The Journal of Supercomputing, 8, 345–369 (1995)

Scalable Cache Consistency for Hierarchically Structured Multiprocessors

KEITH FARKAS farkas@eecg.toronto.edu
ZVONKO VRANESIC zvonko@eecg.toronto.edu
MICHAEL STUMM stumm@eecg.toronto.edu
Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada M5S 1A4

(An earlier version of this paper was presented at Supercomputing '92.
Received March 1993; final version accepted July 1994.)

Abstract. This paper presents a new cache consistency scheme for hierarchically structured shared-memory multiprocessors. The scheme is simple, fast and efficient, and it does not require a large amount of state information to be maintained. The scheme exploits the broadcast capability of these systems, but limits the extent of the broadcasts by means of a novel filtering mechanism. As a specific example, it is shown how the proposed cache consistency scheme can be implemented on the Hector multiprocessor architecture. Using trace-driven simulations, we demonstrate that the scheme is scalable and performs well for common applications.

Keywords: Cache consistency, shared-memory multiprocessors, hierarchically structured multiprocessors, limited broadcast

1. Introduction

In shared-memory multiprocessors, the performance of parallel applications can be improved through the use of hardware-enforced cache consistency schemes. Yet hardware-based schemes do not scale easily to large systems. Large multiprocessors require interconnection networks that are more complex than the single-bus-like structures used in small multiprocessors. In these smaller systems, there exists a natural broadcast mechanism allowing for simple snooping-based cache consistency protocols. In larger systems, the networks are often segmented in order to increase the total system bandwidth. Extensive use of broadcasts in these segmented networks quickly increases the network utilization to an unacceptable level as the size of the system increases. Moreover, such networks are often not race-free, making it difficult to impose a global ordering on memory accesses. For these reasons, many of the cache consistency schemes proposed for large systems do not rely on broadcasts, but on communication that targets only the caches with copies of the particular block of memory.

In order to be able to target specific caches, it is necessary to store the identity of all caches that have a copy of a particular block. The hardware cost of maintaining this information increases greatly with both the memory size and the number of processors. Hardware complexity is further increased because the state information must be quickly accessible in order to avoid degrading the system performance.

A number of cache consistency protocols have been proposed that address the above issues, with the aim of achieving a high degree of scalability. Directory schemes attempt to minimize the bandwidth consumption by sending the cache control messages to only those processors that have a copy of the accessed item. The MIT Alewife multiprocessor [Chaiken et al. 1991] implements a version of such a protocol in which the message distribution is handled by the hardware if the degree of sharing is small and otherwise by software. By relying on software to enforce consistency when the degree of sharing is large, which is a relatively infrequent situation [Chaiken et al. 1991; Lenoski et al. 1992], the amount of state information maintained by the hardware remains small. The DASH multiprocessor [Lenoski et al. 1990] reduces the amount of state information required by grouping processors into clusters and maintaining the information on a per-cluster basis. Consistency within a cluster is maintained by means of snooping.

Another cache consistency scheme, which is implemented by the IEEE SCI protocol [Gustavson 1992], uses linked lists. A linked list, maintained by pointers in each cache block frame, is 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. In systems without point-to-point interconnections between all processors, it is possible that traversing a list can result in messages flowing over the same network links several times. 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. In systems based on a single unidirectional ring, this multiple-traversal problem can be avoided through the use of a snooping-based scheme proposed by Barroso and Dubois [1991].

In this paper we describe a new cache consistency scheme for hierarchically structured shared-memory multiprocessors that is simple, fast and efficient, and it does not require a large amount of state information to be maintained. It can be used to achieve various consistency models from sequential consistency [Lamport 1979] to release consistency [Gharachorloo et al. 1990]. It is suitable for hierarchical multiprocessors that employ interconnection networks with the following properties: There is a unique path between any two nodes, it is impossible for messages to pass each other, and the messages are processed at each network node in the order in which they arrive. Networks of this type are race-free [Landin et al. 1991]. Race-free networks have been used in a number of experimental multiprocessors built in university environments, such as Cm* [Gehringer et al. 1987], Cedar [Konicek 1991], and Hector [Vranesic et al. 1991], as well as in the KSR-1 [KSR 1992; Frank et al. 1993], a commercially available machine.

Our scheme exploits the broadcast capability, but limits the extent of the broadcast by means of a novel filtering mechanism. Filters, located in each routing node of the network, decide whether a broadcast message should be propagated further by inspecting information recorded in the message as a bit mask. The bit masks, whose size is a function of the number of levels of hierarchy in the network, are maintained with the memory on a per-block basis.

Our scheme differs from existing directory-based cache consistency schemes in two important aspects. First, it requires much less additional memory because the state information that has to be kept is less specific and hence requires only a relatively small bit mask per memory block. Second, invalidations of shared data requiring a consistency-

message broadcast are performed within the maximum time for a single broadcast. In directory schemes, the time to complete the invalidation process is determined by the time needed to transmit (possibly individual) messages to all caches that are affected.

Without loss of generality, we will present our scheme as applied to the Hector multiprocessor architecture [Vranesic et al. 1991], which uses a hierarchy of unidirectional rings to interconnect processor and memory modules (as described briefly in the next section). In Section 3 we discuss a protocol for enforcing sequential consistency, assuming a full broadcast in order to simplify the presentation. In Section 4 we introduce our filtering mechanism. Finally, in Section 5 we analyze the performance of the scheme by presenting the results of trace-driven simulations. These simulations show that the limited-broadcast scheme is scalable and that it performs well.

2. The Hector Multiprocessor Architecture

In this section, we present a brief overview of the Hector multiprocessor architecture [Vranesic et al. 1991]. The Hector architecture consists of a set of *stations* connected by a hierarchy of unidirectional rings. Each station contains a cluster of processor and memory modules, connected by a split-cycle request-response bus. The hierarchical multiprocessor is formed by interconnecting unique sets of stations by *local* rings that are in turn interconnected by higher-level rings; the ring at the top of the hierarchy is referred to as the *central* ring. While this interconnection scheme is scalable to an arbitrary number of levels, for simplicity we will assume the two-level ring hierarchy shown in Figure 1.

Each processor module consists of a processor, cache memory, a cache controller and a communication submodule. Each memory module occupies a unique contiguous portion of a flat, global (physical) address space that is transparently accessible by each processor. Information is transferred using a bit-parallel packet transfer protocol. 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 transfers between modules on the same station 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 synchronous transfer from one ring segment to the next. 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 from the station to the local ring only when the local-ring segment does not already hold a valid packet. Therefore, no flow control problems occur at the station level.

The inter-ring interfaces connect two rings and require FIFO buffers to store packets and prevent packet collisions. A collision would occur if in a given cycle the input packets from both rings are to be routed to the same output. Packets on the central ring usually have priority over those on the local ring. To make flow control unnecessary,

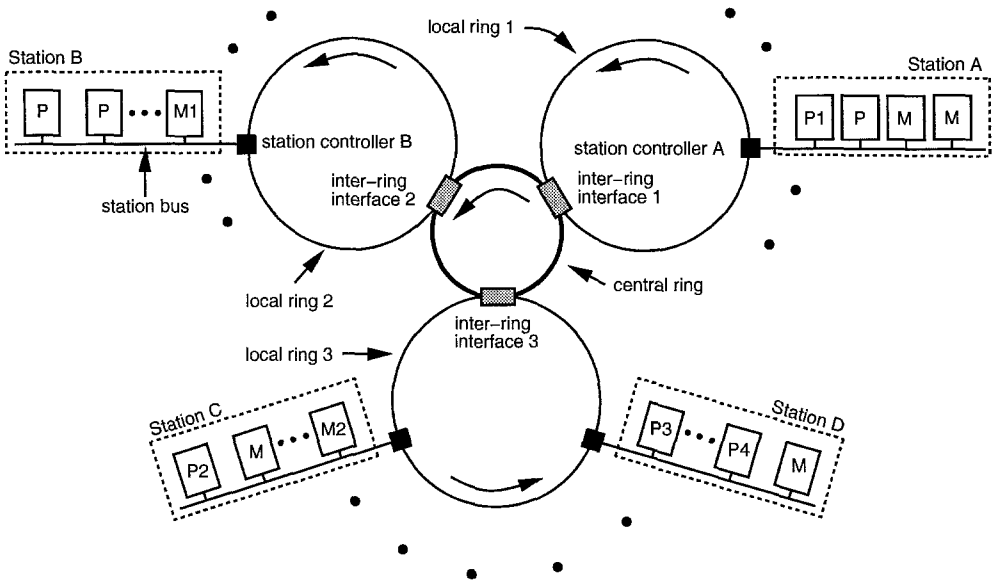

Figure 1. Structure of the Target multiprocessor.

the FIFO buffers are large enough so that they can accommodate any packet that might arrive. These FIFOs are of fixed size, because the number of processors in the system limits the number of packets being transferred in the system at any given time and each processor can have at most a limited number of outstanding memory requests. Thus, in a system with P processors and N outstanding memory requests per processor, the FIFO buffers at the inter-ring interfaces will need to be large enough to accommodate $P \times N$ requests. Finally, note that packet collisions cannot occur on the station bus because the station controller grants an on-station module use of the bus only if there is no incoming ring-packet delivery.

In each cycle, every packet is transferred to the next ring segment or the next station bus, unless the packet is buffered in an inter-ring FIFO. If a memory access packet reaches a memory module that has too many packets already queued for service, then a negative acknowledgment packet is sent back to the requesting processor so that it can retransmit the request [Vranesic et al. 1991]; the requesting processor will always be able to receive the negative acknowledgment. The invalidation packets used by the proposed cache consistency protocol are always processed by the cache modules when these packets arrive at a station. Therefore, these invalidation packets are never negatively acknowledged.

3. The Cache Consistency Protocol

We now present the cache consistency scheme in the context of the Hector multiprocessor architecture. The scheme exploits the broadcast capability of rings and uses snooping within stations to keep the caches consistent. Packet filters within each network node (i.e., each station controller and each inter-ring interface) are used to limit the scope of the broadcast messages and thereby improve the scalability of the protocol.

The protocol is suitable for invalidating and updating caches using write-through or write-back. To simplify the discussion, we will first describe the basic protocol for write-through invalidating caches, and then show how the protocol can be used with write-back invalidating caches. The described protocol enforces sequential consistency, but it can also be adapted for weaker consistency models (as discussed in Section 3.3). We begin by presenting the protocol with no packet filters, and then in Section 4 describe the packet filtering mechanism.

3.1. The Basic Protocol

With invalidating write-through caches, consistency is maintained when shared data is modified by writing the change through to the main memory and invalidating all cached copies of the data aside from the copy belonging to the processor that is the source of the write. The *invalidating* protocol entails five key steps in the common case:

1. A processor sends a write message to the target memory; the processor is then blocked.[1]
2. The memory block that is being modified in the target memory is locked.
3. A message requesting cache invalidations is sent by the target memory to the highest-level ring; this message takes the form of a *write-invalidate-pending* (WIP) packet.
4. An invalidate-cache-block message is broadcast from the highest-level ring to all stations; this message takes the form of a *write-invalidate* (WI) packet.
5. Upon receipt of the broadcast WI packet, the target memory location is unlocked. Similarly, the processor is unblocked when it receives the WI packet corresponding to its write request.

To illustrate the protocol, consider the following example. Assume that in Figure 1, processor module P1 on station A issues a write to memory module M1 located on station B. Then, P1 is blocked from making further memory accesses until it receives a WI packet, broadcast in response to its write request. 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, its station controller forms a WIP packet. This packet then ascends to the central ring where it is transformed into a WI packet by inter-ring interface 2. This WI packet then circulates around the entire central ring and is removed by the inter-ring interface that created it (inter-ring interface

2). As the WI packet circulates around the central ring, a copy of the WI packet is formed at each inter-ring interface and passed down to the associated local ring. Each newly formed WI packet in turn circulates around the local ring and is removed when it returns to the inter-ring interface responsible for its formation. As the WI packet circulates around the local ring, each station controller makes a copy of the WI packet and switches it onto the station bus to allow snooping by the on-station caches. When a WI packet reaches the memory module that initiated the invalidation (M1), the location is unlocked. Likewise, when a WI packet reaches the processor module that initiated the write (P1), the processor is unblocked. It is essential to ensure that P1 is not unblocked by a WI packet generated in response to a write by another processor. This requirement is easily achieved by including the processor ID in the WI packet.

The following details concerning the invalidation process should be noted:

- 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; this is the same interface that transformed the WIP packet into the WI packet.
- The WI packet returns to the memory module *only* after the corresponding WI packet 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, since some of the other WI packets may still be in transit.
- 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.

The locking of the memory location during the invalidation process is required in order to meet a necessary memory-access ordering requirement for sequential consistency, as shown in the appendix. While a location is locked, that is during the invalidation process, other write requests to the same location may be accepted and performed, but with the provision that the location remains locked until all the corresponding WI packets have returned to the memory module. This provision can be met with a counter for each location to count the number of outstanding WI packets. Of course, adding these counters incurs an extra implementation-dependent cost. Read requests may also be received while the location is locked, but the requested data cannot be returned to the requesting processor until the location is unlocked. With this one exception, the read requests require no special action in this cache consistency protocol.

3.2. *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. A number of researchers [Dubois et al. 1986; Gharachorloo et al. 1990; Landin et al. 1991; Scheurich 1989] have proposed methods for demonstrating that this order is preserved. These methods involve showing that a set of memory-model specific conditions are adhered to by each processor in the multiprocessor system. If this fact can be demonstrated, then the system correctly enforces the desired memory model. Using the conditions developed by Scheurich, it can be shown that the invalidation protocol described above enforces sequential consistency [Farkas 1991]. Three features of the Hector architecture, along with the blocking requirement described above, guarantee that the necessary ordering of shared accesses is preserved: (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. The need for the locking is discussed further in the appendix.

3.3. Variants of the Basic Protocol

Our cache consistency scheme can be adapted easily to implement other caching strategies and weaker consistency models.

3.3.1. Write-back Version

For simplicity, we have presented the basic protocol in terms of write-through caches. However, for many applications, better performance can be achieved with write-back caches. Indeed, the latest high-performance microprocessors feature write-back cache support. The basic protocol described above can be modified for use with such caches and processors.

The protocol for write-back caches allows a data item to be present in multiple caches when the item is only being read, but when a processor wants to modify the item, the processor must obtain exclusive ownership.

As with write-through caches, a read request for data that is not in the requesting processor's cache will result in the corresponding cache line being fetched into the cache. This line will be obtained from the present owner. The owner of the line is either the main memory or the cache with exclusive use of the cache line. If the owner of the line is a cache module, then ownership of the line is transferred back to the main memory.

Write operations are more complex. The requesting processor must become the exclusive owner of the cache line before the write can be performed. Obtaining exclusive ownership entails obtaining a valid copy of the line should it not already be in the writing processor's cache, invalidating all other cached copies of the line, and recording at the main memory the ID of the writing processor. The invalidation is achieved using the broadcast mechanism described above. As with write-through caches, the target memory location must be locked until the corresponding WI packet returns to the memory module. However, unlike with write-through caches, no subsequent read or write requests to this location may be serviced by the memory while the location is locked. When the

cache line is ejected from the cache, no special action is required; the ownership and the contents of the line are merely transferred back to the memory module.

3.3.2. *Updating Version*

Updating caches may be used by augmenting the invalidating protocol with a third phase. In the first phase of the *updating* protocol, as in the invalidating protocol, the write request is forwarded to the target memory module. In the second phase, the cached copies are updated and are also locked, thus preventing read accesses to the data item. Then, in the third phase, all copies are unlocked [Farkas 1991]. As in the case of the invalidating protocol, the locking of the cache lines is required to prevent two processors from observing writes to different locations in different orders.

3.3.3. *Relaxing the Consistency Model*

While sequential consistency is conceptually simple, it imposes restrictions on the permissible outstanding memory accesses of a processor. In addition, a performance penalty may be incurred due to the need for restricting access to a memory location during the invalidation (updating) process. By adopting a weaker consistency model, hardware optimizations are possible that increase the system performance. One such model is *processor consistency* [Goodman 1991] provided by several commercial multiprocessors, including the Silicon Graphics POWER workstation [Basket et al. 1988] and the VAX 8800 [Fu et al. 1987], both of which use a single bus, thus allowing for a simpler consistency protocol than the one we propose.

Processor consistency 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 sequential consistency and processor consistency differ. The protocol for processor consistency is very similar to the invalidating protocol described above. To achieve processor consistency, we can remove either the requirement that the memory be locked in the second phase of the protocol or the requirement that the processors be blocked after issuing a write. For the updating protocol mentioned above, either nonblocking processors can be used or the third phase need not be used.

System performance can be increased even more by further relaxing the consistency model. One such model is *release consistency* [Gharachorloo et al. 1990] in which a global ordering of memory accesses need only occur at specific synchronization points; at other times, writes can be observed by other processors in an arbitrary order. Release consistency can be implemented by adapting our protocol so that processors are blocked only at the synchronization points.

4. Scalability

In the above discussion, we have assumed that invalidation packets are disseminated using a full broadcast. While this assumption made the presentation easier to understand, it is clear that a pure broadcast scheme is not scalable. The scalability is constrained by the amount of the network bandwidth consumed by the invalidation-packet broadcasts. With increasing numbers of processors, these broadcasts will grow at a proportional rate, resulting eventually in the saturation of the interconnection network.

To increase the scalability of the proposed invalidation protocol, filters are introduced to the ring interfaces (i.e., the inter-ring interfaces and the station controllers). These filters block the propagation of invalidation packets that need not be passed on. For example, if a station contains no copies of the data being invalidated, there is no need to transmit the corresponding write-invalidate packet to this station.

Two types of filters are used to block the invalidation packets. *Outgoing filters* block the propagation to the next higher level of the hierarchy if all copies of the data being invalidated are within the portion of the system that lies below the filter. These filters limit the scope of broadcasts by blocking the write-invalidate-pending (WIP) packets. In addition, they will reduce the average time required to write a shared location, because many broadcasts will no longer start at the highest point in the hierarchy. *Incoming filters*, on the other hand, block the propagation to the next lower level of the hierarchy if there are no copies of the data being invalidated in that portion of the system. These filters reduce the unnecessary penetration of the broadcast by blocking the write-invalidate (WI) packets.

4.1. Outgoing Filters

Outgoing filters limit the ascent of write-invalidate-pending (WIP) packets to higher levels. This blocking is effected by the filter in the ring interface that is the lowest common ancestor relative to all copies of the data being written. When the blocking occurs, the ring interface transforms the WIP packet into a WI packet and broadcasts the packet to the subsystem below.

The key to the filtering is knowing the ring interface at which the WIP packet should be blocked. If the locations of all copies of a memory block are known, then this ring interface can be identified by its height in the hierarchy. The height serves to uniquely identify the ring interface because (1) there is a (unique) direct path from the memory module that sources the WIP packet to the central ring, and (2) there is exactly one ring interface at each level along this path. Each WIP packet includes the height entry corresponding to the height that must be obtained in order to reach all copies of the memory block being invalidated. When a ring module receives a WIP packet from a lower-level ring, it compares the height entry in the packet with its own height to determine whether to block the WIP packet.

In our design, the height values are maintained on a per cache-line-sized memory-block basis. These values are associated with each memory module and are stored within the

same station as the memory module. (Note that the caches do not require and do not store this information.) The height values are set on each access request to the memory module. On a write access, the value corresponding to the data being written is included in the WIP packet. After the WIP packet is formed, the value is set to the height that needs to be reached in order for a WI packet to reach the writing processor. On a read access, the value is set to the height for a WI packet to reach the accessing processor if that height is larger than the already existing height value.

4.2. Incoming Filters

Incoming filters reduce the unnecessary penetration by preventing write-invalidate packets from descending into a subsystem in which there are no copies of the data being invalidated. To decide whether to pass a WI packet on, a ring interface needs to know if the stations below it have any cached copies. A straightforward method of encoding this information in a WI packet would be to use a bit mask, where each bit in the mask would correspond to a station and would indicate whether the station contains a cached copy of the cache-line-sized memory block being invalidated. The bit masks for each memory block would be associated with the memory modules (together with the height values). The obvious problem with this scheme is that the size of the bit mask corresponds directly to the number of stations in the system and hence scales poorly.

We propose a simpler and less costly design. Our design partitions the bit mask into fields and uses each field to encode information about one level of the interconnection network hierarchy. An example of this bit mask partitioning is shown in Figure 2 for a system with 256 stations configured as a central ring (root level) connecting two level-2 rings, each level-2 ring connecting four level-3 rings, each level-3 ring connecting four local rings, and each local ring having 8 stations. In the bit mask, each bit position in a particular field corresponds to a specific path from this level to the next lower level in the hierarchy, and the value of the bit indicates whether a WI packet should be propagated down that path. The number of bits in a particular field is thus equal to the maximum number of paths from any ring at this level to the next lower level. Thus, in our example, the rightmost field of the bit mask requires 8 bits because each local ring has 8 stations. The overall hardware cost per memory block is the sum of the bits in all fields. In our example, this cost is 18 bits for the 256-station system, which is significantly less than the 256 bits required for the 1 bit per station encoding.

For an example of the use of the incoming filter bit masks, consider the very small system shown in Figure 3. Suppose that a memory location is modified, of which copies exist on stations B and C. To maintain consistency, a WI packet must propagate to these stations. In order for the WI packet to reach station B, the following bits in the mask must be set: bit 1 of field 1 (to indicate the presence of a copy within local ring 1) and bit 2 of field 2 (to indicate the presence of a copy within station 2). Similarly, for a WI packet to reach station C, bit 3 of field 1 and bit 1 of field 2 must be set. The bit mask with these four bits set is shown in the figure. With this bit mask, the WI packet will reach station B but will also reach station A because bit 1 of field 2 is set. Similarly, the WI packet will reach both stations C and D. The proposed bit encoding scheme

fields			
central ring	level–2 ring	level–3 ring	local ring
2 bits	4 bits	4 bits	8 bits

Figure 2. Incoming filter bit mask for an example 256-station system.

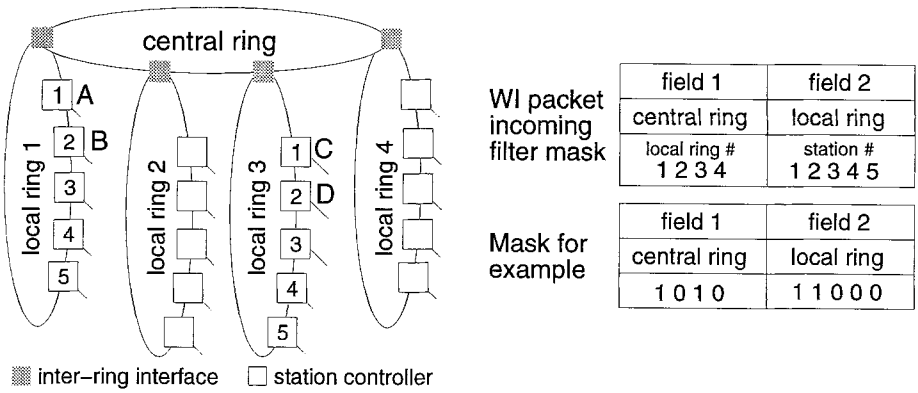

Figure 3. Illustration of the bit mask for the example in which processors on stations B and C have cached copies of a memory location.

is not perfect since the WI packet need not have reached stations A or D. However, investigations of multiprocessor applications have shown that the number of processors that actively share a given data item is small on average [Chaiken et al. 1991; Lenoski et al. 1992]; hence we expect the incoming filters to be effective.

The bit masks are maintained in the same fashion as described for the height values used with the outgoing filters. A read access to a memory block causes the setting of the necessary bits to ensure that a subsequent write to the block will result in a WI packet propagating to the reading processor's station. On a write access, after the bit mask is inserted into the WIP packet, the bits of the mask are cleared except for the bits corresponding to the path between the memory module and the writing processor. It should be noted that the bit mask is not changed when a cache line is evicted. Thus, the bit mask identifies the set of stations that includes those caches which have acquired a copy of the memory block since the last invalidation.

The bit masks reflect the path required for a WI packet to propagate from the ring interface that is the least common ancestor relative to all the copies of the data. Since the bit mask has as many fields as the network has levels, the height value is encoded in this mask by setting to zero all fields corresponding to levels above the level that the WIP packet must reach. Because this zeroing is performed as part of the process of

maintaining the incoming filter bit masks, the height values never have to be explicitly calculated or maintained.

The control structure of the filter mechanism is simple. The decision whether a broadcast packet should be blocked is made by a simple inspection of the bit mask in the packet, which has the same complexity and time delay as that incurred in making normal routing decisions in the ring interfaces.

The main hardware cost of the filters is the memory required to store the bit masks, which grows linearly with the size of the memory and logarithmically with the number of processors. Another potential cost of the filters involves the inclusion of the bit masks in each invalidation packet. In practice, however, because invalidation packets do not include data, the bit masks can be included without increasing the size of the packets.

Our scheme differs from the existing directory-based cache consistency schemes in two important aspects. First, it requires much less additional memory, because the state information that has to be kept is less specific and therefore requires relatively small bit masks. Second, all writes requiring invalidations are performed within the maximum time for a single broadcast. In directory schemes, the time to complete the invalidation process is determined by the time needed to transmit (possibly individual) messages to all caches that are affected. The only significant drawback of our scheme is that it generates some unnecessary traffic due to the imperfect blocking nature of the filters. However, as indicated by the simulation results in the next section, the effect on the utilization of the communications links is reasonable enough to lead us to believe that the proposed scheme will scale well in large systems.

5. Evaluation of the Proposed Protocol

We have investigated the effectiveness of the proposed protocol using trace-driven simulations. Because a meaningful evaluation demands that a detailed simulation model be used, we decided to model the Hector prototype [Stumm et al. 1993] at the register level. This prototype implements the variant of the Hector multiprocessor architecture in which the memory is distributed among the processor modules instead of residing in separate modules.

5.1. Simulation Methodology

The purpose of our evaluation is to assess the overall performance of the proposed invalidating protocol and to assess its potential for scalability. We use memory access latency as our metric for performance. We compare our scheme against two alternatives: a scheme in which shared data is not cached and an optimistic extreme in which invalidation overhead is assumed to be zero. We have made this comparison for a number of topologies and have found that a balanced topology gives the best performance. We assess scalability by investigating the impact of the proposed protocol on the utilization of the network links for a system with up to 128 processors. Using a balanced topology,

Application	No. of Procs.	No. of Memory Refs.	Distribution of Memory Operations (in %)					
			Instruction fetches	Read–modify–writes	Private Data		Shared Data	
					Reads	Writes	Reads	Writes
Simple	64	19 M	40.9	13.4	19.2	9.9	14.8	1.7
Speech	64	11 M	—	—	—	—	78.2	21.8
Weather	64	19 M	41.5	0.8	40.2	7.0	8.5	2.0
SOR	64	762 M	61.5	—	—	—	30.8	7.7
MP3D	64	15 M	—	1.0	25.9	9.1	38.8	25.2
	128	18 M	—	1.0	26.7	10.7	37.3	24.2
Water	64	27 M	—	0.2	61.0	23.3	13.9	1.6
	128	26 M	—	0.2	60.9	23.3	14.0	1.6

Table 1. Trace characteristics: Simple models the behavior of fluids and uses finite difference methods; Speech implements the lexical decoding stage of a speech interpretation language (the trace contains only data references); Weather solves a set differential equations using finite-difference methods; SOR is an iterative method for solving partial differential equations; MP3D models a wind tunnel using discrete particle-based simulation; Water computes the energy of a system of water molecules.

we show that the proposed protocol scales well when used with both write-through and write-back policies.

In the simulations, the processor was modeled using an event generator to emulate the execution of a multiprocessor application. We will show the results for six different applications running on either 64-processor or 128-processor systems. For these applications, the memory references were obtained from address traces whose distributions of memory operations are shown in Table 1. The traces for Simple, Speech and Weather were obtained from the MIT trace set; Chaiken et al. describe the applications and the methods used to acquire these traces [Chaiken et al. 1990]. The remaining traces were generated using the MINT multiprocessor simulation package [Veenstra and Fowler 1994]: SOR (Successive Over-Relaxation), and Water and MP3D which are applications from the SPLASH benchmark set [Singh et al. 1991]. The traces for the last two applications correspond to the complete parallel-execution stage of the benchmark, but do not include instruction references (in order to decrease the size of the traces and decrease the simulation time).

Because the address traces were acquired from machines dissimilar to Hector, it is only meaningful to compare the relative results for the different caching strategies and topologies; the absolute numbers are not meaningful. Thus, we view the traces merely as a sequence of memory references rather than the reference stream that would be generated if the applications were compiled for and run on Hector. Furthermore, while our MINT-generated traces contain inter-reference timing information, the three MIT traces do not. For these traces we assumed a one-cycle delay between memory references. The traces do not contain mapping information assigning the address-reference streams to physical processors. Similarly, the traces do not specify a virtual to physical page assignment. In the simulations, we performed optimal page assignment by first preprocessing the

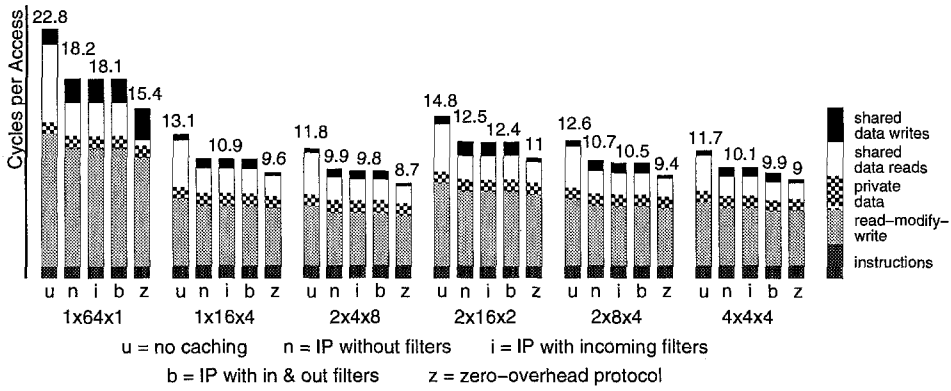

Figure 4. Average memory access latency for the Weather application.

traces and then statically allocating pages such that a page is located closest to those processors that issued the most references to it. Finally, the MIT traces contain only a small percentage of the total memory references the applications would have issued during their entire execution, and the MINT traces contain only the shared-data accesses made in the parallel execution phase of the applications. For these reasons, the results we present correspond to the data gathered to the point where the first processor exhausts its stream of references.

5.2. *Latency Analysis*

To assess the effects on memory access latency, five scenarios were simulated using the traces of Table 1: (1) Shared data is not cached, (2) shared data is cached using the write-through invalidating protocol without filters, (3) the same protocol is used with incoming filters, (4) the same protocol is used with both incoming and outgoing filters, and (5) 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. In each of these scenarios, the invalidating protocol enforces sequential consistency. The metric we use in comparing these scenarios is the weighted-mean number of clock cycles. This metric is independent of the cycle time of a system and hence allows us to conduct a comparison that is not tied to a particular implementation.

Figure 4 presents the weighted-mean memory access latency for the execution of the Weather application. The various degrees of shading in the figure show 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 local rings, the second the number of stations per local ring and the third the number of processors

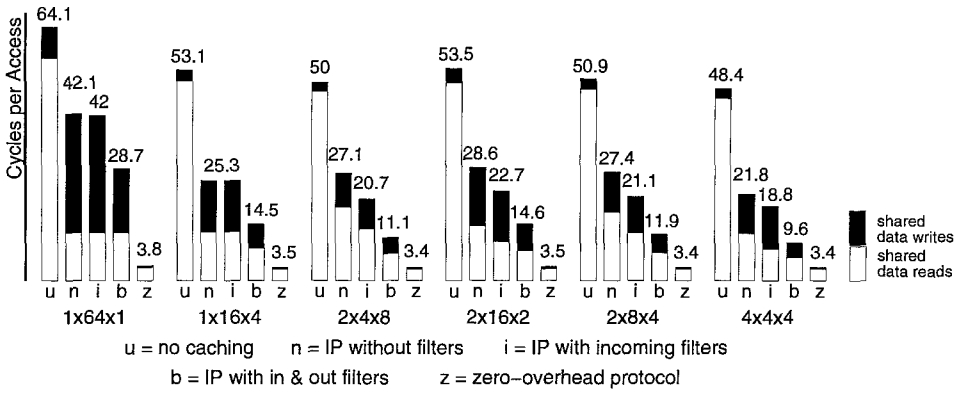

Figure 5. Average memory access latency for the Speech application.

per station. Because the cache hit rates for instruction and private data accesses are very high, the contribution to the average latency by these access types is constant. In this particular application, most of the latency is attributable to the read-modify-write operations. As can be seen, the average latency is generally reduced with cache consistency. The scenarios using the invalidating protocol perform within 12% of the ideal zero-cycle overhead scheme; their near identical performance is attributable to low network utilizations.

Figure 5 presents similar results for the Speech application. Compared to Weather, Speech shows a more pronounced reduction in the memory access latency with the use of the invalidating protocol. This observation is not surprising because the Speech application trace essentially contains only shared-data references (see Table 1). The use of incoming filters alone improves the performance by an average of 17% over the topologies investigated. This improvement is brought about by the considerable reduction in the average station bus utilization from 56% without filters to 13% with incoming filters. When outgoing filters are also used, the performance improves further by a factor of two on average, as a result of the reduction in shared-data access latency brought about by the outgoing filters.

Figure 6 gives the results for the SOR application. As shown in the figure, the use of both types of filters achieves latencies within 20% of the ideal zero-overhead scheme. However, it is interesting to note that the memory access latency of the invalidating protocol without filters can be worse than uncached operation if the topology is unbalanced. (We should also note that in this application the filters do not improve noticeably the average latency of read accesses. The reason is that the vast majority of reads are local due to our page placement policy; hence, they are not affected by the changes in network utilization.)

Our simulations have shown that for the traces in Table 1, the average memory access latency is reduced with the use of the invalidating protocol. The obtained results also reveal that the topology of the system has a significant impact on the performance. The

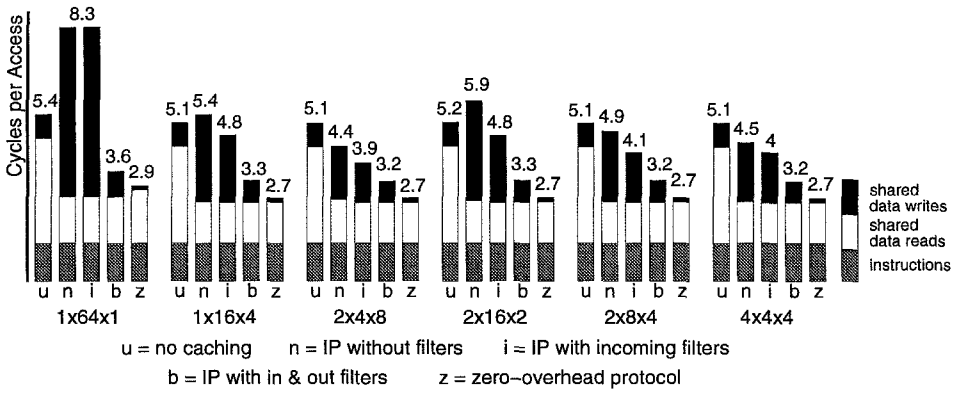

Figure 6. Average memory access latency for the SOR application.

best topologies are those that are balanced in terms of the number of processors, stations and rings. This preference occurs because the more balanced topologies tend to maximize the number of independent concurrent transactions in the interconnection network. In addition, the average distance between any two modules is shorter in these topologies than in unbalanced topologies. This distance has a direct impact on the time required to access a remote memory location as well as the length of time a memory location remains locked during the second phase of the invalidating protocol.

5.3. *Traffic Analysis*

Figures 4–6 show that performance can be improved significantly with the use of the invalidating write-through protocol. We now consider the utilization of the network links to show that the filter mechanism improves scalability. Since the balanced topologies generally exhibit superior performance, we will present only the results obtained from the simulations of balanced systems. We begin by presenting data for simulations of a 64-processor system for the 4×4×4 topology with write-through. We then present data for simulations of a system using write-back. Finally, we show results from the simulations of 128-processor systems.

5.3.1. *Write-through*

The primary purpose of the filters is to reduce the utilization of the interconnection network by limiting the scope and penetration of the invalidation packet broadcasts. The effectiveness of the filters may be assessed by looking at the utilization of the various links in the interconnection network. Figure 7 depicts the effect of incoming filters on the utilization of the station buses and local rings for the six different applications. The utilization of the central ring is not given because incoming filters have no effect on

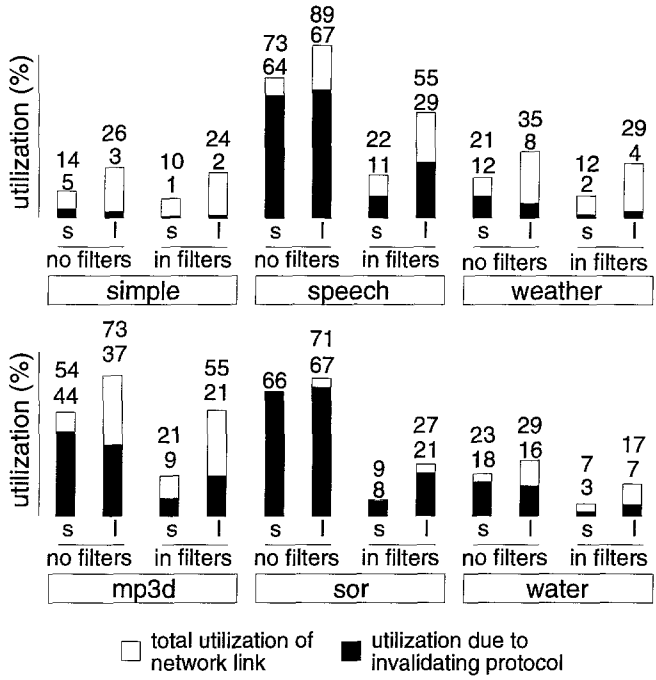

Figure 7. Utilization of the station buses (s) and local rings (l) using incoming filters with write-through.

the traffic on this ring. In the figure, the upper number above each bar gives the actual utilization of the network link while the lower number gives the utilization of the link due to invalidation packets. It is apparent from the figure that incoming filters reduce significantly both the utilization of the network links and the portion of this utilization attributable to invalidation packets. Incoming filters therefore enhance scalability.

Figure 8 gives the percentage of the invalidation packets that are blocked by the incoming filters. The second and third bars (for each application) correspond to the invalidation packets that pass through the inter-ring and station-controller filters. The fourth bar (black) indicates how many invalidation packets are actually required to reach the stations because of the existence of a copy of the data being invalidated. As seen in the figure, for some applications the incoming filters reduce the invalidation traffic close to the minimum required. For the other applications the reduction is not as great, because of the imperfect nature of our simple filtering mechanism; still, less than a third of the invalidation packets generated reach a station, which indicates the effectiveness of the filters.

For the write-through protocol, outgoing filters have an even greater potential impact on scalability than incoming filters. As illustrated in Figures 4–6, the use of these filters can greatly decrease the shared-data write latency, thereby improving the performance. In addition, the latencies for all types of accesses may decrease due to the reduced network

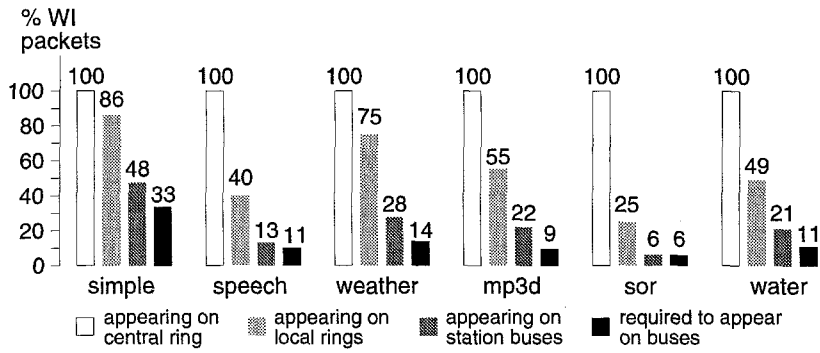

Figure 8. Effectiveness of incoming filters in blocking invalidation packets with write-through.

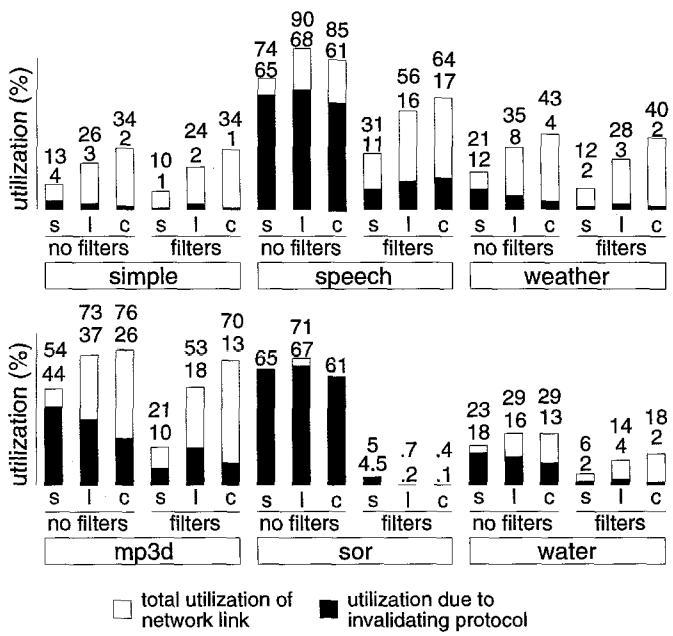

Figure 9. Utilization of the station buses (s), the local rings (l) and the central ring (c), using both incoming and outgoing filters with the write-through invalidating protocol.

load brought about by restricting the scope of the broadcasts. Figure 9 depicts the effect of using both incoming and outgoing filters on the utilization of the network as well as the portion of this utilization attributable to invalidation packets. It is apparent that the use of the outgoing filters further reduces the utilization, thus enhancing the scalability of the protocol.

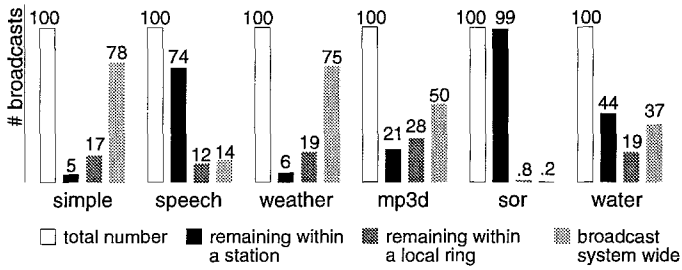

Figure 10. The effectiveness of outgoing filters in achieving a limited broadcast with write-through.

Figure 10 depicts the effectiveness of the outgoing filter mechanism in achieving a limited broadcast. The second bar gives the portion of the broadcasts limited to a single station, the third bar shows the portion limited to a single local ring, and the fourth shows the portion that results in a system-wide broadcast. These portions are expressed as a percentage of the total number of invalidation packets generated by the memory modules. The results show that the outgoing filters reduce the scope of the broadcasts significantly in all cases and produce a dramatic reduction in the cases of the Speech and SOR applications.

5.3.2. *Write-back*

Shared-memory multiprocessors often use the write-back invalidation protocol for cache consistency because of its superior performance characteristics. In Figure 11 we depict the effectiveness of our filtering scheme when used with write-back, in which an invalidation packet is broadcast only if the data being written is actively shared. In this case, the cache itself prevents broadcasts caused by writes to data that is not actively shared. In effect, the write-back cache acts as an outgoing filter at the processor level. However, the outgoing filters can still have a beneficiary effect at the station and ring levels, as indicated by Figure 12. In the figure, the second bar of each application represents the number of writes that remain local to the cache; the third and fourth bars represent the broadcasts affected by the outgoing filters.

5.3.3. *Larger Configurations*

We have simulated some of the applications running on a 128-processor system. Figure 13 shows the utilization of network links for the MP3D and Weather applications, using both the write-through and the write-back protocols. In each case, invalidation packets constitute most of the traffic at the station level if filters are not used, but this traffic is substantially reduced using filters. As the system becomes even larger, the

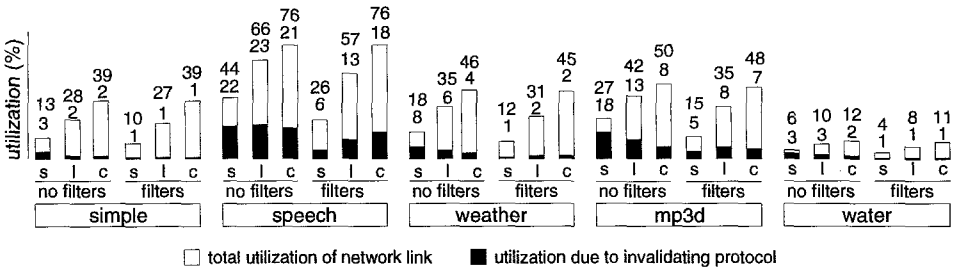

Figure 11. Utilization of the station buses (s), the local rings (l) and the central ring (c), using both incoming and outgoing filters with the write-back invalidating protocol.

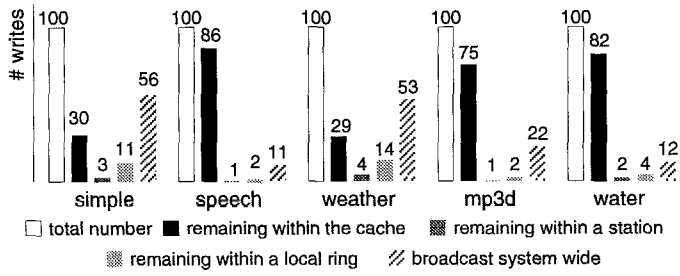

Figure 12. The effectiveness of outgoing filters to achieve a limited broadcast with write-back.

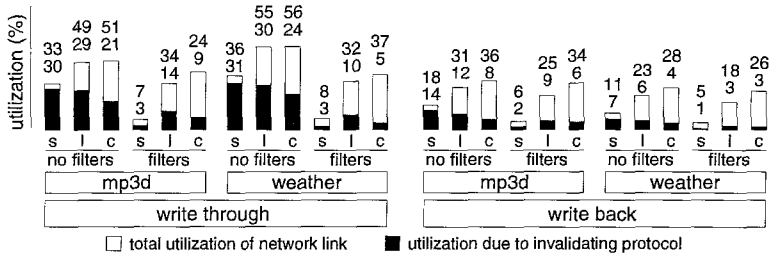

Figure 13. Utilization of the station buses (s), the local rings (l) and the central ring (c), using both incoming and outgoing filters for a system with 128 processors arranged in a 4×8×4 topology.

relative proportion of the invalidation traffic increases accordingly. Therefore, the role of filters becomes more important.

6. Concluding Remarks

We have presented a new cache consistency scheme, targeted for hierarchically structured shared-memory multiprocessors. The protocol is based on broadcasting invalidation packets, using a packet filtering mechanism to limit the extent of broadcasts. The key advantages of our protocol follow:

1. It is simple and inexpensive to implement,
2. it has a minimal effect on packet transfer delays, and
3. it completes invalidation in minimal and bounded time.

We have shown that our protocol scales well to medium-sized multiprocessors and that it performs well for several typical applications.

It is interesting to compare the main features of the proposed consistency scheme with other existing schemes. For example, the KSR multiprocessor [KSR 1992; Frank et al. 1993] is also based on a hierarchy of rings and uses a consistency protocol that involves broadcasting of invalidation packets. It uses filters at the inter-ring interfaces. The filters are precise and hence require much state information to be maintained at the filters, since each filter must be able to keep track of the contents of all caches below it in the hierarchy. Thus, the size required for the state information grows by a multiplicative factor with the level of the ring in the hierarchy, essentially precluding scaling to beyond two or three levels of hierarchy. Moreover, the state information in a filter must be searched with each passing invalidation packet to determine how to process that packet. The time required for this search effectively limits the speed with which a packet can traverse a network node. In contrast, in our scheme, the decision whether to filter out a packet is made by examining only one bit in the packet, which can be achieved easily in one clock cycle and requires no state information in the filters themselves.

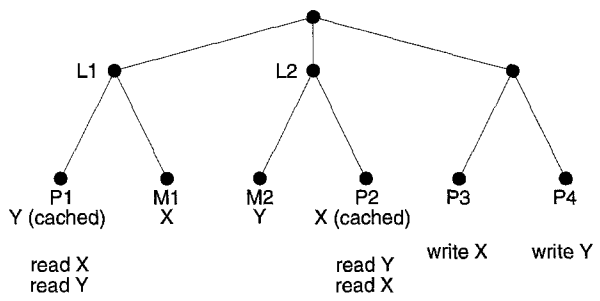

Figure 14. A portion of the hierarchy of a system that includes four processors and two memory modules.

Directory-based schemes attempt to send point-to-point invalidation packets directly to the caches that have a copy of the data to be invalidated. Therefore, they are either serial in nature, invalidating one copy after another, or they require complex bookkeeping to keep track of the acknowledgments of multiple outstanding invalidation packets if invalidations can occur concurrently. The SCI protocol (used to implement ring-based topologies, including hierarchical ones) is an example of a serial protocol, in which actively shared data is kept in a linked list that spans the caches with a copy of the data [Gustavson 1992]. Invalidation proceeds along the linked list so that the invalidation of n cached copies will require sending n packets, which in the worst case have to traverse the entire system. Our protocol results in concurrent invalidation rather than sequential. While it generates some superfluous traffic at the lower levels of the hierarchy due to the imperfect nature of our filters, it minimizes the invalidation traffic at the top level of the hierarchy. For these reasons, the elapsed time of the invalidation process in our scheme will never exceed that of any directory-based scheme.

Presently, we are applying the proposed cache consistency scheme in the design of a 64-processor prototype machine at the University of Toronto. This multiprocessor will use R10000 processors in a balanced topology that has one central ring connecting four local rings, each having four stations with four processors.

Appendix

Enforcing Sequential Consistency

In multiprocessors, writes to shared-memory locations are said to occur when they are observed by the other processors. Thus, of concern is the order in which writes are observed rather than the order in which they are issued. For sequential consistency, the observed order of all writes must be the same for all processors. This requirement is enforced by the protocols presented in this paper. The enforcement requires that the location being written to is locked during the invalidation (updating) phase. We will illustrate by an example why locking is necessary for imposing a global ordering on

writes; we assume invalidating caches. A more formal discussion on locking and on the requirements for sequential consistency is presented in [Farkas 1991].

Figure 14 shows a portion of the hierarchy of a system that includes processors P_1, P_2, P_3 and P_4, and memory modules M_1 and M_2. Let X be a memory location in M_1 and Y be a location in M_2. Also, assume that P_1 has a cached copy of Y, while P_2 has a cached copy of X. Now, suppose P_3 performs (successfully) a write to X and P_4 performs a write to Y. If the invalidating protocol is used without locking, the following sequence of events is possible. Let P_1 issue a read of location X. Since X has been changed, P_1 will receive the new value of X. Next P_1 issues a read of location Y. If we assume that the write-invalidate (WI) packet for Y has not yet reached P_1, then P_1 will get the old value of Y from its cache. Similarly, if P_2 issues a read of location Y, it will receive the new value of Y. If it next issues a read of X, it will read the old value of X from its cache if the WI packet corresponding to the change of X has not yet reached P_2. Therefore, P_1 and P_2 see the changes made to locations X and Y in different orders.

With locking, the above scenario cannot occur. In our example, the WI packets for both X and Y will reach the topmost node in the hierarchy. Assume that WI_X, the WI packet for X, descends to node L_2 before WI_Y. Then, it is guaranteed that P_2's copy of X will be invalidated before P_2 can obtain a new value from Y because this memory location is locked until WI_Y reaches M_2. Clearly, a symmetrical situation can occur when WI_Y precedes WI_X at node L_1, thus preventing P_1 from reading the old value from its cache.

Acknowledgments

The research described in this paper has been partially funded by the Natural Sciences and Engineering Research Council of Canada and by the Information Technology Research Center of Ontario.

We thank Stephen Brown, Orran Krieger, Michiel van de Panne, Harjinder Sandhu and the anonymous reviewers for their useful comments. Orran deserves additional thanks for pointing out that the outgoing filter height value can be encoded in the incoming filter bit masks.

Notes

1. We assume that the processor does not issue multiple writes because we are trying to achieve sequential consistency.

References

Barroso, L., and Dubois, M. 1991. Cache coherence on a slotted ring. In *Conference Proceedings–International Conference on Parallel Processing* (Austin, Texas, Aug. 12-16), CRC Press Inc., pp. I-230-I-237.

Basket, F., Jermoluk, T., and Solomon, D. 1988. The 4D-MP graphics superworkstation: Computing + graphics = 40 MIPS + 40 MFLOPS and 100,000 lighted polygons per second. In *Conference Proceedings–The 33rd IEEE Computer Society International Conference - COMPCON* (San Francisco, California, Feb. 24 - Mar. 4), IEEE Computer Society Press, pp. 468-471.

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

Chaiken, D., Kubiatowicz, J., and Agarwal, A. 1991. LimitLESS directories: A scalable cache coherence scheme. In *Conference Proceedings–The Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Santa Clara, California, April 8-11), The Association for Computing Machinery (ACM), pp. 224-234.

Dubois, M., Scheurich, C., and Briggs, F. A. 1986. Memory access buffering in multiprocessors. In *Conference Proceedings–The 13th Annual International Symposium on Computer Architecture* (Tokyo, Japan, June 2-5), IEEE Computer Society Press, pp. 434-442.

Farkas, K. I. 1991. A decentralized hierarchical cache-consistency scheme for shared-memory multiprocessors. Master's thesis, University of Toronto, Technical Report no. EECG TR-91-04-01 (Electrical Engineering Computer Group).

Frank, S., Rothnie, J., and Burkhardt, H. 1993. The KSR1: Bridging the gap between shared memory and MPPs. In *Conference Proceedings–IEEE Compcon 1993 Digest of Papers*, (San Francisco, California, Feb. 22-26), IEEE Computer Society Press, pp. 285-294.

Fu, J., Keller, J., and Haduch, K. 1987. Aspects of the VAX 8800 C box design. *Digital Technical Journal*, 4, 2 (Feb.) : 41-51.

Gehringer, E., Siewiorek, D., and Segall, Z. 1987. *Parallel Processing: The Cm* Experience*. Digital Press.

Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A., and Hennessy, J. 1990. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Conference Proceedings–The 17th Annual International Symposium on Computer Architecture* (Seattle, Washington, May 28-31), IEEE Computer Society Press, pp. 15-26.

Goodman, J. 1991. Cache consistency and sequential consistency. Technical Report no. 1006, Computer Sciences Department, University of Wisconsin-Madison.

Gustavson, D. 1992. The Scalable Coherent Interface and related standards projects. *IEEE Micro*, 12, 1 (Jan.) : 10-22.

Konicek, J. 1991. The organization of the Cedar system. In *Conference Proceedings–The 1991 International Conference on Parallel Processing*, (Austin, Texas, Aug. 12-16), CRC Press Inc., pp. 149-156.

KSR. 1992. KSR1 principles of operation. Technical report, Kendall Square Research.

Lamport, L. 1979. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28, 9 (Sept.) : 690-691.

Landin, A., Hagersten, E., and Haridi, S. 1991. Race-free interconnection networks and multiprocessor consistency. In *Conference Proceedings–The 18th Annual International Symposium on Computer Architecture* (Toronto, Canada, May 27-30), IEEE Computer Society Press, pp.106-115.

Lenoski, A. D., Laudon, J., Gharachorloo, K., Gupta, A., and Hennessy, J. 1990. Directory-based cache coherence protocol for the DASH multiprocessor. In *Conference Proceedings–The 17th Annual International Symposium on Computer Architecture* (Seattle, Washington, May 28-31), IEEE Computer Society Press, pp. 148-158.

Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J., Horowitz, M., and Lam, M. 1992. The Stanford Dash multiprocessor. *Computer*, 25, 3 (March) : 63-79.

Scheurich, C. E. 1989. Access ordering and coherence in shared memory multiprocessors. Ph.D. thesis, University of Southern California, Technical Report no. CENG 89-19 (Computer Engineering).

Singh, J. P., Weber, W.-D., and Gupta, A. 1991. SPLASH: Stanford parallel applications for shared memory. Technical Report CSL-TR-91-469, Computer Systems Laboratory, Stanford University.

Stumm, M., Vranesic, Z., White, R., Farkas, K., and Unrau, R. 1993. Experiences with the Hector multiprocessor. In *Conference Proceedings–Seventh International Parallel Processing Symposium* (Newport Beach, California, April 13-16), IEEE Computer Society Press, pp. 10-18.

Veenstra, J. E., and Fowler, R. J. 1994. MINT: A front end for efficient simulation of shared-memory multiprocessors. In *Workshop Proceedings–The Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (Los Alamitos, Jan.), IEEE Computer Society Press, pp. 201-207.

Vranesic, Z. G., Stumm, M., Lewis, D. M., and White, R. 1991. Hector: A hierarchically structured shared-memory multiprocessor. *Computer*, 24, 1 (Jan.) : 72-79.