, 10–22, February 1992.

[5] Hitoshi Oi and N. Ranganathan, "Performance Analysis of the Bidirectional Ring-Based Multiprocessor", in *Proceedings of ISCA 10th International Conference on Parallel & Distributed Computing Systems*, 397–400, October 1997.

[6] Hitoshi Oi and N. Ranganathan, "Effect of Message Length and Processor Speed on the Performance of the Bidirectional Ring-Based Multiprocessor", in *Proceedings of International Conference on Computer Design*, 267–272, Autstin, Texas, October 1997.

[7] S. V. Adve and K. Gharachorloo, *Shared Memory Consistency Models: A Tutorial*, WRL Research Report 95/7, Digital Equipment Corporation, September 1995.

[8] A-T. Nguyen, M. Michael, A. Sharma and J. Torrellas, "The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures", in *Proceedings of 1996 International Conference on Computer Design*, 486–490, October 1996.

[9] S. C. Woo et.al., "The SPLASH-2 Programs: Characterization and Methodological Considerations", in *Proceedings of the 22nd International Symposium on Computer Architecture*, 24–36, June 1995.

[10] Z. Zhang and J. Torrellas, "Reducing Remote Conflict Misses: NUMA with Remote Cache COMA", in *Proceedings of International Symposium on High Performance Computer Architecture*, 272–281, February 1997.

[11] Per Stenström, et. al., "Boosting the Performance of Shared Memory Multiprocessors", IEEE COMPUTER, Vol. 30, No. 7, 63–70, July 1997.