Appeared in Proc. Supercomputing 92, Minneapolis, MN USA, November 1992, pp. 348-357.

Cache Consistency in Hierarchical-Ring-Based Multiprocessors[†]

Keith Farkas Zvonko Vranesic Michael Stumm

Department of Electrical Engineering
University of Toronto
Toronto, Ontario, Canada M5S 1A4
email: farkas@eecg.toronto.edu

EECG TR-92-09-01

Abstract

A cache consistency scheme is presented for a class of multiprocessors based on a hierarchy of rings. By taking advantage of the natural broadcast and ordering properties of rings, cache consistency is achieved via a simple, selective-broadcast based protocol requiring no complex hardware. Using address-trace driven simulations of the Hector shared-memory multiprocessor, it is shown that the scheme performs well.

1 Introduction

The design of cache consistency protocols for large-scale shared-memory multiprocessors is complicated because of several factors. First, due to limited bandwidth of a single bus, large-scale multiprocessors have more complex interconnection networks. These networks use split-cycle protocols and allow concurrent memory accesses. In some cases these networks are not race-free [13], which makes it difficult to impose a global ordering on accesses.

The second complication pertains to the increased potential for contention at some of the system nodes such as network links or memory modules. Contention for such resources requires that requests for them be either queued in some fashion, or refused by returning a negative acknowledgment to the requester. Since using infinite queues can be impractical, the unsuccessful requesters may be allowed to retry the request later. As a result, consistency protocols must be capable of gracefully handling rejected requests and retries, as well as permitting the queuing of requests.

[†]To appear in "Supercomputing '92", November 1992.

Finally, growing memory sizes and increasing number of processors imposes limits on the scalability of consistency protocols. All protocols require some state information to be kept and the amount of this information increases with the number of processors and memory size. Moreover, a particular data item may be shared by a large number of processors, all of which must be notified when a change to the data item is made. This notification process can inflict a large delay on the source of the update and can consume a significant portion of the available network bandwidth, depending on the implementation details of the protocol.

A number of cache consistency protocols have been proposed, all of which address the above three complications in various ways. *Limited-map* directory schemes attempt to address the issue of bandwidth consumption by limiting the broadcasts of cache control messages to only those processors which have a copy of the accessed item, while at the same time reducing the amount of state information. The MIT Alewife multiprocessor [3] implements a version of such a protocol in which the broadcasting is handled by the hardware if the degree of sharing is small and otherwise by software. In using this hardware/software approach, the poor scalability of limited-directory schemes is avoided [3]. A related approach is implemented by the DASH multiprocessor [14], which uses a *full-map* directory but the amount of state information is reduced by grouping processors into clusters. Consistency within a cluster is maintained via snooping.

A third cache-consistency scheme, implemented by the IEEE SCI protocol [9], uses linked lists as opposed to the directories used in the DASH and Alewife systems. The linked lists, maintained by pointers in each

cache block frame, are used to identify the nodes with a copy of a given data item. Consistency is maintained by traversing the list anytime this data item is modified. Traversing a list, however, can result in messages flowing over the same network links several times, especially in networks without point-to-point interconnections between all modules. For example, in a ring-connected system a message may have to traverse the entire ring n times in the worst case, if there are n active copies of the data to be invalidated. Barroso and Dubois [1] have proposed a scheme for a system of processors interconnected by a unidirectional ring that relies on snooping and thus avoids the multiple-traversal problem of the SCI protocol.

In this paper, we propose a selective-broadcast based cache consistency protocol that addresses the three complications listed above for a class of multiprocessors based on hierarchical rings. Ring-based networks have been investigated [1, 6, 10, 11, 16] as a means for implementing high performance interconnection backplanes because they offer a number of advantages. Having point-to-point interconnections, large rings can be driven at very high clock rates. Rings also exhibit natural broadcast and ordering properties that facilitate the implementation of cache consistency protocols. The proposed protocol can easily be used to achieve various consistency models, including sequential consistency [12] and processor consistency [8].

In the next section, we define the class of machines for which the protocol is targeted. Section 3 presents the new protocol. Section 4 discusses several performance issues and enhancements to the basic protocol. Finally, in Section 5, we analyze the performance of the protocol through address trace driven simulations.

2 The Architecture of the Target Multiprocessor

The protocol presented in this paper has been developed for general multiprocessors that are based on hierarchical rings. In order to clarify the presentation, we will describe without loss of generality the protocol as it would apply to the Hector multiprocessor architecture [16].

The target architecture consists of clusters of processor and memory modules, interconnected by rings. In Hector, a cluster is called a *station*, within which a split-cycle bus is used as the interconnection medium. The hierarchical multiprocessor is formed by interconnecting sets of stations by *local* rings, which are then interconnected by higher level rings. While the proposed scheme is scalable to an arbitrary number of levels, for simplicity we will assume the two-level ring hierarchy shown in Figure 1, comprising local rings interconnected by a single *central* ring.

Each memory module occupies a unique contiguous portion of a flat, global (physical) address space. The processor modules can transparently access all memory locations. The modules consist of a processor, cache memory, a cache controller and a communication submodule.

Information is transferred between processor and memory modules using a packetized synchronous transfer protocol. The interconnection network traffic consists of request packets and response packets. Communication submodules associated with each processor and memory module handle the sending and receiving of the packets. The transfer of packets between modules is managed by *station controllers* and *inter-ring interfaces*. Each station controller is responsible for controlling on-station transfers as well as the traffic on the local ring in the vicinity of its station.

Each ring can be thought of as consisting of multiple *segments* in which, in any given cycle, a single packet may reside. Packets travel around the ring by being synchronously transferred from one ring segment to the next; packets are assumed to travel in the counter-clockwise direction. As long as a packet on a particular segment is not destined for the associated station, on-station and local ring transfers may occur concurrently. If a ring packet is to be delivered to a module on a station, its delivery takes precedence over on-station transfers. Packets destined for another station are switched onto the station bus and local ring if the local ring segment does not contain a valid packet.

The inter-ring interfaces require FIFO buffers to store packets due to the possibility of packet collisions. A collision will occur if in a given cycle, input packets from both rings are to be routed to the same output. A simple interface is shown in Figure 2. As with the delivery of packets to a station from a local ring, packets on the central ring have priority over those on the local ring.

3 The Cache-Consistency Protocol

In this section, we present a consistency protocol in the context of the target multiprocessor. The protocol exploits the broadcast capability of rings and uses snooping within stations to maintain the caches consistent. Packet filters within each network node

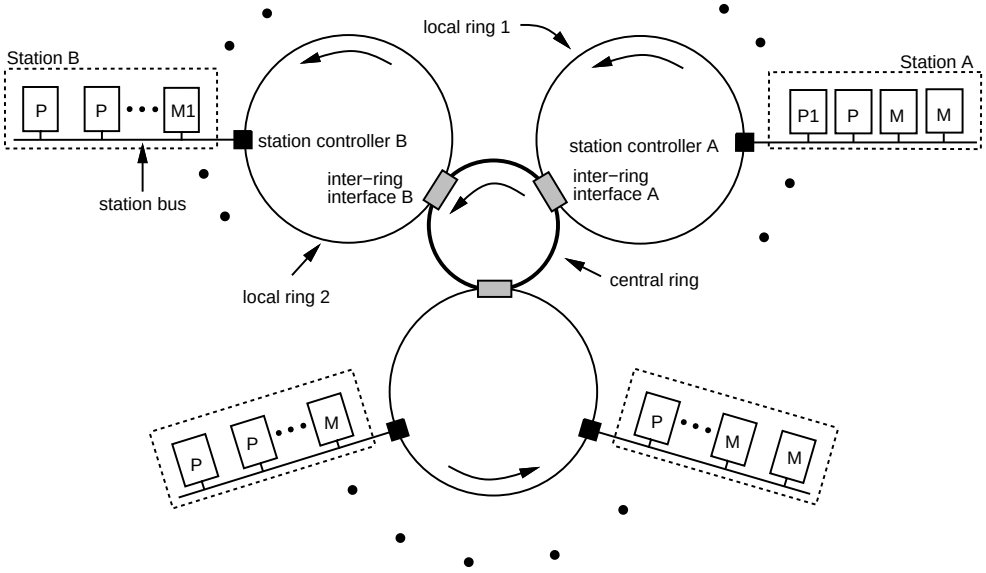


Figure 1: Structure of the Target Multiprocessor

are used to limit the scope of the broadcast messages and thereby improve the scalability of the protocol. The presentation describes the protocol in the target system with no packet filtering and with *invalidating* caches that use a write-through policy; packet filters and alternative cache strategies are then discussed in Section 4. Cache consistency is maintained whenever shared data is modified by writing the change through to main memory and invalidating all cached copies of the data. It should be noted that reads of shared data require no consistency actions because they are processed in the same manner as reads of private data. Moreover, it is assumed that the issuing processor is blocked until the requested data is returned.

A write of shared-data begins with the source processor module sending a *write-request* packet to the destination memory module, say memory module M1 in Figure 1. If the destination cannot accept the packet, a *write-nack* packet is returned to the source and the source will later reissue the write request. If the destination can accept the write-request packet,

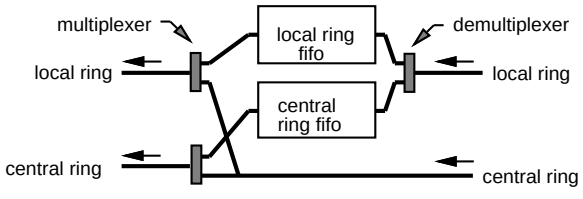


Figure 2: Inter-ring Interface

the addressed location is locked, and a *write-invalidate* (WI) packet is formed and broadcast to all processor modules. The broadcast is performed by first propagating the WI packet to the highest-level ring (i.e., the central ring in Figure 1), where it circulates around the entire ring and is switched back onto the destination's local ring (local ring 2 in Figure 1). At each inter-ring interface, a copy of the packet is made and passed down one level where it continues to propagate via the lower-level rings to all stations. When the WI packet returns to memory module M1, the updated location is unlocked; when the WI packet arrives for the third time at inter-ring interface B, it is removed.

From the time a memory location is locked until it is unlocked, other accesses to the same location are not accepted. This locking of the memory location prevents two processors from observing updates to two different memory locations in different orders. This ordering requirement is necessary to ensure sequential consistency (discussed later in this section), although it is not necessary for some weaker forms of consistency. In the appendix we illustrate how this locking enforces the necessary ordering for sequential consistency.

To illustrate the protocol in more detail, consider the following example. Assume that in Figure 1, processor module P1 issues a write to memory module M1 located on another station. Then, P1 is blocked from making further requests until it receives an acknowledgment from the destination. The write-request packet travels to the destination via local ring 1, the

central ring, local ring 2, and station bus B. If the destination memory module accepts the write-request packet, it forms a WI packet. This packet then visits each station in the system using the broadcast scheme described above. As the WI packet flows around the local rings, each station controller switches a copy of the packet onto its station bus to allow snooping and invalidation by the on-station caches. The only exception occurs at the source processor module where no invalidation occurs. Instead, the source is unblocked and as far as it is concerned the access is completed. When the WI packet returns to station B, the destination memory module M1 notes the return of the WI packet and unlocks the location to which this WI packet corresponds.
It is important to note the following details concerning the invalidation process:

- The WI packet on a local ring is removed by the associated inter-ring interface. The WI packet on the central ring is removed by the inter-ring interface through which it entered the central ring.

- The WI packet returns to the destination only after it has circulated around the entire central ring. The unlocking of the memory location, however, may occur before all cached copies of the corresponding data have been invalidated.

- The source of the write request is unblocked when the WI packet visits its station rather than when the memory location is unlocked.

- Any copies of the data item resident on the station on which the source is located are invalidated when the WI packet visits that station.

In summary, the invalidating protocol entails two distinct phases:

1. In phase one, the write request is sent to the destination and a decision is made whether or not to accept the request. Until the request is accepted by the destination, only the source is aware of the existence of the pending write.

2. If the request is accepted, the second phase is begun with the formation of a WI packet. This packet is broadcast to all modules to effect both the invalidation of cached copies of the location and the unblocking of the source. As far as the destination is concerned, this phase ends with the WI packet "visiting" the destination's station resulting in the unlocking of the memory location.

Sequential Consistency

It is essential to be able to demonstrate that a cache-consistency protocol correctly enforces the desired memory model. To do so entails showing that the necessary ordering of shared-memory accesses is preserved. Methods for demonstrating consistency have been put forth by a number of researchers [4, 7, 13, 15], where a set of memory-model specific conditions are specified that must be met by each processor in the multiprocessor system. If it can be shown that each processor adheres to these conditions, then the system correctly enforces the desired memory model. Using the conditions proposed by Scheurich, it can be shown that the proposed invalidation protocol enforces sequential consistency [5]. Three features of the Hector architecture guarantee that the necessary ordering of shared accesses is preserved. These features are: (1) there is a unique path between any two modules; (2) it is impossible for two packets to overtake each other; and (3) packets are processed at each network node in the order in which they arrive.

4 Extensions and Enhancements

This section discusses several performance issues and enhancements to the invalidating protocol that was presented in the previous section.

4.1 Alternative Cache Strategies

We have described a consistency protocol for invalidating caches. This protocol can also be extended to work with updating caches, but now the cache lines being updated must be locked during the time that the copies are being updated. To implement this locking, the protocol is augmented by a third phase. In the second phase of the protocol, the cached copies are updated and are also locked, thus preventing read or write accesses to the data item. Then, in the third phase, all copies are unlocked [5]. As in the case of the invalidation protocol, the locking of the cache lines is required to prevent two processors from observing updates to different locations in different orders.

The proposed invalidating protocol can be extended for use with the copy-back cache policy, but at the cost of more complex hardware. This policy requires first detecting that an access to an item that is dirty has been made and second guaranteeing that for a read request, the requester gets a copy of the valid data; the difficulties with this approach are discussed in more detail in [5]. For single bus-based systems,

these two requirements are easily accommodated as all data transfers are simultaneously visible to all processor modules. However, for systems in which multiple data transfers can occur concurrently, but invisibly to some of the processor modules, these two requirements are more difficult to meet.

4.2 Relaxing the Consistency Model

While the sequential consistency memory model is conceptually simple, it imposes restrictions on the permissible outstanding memory accesses of a processor. In so doing, it prevents many hardware optimizations that could increase system performance. For these reasons, weaker memory models have been considered. One such model is the *processor consistency* model [8] provided by several commercial multiprocessors, including the VAX 8800 and the Silicon Graphics POWER Station [7] which both employ a single bus thus rendering the consistency protocol simpler that the one we propose.

The processor consistency memory model stipulates that the write operations issued by a processor be observed in the order in which they were issued, but the writes issued by different processors may be observed in different orders. It is in this last point that the sequential consistency and processor consistency memory models differ. The protocol for processor consistency is very similar to the invalidating protocol, described in Section 3, but it does not require the memory to be locked during the second phase.

4.3 Scalability

Scalability of the proposed protocol is predominantly influenced by the bandwidth consumed by consistency messages which are broadcast to all processor modules. As the number of processors in the system increases, the broadcast traffic will also increase correspondingly. To prevent the broadcast traffic from swamping the interconnection network, it is necessary to either limit the number or the scope of the broadcasts.

An attractive possibility is to use a filter mechanism. Filters are located at each node in the ring-hierarchy to limit the propagation of the write-invalidate packets to only those sections in the system where a cached copy of the data exists. Each station controller and inter-ring interface has two filters, one to restrict broadcast packets from going farther up in the hierarchy, the other to restrict broadcast packets from entering the subsystem below. The hardware costs of such filters can be reduced by providing the filtering on a per page basis (as opposed to on a per cache line basis). With the page-based granularity, it is easy for the operating system to manage these filters along with the page tables it already must manage.

5 Evaluation of the Proposed Protocol

Using address driven simulations, we have investigated the effects of the proposed invalidating protocol on the performance of the target system. Because a meaningful evaluation demands that a detailed simulation model be employed, we decided to model the Hector multiprocessor at the register level. The Hector multiprocessor [16] is similar to the target system, with the most important differences being that in Hector the memory is distributed among the processor modules instead of residing in separate modules. Because of the differences, the invalidating protocol described in Section 3 had to be slightly modified. These modifications and the simulator are described in [5].

5.1 Simulation Methodology

A number of different system topologies were simulated for 32 processor and 64 processor systems. In each system, processor modules comprised 64-Kbyte instruction and 64-Kbyte data caches, a processor, and 16 Mbytes of the global memory. To reduce the number of lock bits required for locking memory locations during phase two of the invalidating protocol, the locking was done on a per (physical) page basis[1]. Packet filtering was employed in the 64 processor systems.

The processor was modeled using an event generator to emulate the execution of a number of multiprocessor applications. We will show the results for four of these applications running on a 32 processor system and the results for a fifth running on a 64 processor system. This latter application, SOR[2], is an iterative method for solving partial differential equations. The simulation of this application involved 5 million iterations over the array, during which the memory references and inter-reference timings were generated using a state machine.

For the other four applications, the memory references were obtained from address traces that, as shown in Table 1, exhibit a varied distribution of memory operations. The first three of these are part of the SPLASH parallel benchmark set of traces that is available from Stanford University; a description of

[1] The contention for these locks is shown in the next section.

[2] Successive Over-Relaxation

Application	Number of Memory References	Distribution of Memory Operations					
		% Instructions	% Atomic Operations	% Private Data		% Shared Data	
				Reads	Writes	Reads	Writes
LocusRoute	7.7 M	51.4	0.0	32.5	11.3	4.2	0.6
SA-TSP	7.1 M	46.5	0.0	29.1	5.0	18.3	1.1
PTHOR	7.1 M	49.7	0.0	25.4	9.6	14.0	1.3
Speech	4.7 M	–	–	–	–	78.2	21.8

Table 1: Address Trace Characteristics. LocusRoute is a global router for VLSI standard cells, SA-TSP solves the traveling salesman problem using simulated annealing, PTHOR is a parallel logic simulator and Speech implements the lexical decoding stage of a speech interpretation language.

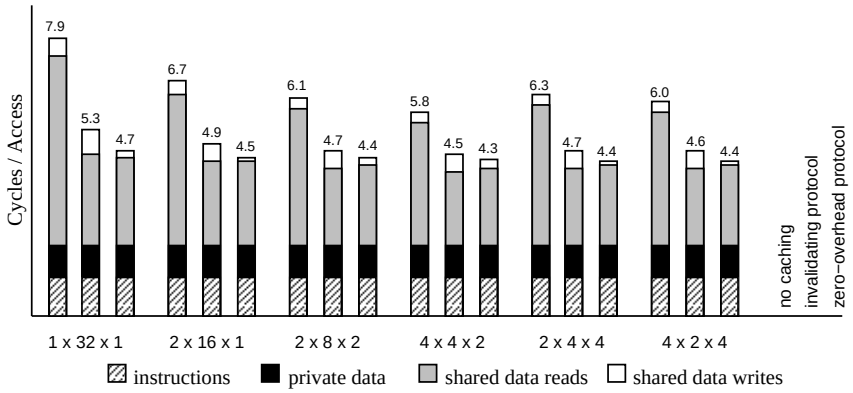


Figure 3: Average memory latency for the concurrent execution of the SA-TSP and LocusRoute applications, measured at the time the first processor completed its task.

the applications and the methods used to acquire the traces was presented by Weber and Gupta [17]. A description of the fourth application and the method used to acquire the trace was presented by Chaiken et al. [2]. This is a pure data trace that contains only shared-data accesses.

To investigate the performance impact of the invalidating protocol on the run-time behavior of the benchmarks, three different scenarios were simulated: (1) shared data is not cached; (2) shared data is cached using the proposed invalidating protocol; and (3) shared data is cached using a zero-cycle overhead cache consistency scheme. The last scenario, in which all copies of a location are assumed to be invalidated in a single cycle with no messages transmitted, is provided to gauge the overhead due to the use of the invalidating protocol.

For each system size, six different system topologies were simulated. However, for the 32-processor system, since the SPLASH address traces include references for only 16 processors, we chose to simulate the concurrent execution of two SPLASH applications. The Speech address trace contains references for each of the 32 processors, so concurrent execution of it was not simulated. Various memory page distribution schemes were implemented, all of which gave virtually identical results. For the results presented below, round-robin distribution was used to evenly distribute the pages across all processor modules.

Because the address traces listed in Table 1 were acquired from machines dissimilar to Hector, it is only meaningful to compare the results for the different caching strategies and topologies; the absolute numbers are not meaningful. In addition, the traces contain only memory references with no inter-reference timing information[3]. Also lacking from the traces is sufficient information to allow the efficient mapping of the address streams to the processors so as to guarantee that cooperating processors are located physically close to each other. A similar comment applies to the virtual to physical page allocation. Hence, the results presented are pessimistic.

5.2 Results

Figure 3 presents the average memory access latency for the concurrent execution of the SA-TSP and LocusRoute applications. It shows the latency attributable to each access type. The ordered triples labeling the horizontal axis specify the topology simulated: the first coordinate indicates the number of processors per station, the second the number of stations per local ring and the third the number of local rings. Because the cache hit rates for instruction and private-data accesses were very high, the contribution to the average latency by these access types is constant.

Figure 4 presents similar results for the concurrent execution of two instances of the PTHOR application. Both Figures 3 and 4 show that the average latency is reduced with a cache-consistency scheme and that the invalidating protocol performs within 20% of the ideal zero-cycle overhead scheme. However, the zero-cycle overhead scheme performs much better than the invalidating protocol for the Speech application, as shown in Figure 5. This result is not surprising since the Speech application essentially contains only shared-data accesses.

The difference in performance between the proposed cache consistency scheme and the zero-overhead scheme is due to the broadcast traffic (which is zero in the zero-overhead case). Another reason for this difference is the locking of the memory location during the second phase of the protocol. The effects of locking can be diminished by locking at a finer granularity than on a per page basis. However, it should be noted that finer-grain locking is not necessary for many applications. As seen in Figure 6, the percentage of unsuccessful shared-data accesses due to a locked memory page was low in our simulation runs. The dependence on topology exhibited by the unsuccessful-access rates is attributable to the length of time that a given location is locked while waiting for the return of the WI packet. In the single-ring topologies, this time is equivalent to the time required to traverse the ring once. On the other hand, for the multiple-ring topologies, this time will be less than the time for the single-ring topologies due to the hierarchical broadcast scheme.

The results in Figures 3 to 5 are based on hardware configurations that do not make use of the filter mechanism discussed in Section 4.3. It is apparent that in a Hector machine with up to 32 processors, the utilization of the interconnection network is low enough so that broadcast traffic generated by the proposed cache consistency scheme will not have a significant impact on normal memory accesses. However, as the number of processors increases, there will be much more broadcast traffic, resulting in poor performance. Our simulations have shown that with 64 processors the broadcast traffic may dominate to the extent that the interconnection network approaches saturation. This phenomenon was observed in our simulations of the

[3] In the simulations, a one cycle delay between memory references was assumed.

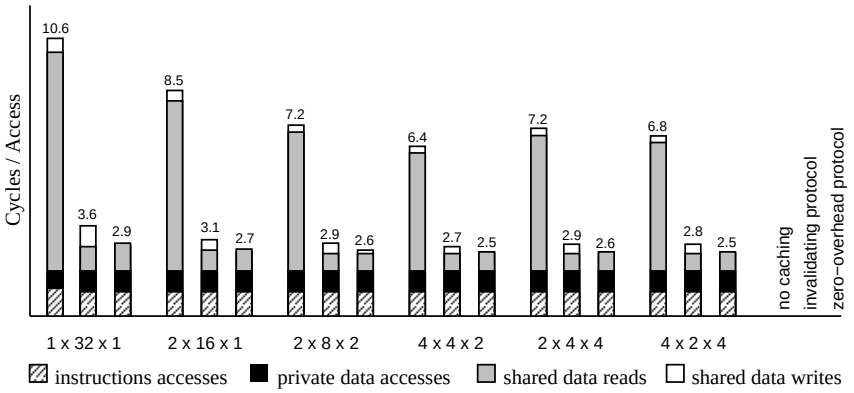


Figure 4: Memory latency for the concurrent execution of two instances of the PTHOR application, measured at the time the last processor completed its task.

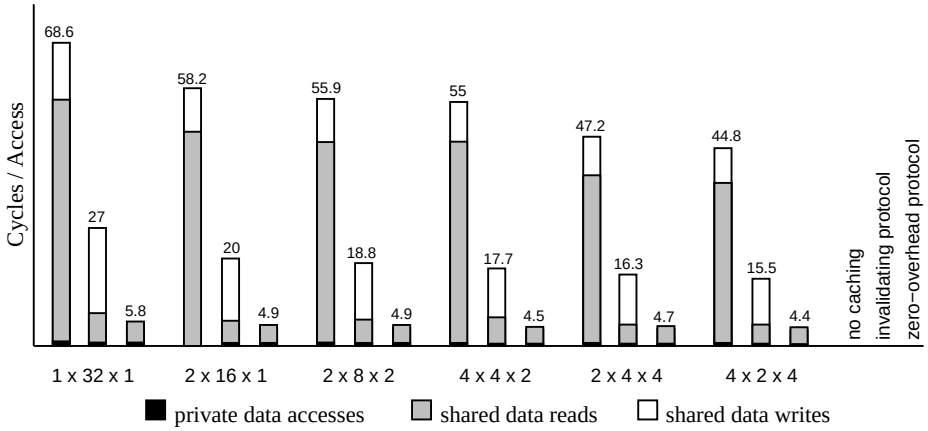
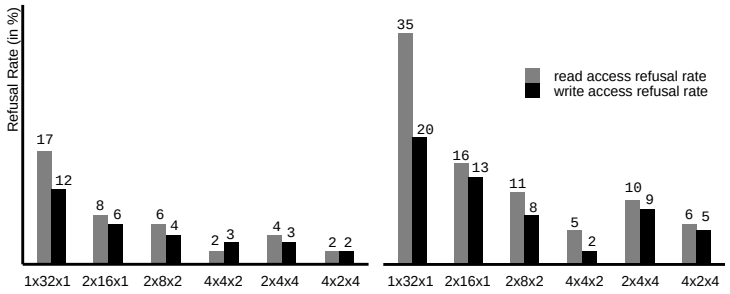


Figure 5: Memory latency for the execution of the Speech application, measured at the time the last processor completed its task.



(a) Concurrent execution of SA-TSP and LocusRoute Applications

(b) Speech Application

Figure 6: Unsuccessful shared-data accesses due to a locked memory page.

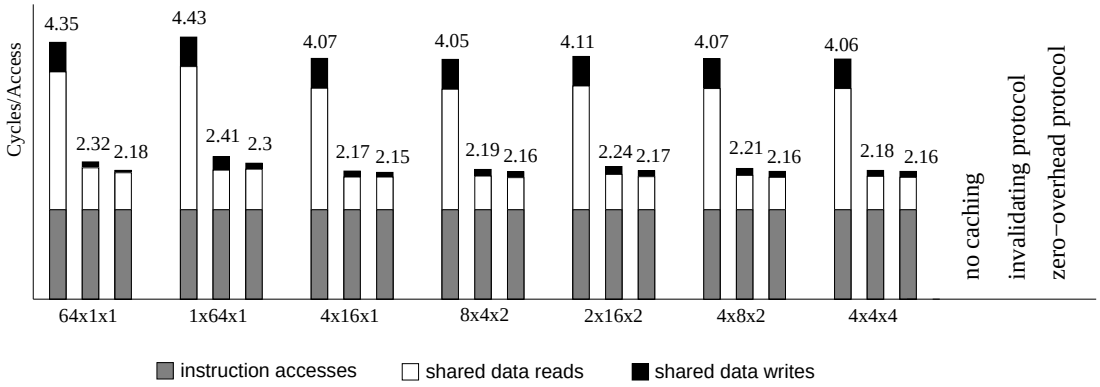


Figure 7: Average memory latency for the SOR application with the use of filters to limit the scope of invalidation packets, for a 64 processor system. In this application, 75% of memory accesses are instruction fetches, 20% are shared-data reads and 5% are shared-data writes.

SOR application. While this application is characterized by a large number of writes, any part of shared data is accessed by at most two processors. Without the filtering mechanism on a 64-processor machine, this application showed very poor performance, reaching a point where it would have been better not to use caching at all. Using a filter on the same machine improves the performance dramatically. As shown in Figure 7, the performance obtained is close to the zero-overhead case.

Finally, it is interesting to note that the results presented in this section suggest that the topology of the multiprocessor machine has a significant impact on performance. In the figures, the best topologies are those that are balanced in terms of the number of processors, stations and rings.

6 Conclusions

A key characteristic of cache consistency protocols is the one-to-many relationship between a shared-data update and the resulting consistency messages that are directed to all copies of the updated location. Thus, while cache consistency protocols seek to the reduce access latency for shared data, their use may degrade the overall system performance by increasing the interconnection network utilization, and contention for other system resources. Owing to this one-to-many relationship, no consistency protocol can be truly scalable. Nevertheless, a high degree of scalability can be achieved especially if the protocol seeks to minimize the number and frequency of the consistency messages without imposing additional costs on shared data accesses.

In this paper, we have presented such a cache consistency protocol that is targeted for shared-memory multiprocessors consisting of processor and memory modules interconnected by a hierarchy of ring-connected buses. By making use of two key features of the architecture, the scalability of the protocol is greatly enhanced. First, the hierarchical nature of the interconnection network offers a simple way to implement a packet filtering mechanism. And secondly, due to the natural broadcast property of rings, the number of consistency messages appearing in the network is usually far less than the number of cached copies. That is, it is not necessary to send individual messages to all processors with cached copies of the location. The results presented in Section 5.2 show that such a scheme noticeably improves the performance of the system.

Finally, the cache consistency scheme presented enforces sequential consistency with simple hardware and protocols, and it is easily extendible to systems employing updating caches or a less strict processor consistency memory model.

Appendix: Enforcing Sequential Consistency

Sequential consistency requires the imposition of a system-wide order on all accesses to shared memory locations. In multiprocessors, updates issued to shared-memory locations are said to occur when they are observed by the other processors. Thus, of concern is the order in which updates are observed rather

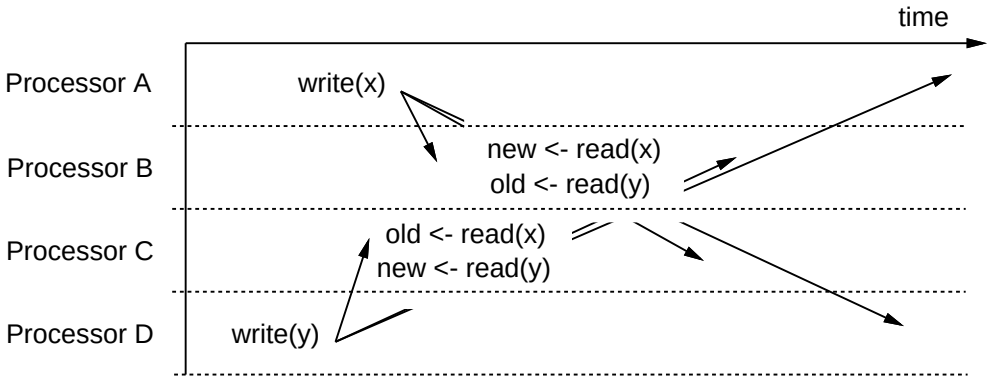


Figure 8: This diagram illustrates how two processors B and C might observe updates to two different memory locations in different orders when the locations being updated are not locked. The directed line segments represent the write-invalidate packets. It is assumed that processors A and B are located close together in the network, as are C and D.

than the system-wide order in which they are issued. For sequential consistency, the observed order must be the same for all processors. The multiphase protocols discussed in this paper enforce this requirement by locking updated memory locations during the invalidation (updating) phase.

We will illustrate by an example why locking is necessary to impose a global ordering on the updates. A more formal discussion on locking and on the requirements for sequential consistency is given in [5]. Invalidating caches are assumed, although the discussion is also applicable to updating caches.

Consider four processors A,B,C and D which share two locations X and Y. Assume that processor A is close to B in the network, and processor C is close to D. Each processor initially has a cached copy of both locations. Then, suppose that processor A updates location X, processor D updates location Y and processors B and C read both locations. The absence of locking can lead to a non-global ordering as illustrated in Figure 8. In this figure, the directed line segments represent the write-invalidate packets resulting from the updates to X and Y. Processor C issues a read of location Y, and because its cached copy has been invalidated, the new value is acquired from the main memory. Processor C then issues a read of location X, and receives the old value of X since processor C's cached copy has not yet been invalidated. A similar scenario is possible for processor B resulting in it acquiring the new value of location X and the old value of Y. Clearly the update orderings observed by B and C are not the same, and thus a global ordering does not exist.

With locking, processor C will not be able to acquire the new copy of location Y before processor B's copy of Y is invalidated. Similarly, processor B cannot acquire the new copy of location X before Processor C's. Thus, it is impossible for Processors B and C to observe the updates to X and Y in different orders. This example demonstrates why the locking of memory locations prevents updates from being seen out of order and thereby preserves sequential consistency.

References

[1] Luiz Barroso and Michel Dubois. Cache coherence on a slotted ring. *Proc. of the International Conference on Parallel Processing*, 1 (Architecture):230–237, 1991.

[2] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal. Directory-based cache coherence in large-scale multiprocessors. *Computer*, 23(6):49–58, 1990.

[3] D. Chaiken, J. Kubiatowicz, and A. Agarwal. Limitless directories: A scalable cache coherence scheme. *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 224–234, 1991.

[4] Michel Dubois, Christoph Scheurich, and Fayé A. Briggs. Memory access buffering in multiprocessors. *Proc. of the 13th Annual International Symposium on Computer Architecture*, pages 434–442, 1986.

[5] Keith I. Farkas. A decentralized hierarchical cache-consistency scheme for shared-memory multiprocessors. Master's thesis, University of Toronto, April 1991. April.

[6] M. Ferrante. Cyberplus and map v interprocessor communications for parallel and array processor systems. *Proc. of Third Conference on Multiprocessors and Array Processors*, pages 45–54, 1987.

[7] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. *Proc. of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, 1990.

[8] James Goodman. Cache consistency and sequential consistency. Technical Report 61, SCI Committee, 1989.

[9] D.B. Gustavson. The scalable coherent interface and related standards projects. *IEEE Micro*, 12(1):10–22, 1992.

[10] Robert Halstead, Jr., Thomas I. Anderson, Randy B. Osborne, and Thomas L. Sterling. Concert: Design of a multiprocessor development system. *Proc. of the 13th Annual International Symposium on Computer Architecture*, pages 40–48, 1986.

[11] David V. James, Anthony T. Laundrie, Stein Gjessing, and Gurindar S. Sohi. Scalable coherent interface. *Computer*, 23(6):74–77, June 1990.

[12] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, c-28(9):690–691, Sep 1979.

[13] A. Landin, E. Hagersten, and S. Haridi. Race-free interconnection networks and multiprocessor consistency. *Proc. of the 18th Annual International Symposium on Computer Architecture*, pages 106–115, 1991.

[14] A. D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. Directory-based cache coherence protocol for the DASH multiprocessor. *Proc. of the 17th Annual International Symposium on Computer Architecture*, pages 148–158, 1990.

[15] Christoph Ernst Scheurich. *Access Ordering and Coherence in Shared Memory Multiprocessors.* PhD thesis, University of Southern California, May 1989. Tech Report no. CENG 89-19.

[16] Zvonko G. Vranesic, Michael Stumm, David M. Lewis, and Ron White. Hector: A hierarchically structured shared-memory multiprocessor. *Computer*, 24(1):72–79, Jan 1991.

[17] Wolf-Dietrich Weber and Anoop Gupta. Analysis of cache invalidations patterns in multiprocessors. *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 243–255, 1989.