In Proc. of the 1995 Int. Conf. on System Sciences
(28th HICSS), vol. I, pp. 242-251, IEEE Jan. 1995

Using Hints to Reduce the Read Miss Penalty for Flat COMA Protocols*

Mårten Björkman, Fredrik Dahlgren, and Per Stenström

Department of Computer Engineering, Lund University
P.O. Box 118, S-221 00 LUND, Sweden

Abstract

In flat COMA architectures, an attraction-memory miss must first interrogate a directory before a copy of the requested data can be located which often involves three network traversals. By keeping track of the identity of a potential holder of the copy—called a hint—one network traversal can be saved which reduces the read penalty.

We have evaluated the reduction of the read miss penalty provided by hints using detailed architectural simulations and four benchmark applications. The results show that a previously proposed protocol using hints actually can make the read miss penalty larger because when the hint is not correct, an extra network traversal is needed. This has motivated us to study a new protocol using hints that simultaneously sends a request to the potential holder and to the directory. This protocol reduces the read miss penalty for all applications but the protocol complexity does not seem to justify the performance improvement.

1. Introduction

Cache-coherent NUMA (CC-NUMA) and cache-only memory architectures (COMA) are two emerging styles of building scalable shared-memory architectures. Examples of the former type include the Stanford DASH [14] and the MIT Alewife [1] whereas the Swedish Institute of Computer Science's Data Diffusion Machine (DDM) [13] and Kendall Square Research's KSR1 [4] are examples of the latter type. Both styles use processing nodes that consist of processors, caches, and a portion of the distributed main memory. In contrast to CC-NUMA machines, main memory in COMA is converted into huge caches called *attraction memories*, that support replication of data not only across caches, but also across memories.

The main advantage of COMA as compared to CC-NUMA machines is that a vast majority of replacement cache-misses can be handled in the local attraction memory [17]. However, to handle cache misses that cannot be carried out locally, a mechanism is needed that locates the node in which a copy of the memory block resides. DDM and KSR1—examples of hierarchical COMAs—use a hierarchical directory structure. Therefore, the latency of locating a copy can include several directory lookups. This is in contrast to in CC-NUMA machines where a single directory lookup locates a copy.

COMA machines can also locate copies using a single directory lookup as in CC-NUMA machines which is the basic idea behind the flat COMA (COMA-F) proposal by Stenström *et al.* [17]. When a cache miss cannot be serviced by the local attraction memory, the miss request is sent to a directory which then forwards the request to an attraction memory that keeps a copy of the block. This attraction memory then returns the copy to the requesting node. While the whole transaction often includes three network traversals, two network traversals would suffice, did the requesting node know where to retrieve a copy.

To be able to send the read-miss request directly to a holder of the block, one can associate an identifier of the potential block holder—called a *hint*—with each attraction-memory block-frame. This concept was incorporated in a COMA-F protocol by Gupta *et al.* in [12]. If the hint is correct, the copy is retrieved in two network traversals but if the hint turns out to be wrong, the directory has to be interrogated. Since this costs the latency of an extra network traversal, the hints must be correct in at least fifty percent of the cases to reduce the read miss penalty.

We evaluate in this paper the performance improvement using hints on a simulated flat COMA machine and four benchmark applications. While we evaluate the protocol using hints according to [12], we find that the savings can be offset by the extra network traversals associated with unsuccessful hints. This motivates us to study a new protocol using hints that simultaneously sends a request to the potential holder as well as to the directory; clearly, unsuccessful hints do not introduce extra miss latency in this protocol. Although we find that this protocol cuts the read-miss penalty—in some case by 14%—the improvement is limited by the small fraction of misses that can use hints and the low success rate of hints.

In the next section, we review the latency associated with read misses in protocols for CC-NUMA and COMA machines and Section 3 presents a new COMA protocol that uses hints to reduce the read-miss penalty. We present our architectural simulation results in Sections 4 and 5 before we conclude in Section 6.

*This work was supported by the Swedish National Board for Technical Development (NUTEK) under the contract P855.

2. CC-NUMA and Flat COMA Protocols

Penalties for memory operations, i.e., the number of cycles a processor is stalled waiting for a memory operation to complete, are important to combat in shared-memory multiprocessors. While the penalty in servicing write operations can be eliminated by exploiting relaxed memory consistency models [8], as we assume in this paper, penalties associated with read misses—read miss penalties—are much harder to attack.

Many techniques have been proposed to reduce read miss penalties including prefetching [15 ,5,6] and update-based cache-coherence protocols [10,7]. In this paper, we focus on penalty reduction and cost of COMA protocol optimizations as compared to CC-NUMA protocols. The framework for our comparison is the coherence protocol in DASH [14] and in the flat COMA protocol proposed by Gupta *et al.* [12]. The review of these protocols, that appear in Sections 2.1 and 2.2 and that we simulate in Section 5, provides an intuition as to how *hints* can make a flat COMA protocol performing better. We then review the original flat COMA protocol with hints in Section 2.3.

The general structure of the CC-NUMA and COMA machines we consider in this paper appears in Figure 1. It consists of a number of processing nodes that each contains a processor, a private cache, and a portion of the main memory. The processing nodes are connected by a general interconnection network for which the only requirement is that a request sent from one node to another always uses the same path (i.e. FIFO order is preserved).

2.1 A CC-NUMA Protocol

Data and code pages are initially mapped to the various memory modules in a CC-NUMA machine. The node in which a specific page is mapped is called the *home node* of that page and the memory blocks it contains.

Replication of memory blocks is only supported across the private caches and consistency among cached memory blocks is maintained by a system-level write-invalidate protocol. The basic mechanism consists of a directory

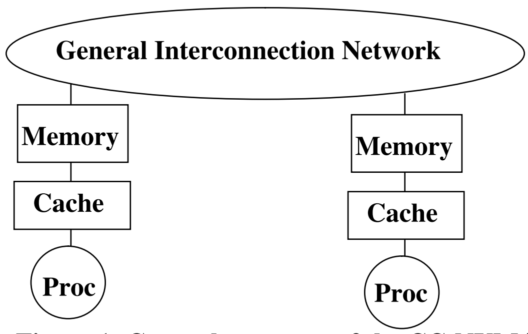


Figure 1: General structure of the CC-NUMA and COMA architectures.

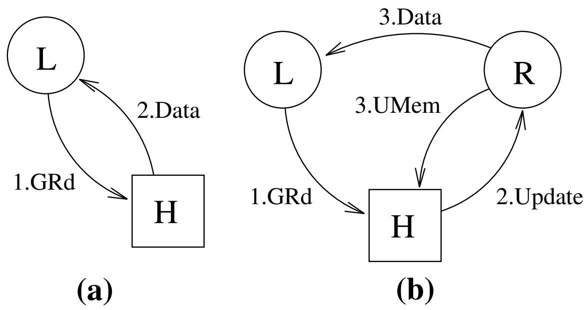


Figure 2: A global read request when the block is (a) SHARED or UNCACHED (b) MODIFED.

entry associated with each memory block that keeps track of which caches have copies using a presence-flag vector. Moreover, the directory entry also encodes the state of the memory copy which can be *UNCACHED*, *SHARED*, or *MODIFIED.* Similarly, each cache copy can be in one of three states: *INVALID, SHARED,* or *DIRTY.*

Let us now recapitulate the cache read-miss transactions in the DASH protocol. (For more details, the reader is referred to [14].) To simplify the discussion, we refer to the node in which the cache miss originates as the *local node*; a node other than the local node and the home node that is involved in the cache-miss transaction is referred to as a *remote node*.

If Local (L) is not the same as Home (H), a cache miss results in a global read-miss request (*GRd*) to Home. If the memory copy is *SHARED* or *UNCACHED*, a copy is returned to Local. This read-miss transaction includes two network traversals (or *hops*) and is shown in Figure 2a. If the memory block is *MODIFIED*, however, the copy must be retrieved from Remote (R) which keeps the only copy in state *DIRTY.* This is done by sending an update request to Remote (*Update* in Figure 2b). When Remote receives the request, it sends a fresh copy to Home (*UMem*) as well as to Local (*Data*). When Home receives the copy, the state of the memory block is changed to *SHARED.* We note that when Local, Home, and Remote are different nodes, a read-miss transaction includes three hops whereas if Home is either the same as Local or Remote, it requires less than three hops.

Whether a read miss is serviced in zero, two, or three hops in CC-NUMA depends on (i) the location of Home with respect to Local and (ii) the state of the memory copy. First, if Local is the same as Home and if the memory copy is clean, i.e., in state *UNCACHED* or *SHARED*, the read miss can be serviced locally; if the memory block is in state *MODIFIED*, the read miss is serviced in two hops. Second, if Local is not the same as Home but the memory copy is clean, the read miss is serviced in two hops, whereas if the memory copy is in state *MODIFIED*, three hops are needed. We note that if a page is mapped to Local, then all Local's misses to that page are serviced in at most two hops. Especially in the absence of invalidations, all misses are serviced locally.

The way CC-NUMA machines reduce the number of hops per cache read miss is by a careful mapping of pages to nodes. This mapping strives at increasing the likelihood of finding the home node in the local node. Unfortunately, the absence of support for page replication limits this approach. This is why COMA machines have a potential to reduce the number of hops per cache read miss by using hardware support for replication at the main-memory level.

2.2 A Flat COMA Protocol

To support replication of memory blocks at the main-memory level, COMA machines convert each memory module into a huge cache by associating tag and state bits with each memory block-frame. Coherence across these main-memory caches, referred to as *attraction memories* (AM), is maintained by a system-level write-invalidate protocol.

Owing to the replication of memory blocks at the main-memory level, a vast majority of the cache replacement misses[1] can be handled by the local AM [17]. On the other hand, cold misses to memory blocks belonging to pages mapped to other nodes and coherence misses must be handled remotely. Since memory blocks can migrate across memories, a mechanism is needed to locate an AM that has a copy for remotely serviced misses. The Data Diffusion Machine[13] and the Kendall Square Research's KSR1[4] employ a hierarchical directory mechanism that may introduce several directory lookups before the copy is located. This is in contrast to in CC-NUMA machines where a single directory lookup (in Home) is needed to locate the copy. In [17], it was shown that the latency caused by the hierarchical mechanism can offset the gains of the low replacement-miss latency even if pages are randomly distributed in a CC-NUMA machine.

The flat COMA (COMA-F) as proposed by Stenström *et al.* in [17] uses a similar notion of a home node for each memory page as in CC-NUMA. Unlike CC-NUMA, however, a memory copy is not necessarily allocated in Home's AM even if it is clean. Instead, another dynamically assigned node—the *master* of the memory block—is responsible for a "master copy" of the block and will service all AM read-miss requests. Each directory entry consists of a pointer to the current master and a presence-flag vector to keep track of memory copies. We next study the

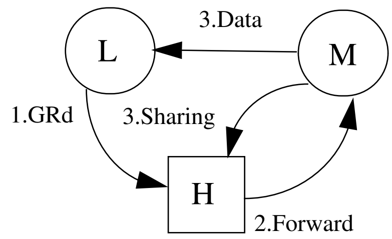


Figure 3: Flow of a global read-miss transaction in a flat COMA protocol.

protocol for inter-node AM read misses in the protocol proposed in [12]. Like in Section 2.1, we refer to the requesting node as Local and the node that keeps the directory as Home. In addition, we refer to the node that keeps the master copy as the Master and any other node involved in an inter-node read-miss transaction as Remote.

Each directory entry can be in two stable states, *EXCLUSIVE* and *SHARED*, that indicate that there is exactly one or more than one memory copy in the system, respectively. Moreover, the directory state can be also in a transient state, *WAIT_INVALIDATE*, indicating that an ownership transaction is in progress.

A cache read-miss that cannot be serviced in the local AM results in a global read-miss request (*GRd*) which is sent to Home as shown in Figure 3. Home then forwards the request to Master (*Forward)* that returns a copy to Local (*Data)*. Local fills the AM as well as the private cache with the block. Home will not change the directory state (if ever) until it receives the transfer request from Master (*Sharing* in Figure 3).

Ownership transactions are handled according to Figure 4. When Home receives an ownership request (*GWr)*, it forwards it to the Master (*WForward)* and the state of the directory entry becomes *WAIT_INVALIDATE*. From now on, all incoming read miss as well as ownership requests will be rejected and have to be retried. When Master receives the forward request, it returns a copy of the block to Local (*WData)* and notifies Home (*Transfer*). When Home receives this message, the state becomes *EXCLUSIVE* and Local is deemed the new master. In parallel, Home issues invalidations to all sharers (called Remote in Figure 4) and sends a message to Local (*WrAck)*. Local will receive all invalidation acknowledgments (*IAck*) as in the DASH protocol [14].

While the transient state *WAIT_INVALIDATE* prevents a race condition if a global read-miss or another ownership request arrives at Home during an ownership transaction, other race conditions can occur. Assume that an ownership request arrives at Home during a global read-miss transaction when Home has sent *Forward* to

1. Misses that result from size or associativity constraints in the private caches in the absence of invalidations.

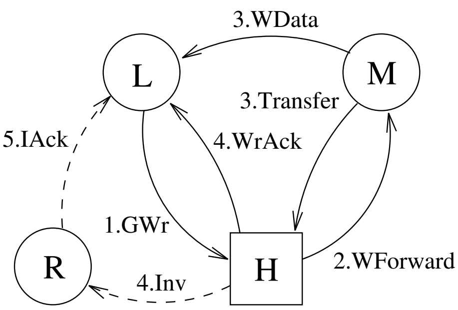


Figure 4: Flow of an ownership transaction in a flat COMA protocol.

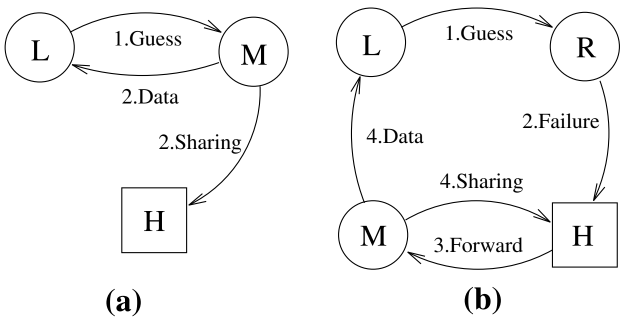


Figure 5: Flow of a global read-miss transaction using hints. (a) Successful hint (b) Unsuccessful hint.

Master (see Figure 3). Since the network preserves FIFO order between any two nodes, Home will receive the *Transfer* message in the ownership transaction **after** it has received the *Sharing* message from Master which guarantees that the directory information cannot be obsolete. Another race can occur when Home subsequently issues invalidations to all sharers. It can so happen that Local receives an invalidation before *Data* in Figure 3 has arrived. This could result in a block fill of inconsistent data. This problem is solved as in the DASH by retrying the read-miss transaction, once the *Data* message arrives.

Handling replacements of AM blocks is a challenge to COMA protocols not apparent in CC-NUMA protocols. While AM copies that are not master copies can be simply discarded, a new master has to be nominated if the master copy has to leave room for another copy. Strategies for doing this is beyond the scope of the paper. (The interested reader is referred to [17].)

In summary, while a flat COMA is expected to have fewer global read misses than CC-NUMA machines, global read-miss transactions often involve three hops when CC-NUMA machines involve two hops. This is because Home does not keep any memory copy in general; rather the current master has to provide it. In the next section we will study how COMA protocols can use the notion of a hint to avoid the detour of read-miss requests via Home.

2.3 Hints: Avoiding Three-Hop Misses

To be able to service a global AM read-miss transaction in two hops, Local could associate with each AM block-frame an identifier—called a *hint* in [12]—of a potential master. If the hint is correct, a copy of the memory block can be retrieved in two hops. While the success rate of a hint depends on the heuristics used to guess who the current master is, we postpone the discussion of the usefulness of various hint heuristics to Section 3.2.

Let us review how the basic global read-miss transaction in Figure 3 can be changed to support hints. In Figure 5, we show the read-miss transaction flow when (a) the hint is successful and (b) when the hint is wrong. Unlike the flat COMA protocol with no hints, the read request is sent to the potential master (*Guess*). If the hint is correct, Master supplies *Data* and sends a sharing request (*Sharing)* to Home. Home updates the state of the directory entry as in the protocol with no hints. If the hint is wrong, however, the remote node that is no longer Master (R in Figure 5) forwards the read request to Home (*Failure).* Home then forwards the read request to the current Master as in the original protocol.

One could use the notion of hints for ownership requests as well. Although such transactions are sketched in [12], we have not incorporated them in our simulated protocols. Instead, the ownership requests are handled according to the transaction flow of Figure 4.

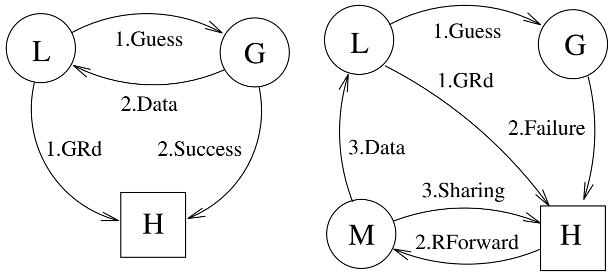
While successful hints can reduce the number of hops in global read-miss transactions by one, unsuccessful hints add another hop to the latency; at least half the guesses must be successful to reduce the overall read miss penalty. We study in the next section a new protocol extension that does not add an extra network hop when the hint is wrong.

3. A New Flat COMA Protocol using Hints

Instead of forwarding the read-miss request to Home, when the hint is wrong, Local could simultaneously send the read-miss request to both Home and the potential master. If the hint is correct, Home could drop the read-miss request. Conversely, if the hint is wrong, the incorrectly inferred master could discard it. This is the general idea of the new flat COMA hint protocol extension. We present in Section 3.1 how the global read-miss transaction has to be changed for the new protocol. Then in Section 3.2 we discuss previously proposed hint heuristics as well as proposing a new one.

3.1 Protocol Transactions

In Figure 6a, the flow of a global read-miss transaction in the case of a successful hint is shown. A global read-miss request (*GRd*) is sent from Local to Home. This request



(a) **(b)**

Figure 6: Flow of a read-miss transaction for the new protocol. (a) Successful hint (b) Unsuccessful hint.

carries the identity of the hint. Simultaneously, a *Guess* message is sent to a potential master (G in Figure 6a). If G has a copy of the block, it will respond with *Data* to Local and Home is notified by the *Success* message so that it updates the directory state. Since the identity of the guessed master is contained in the *GRd* message sent to Home, Home can drop the message if G has a copy. The flow of an unsuccessful global read-miss transaction is shown in Figure 6b. If Home notices that G does not have a copy according to the directory entry, it will forward the read request (*RForward*) to the current Master which responds to the read request in the same manner as in the COMA protocols in Section 2. Therefore, an unsuccessful hint can be serviced in three hops, instead of four using the protocol in Section 2.3. Moreover, this protocol only requires that G has a copy; and not a **master** copy.

A complication arises if G sends a request to Home to give up its AM copy (replacement) and Home receives a global read-miss request before the replacement request from G has arrived at Home. Home would then conclude that the read-miss request is serviced by G and would drop the request. To solve this race condition, G sends a *Failure* message to Home if it has no copy. Home then services this request in the same way as a failure request in the original hint protocol in Section 2.3.

3.2 Hint Heuristics

Two hint heuristics, called *shared hints* and *invalid hints*, where proposed in [12]. Invalid hints consider the node that most recently invalidated the block as the Master, whereas shared hints consider the node that most recently provided a copy as the Master. To support hints, we note that the identity of the node that invalidated or supplied the block must be available in the invalidation requests or in the data replies. Moreover, shared hints associate with each AM block a $\log_2 N$ pointer, given N nodes; invalid hints can use the empty block frame because the block is invalid.

Invalid hints work well for applications in which data is supplied on a read miss from the same node that invalidated the copy prior to the miss. This situation shows up in applications with producer-consumer data where the producer will both invalidate the data and subsequently provide it to a consumer. By contrast, in applications with migratory data [11], a block will be typically invalidated by one node and subsequently supplied by another. Gupta *et al.* [12] studied the success rate of read misses using invalid hints and found that it is less than 50% for applications where migratory data dominate; for applications with producer-consumer data, the success rate was high.

Shared hints in the terminology used by Gupta *et al.* [12] consider the latest node that provided the copy as the one that is the current Master. Gupta *et al.* [12] used shared hints to optimize ownership transactions only, but did not consider it for optimizations of read-miss transactions which is in contrast to what we do in this study. To do this, we let the latest node that provided the copy act as the next one to provide it. While this heuristic is expected to work well for producer-consumer data, as do invalid hints, it is also expected to work well for migratory data when a block always migrates among nodes in the same order.

We note that the new protocol presented in this section is expected to perform better than the one in Section 2.3 because it does not introduce extra network hops when the hint is wrong. However, a successful hint requires an extra message—in essence the *GRd* message in Figure 6a. This will result in a higher traffic than the original hint protocol in Section 2.3 which as a secondary effect can increase the read miss penalty. Note, however, that the new protocol does not involve more messages in read-miss transactions as compared to the COMA protocol with no hints.

The usefulness of hints is dictated by (i) the fraction of misses that can use hints and (ii) the success rate of the hints. Whereas coherence misses can often use hints, we note that cold misses in general can not be optimized because the hint heuristics discussed require that the block has been in the AM before.

4. Simulation Methodology

In order to study the performance improvement obtained by the COMA protocols in the previous sections, we have developed detailed architectural simulation models using the CacheMire Test Bench [3]; a program-driven simulation platform for shared-memory multiprocessors. A simulator consists of two parts: (i) a functional model of multiple SPARC processors driven by a parallel program and (ii) a memory-system simulator. The processors in the functional simulator are delayed according to the timing characteristics of the memory-system simulator. Thus, an interleaving of global memory references that conforms with the target system is maintained.

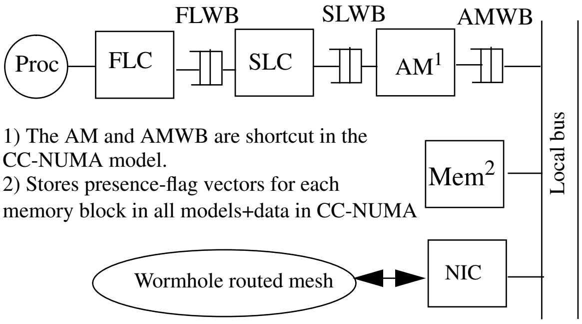


Figure 7: Simulated processing node organization.

The detailed organization of the CC-NUMA and the COMA processing node models appears in Figure 7. It consists of a SPARC processor connected to a 2 Kbyte, write-through, and direct-mapped first-level cache (FLC in Figure 7). The write buffer of the FLC (denoted FLWB) is connected to a direct-mapped second-level cache (SLC). To focus on the relative performance of our CC-NUMA and COMA models, we vary the size of the SLC in the simulations. In the CC-NUMA model, the SLC is lockup-free and copy-back and connected through its request buffer directly to the local bus. The SLC is filled from either the local memory module or from a memory module in another node. By contrast, in the COMA models, the SLC is write-through and interfaces directly to an infinitely sized attraction memory (AM) which in turn is connected to the local bus through its request buffer (AMWB). The particular location of the AM is motivated by the fact that it has to be interrogated on each SLC miss. The block size in the FLC, in the SLC, as well as in the AM is 16 bytes. The FLWB contains eight entries and the SLWB and the AMWB both contain 16 entries. In all models, a memory module stores a presence-flag vector for each memory block; in CC-NUMA, the data block for each memory block is also contained in this module.

Regarding the timing parameters, the processors are clocked at 100 MHz (1 pclock = 10 ns). We handle all instruction and private data references as if they hit in the FLC. These references and all shared data references that hit in the FLC take 10 ns to service and do not stall the processor. The SLC access time is 30 ns and an access that misses in the FLC but hits in the SLC takes 6 pclocks (including 3 pclocks to fill the FLC). The AMs in the COMA models and the fully-interleaved memory modules in the CC-NUMA model have an access time of 90 ns. The time for an FLC block fill from the AM is 18 pclocks whereas an FLC block fill from the memory module takes 30 pclocks. The difference stems from that the latter also includes two local bus accesses that each takes 60 ns.

We simulate systems of 16 processing nodes interconnected by a single 4-by-4 wormhole routed synchronous mesh[2] that is clocked at 100 MHz and with a flit size of 64 bits. A request requires two flits whereas a reply (containing data) requires six flits. It takes on average 12 pclocks and 16 pclocks to transfer a request and a reply from one node to another, respectively, in a conflict free system. We simulate contention in all parts of the machine though.

The latency involved in a global read-miss transaction depends on the initial mapping of pages among nodes. The allocation we assume maps the 4 Kbyte pages to the nodes in a round-robin fashion; consecutive virtual pages end up in nodes with consecutive node numbers. On the other hand, the latency encountered by ownership transactions are completely hidden because we assume release consistency [9] and an aggressive lockup-free second-level cache design. Finally, synchronizations use queue-based locks and we allocate a single lock per memory block to avoid false sharing.

To evaluate the performance of the implemented protocols, four benchmark programs summarized in Table 1 are used. The programs are written in C, compiled with `gcc` (version 2.1) with optimization level -O2, using ANL macros [2] to express parallelism. Three of the applications (MP3D, Water, and Cholesky) are part of the SPLASH suite [16] and the fourth application—the multigrid version of Ocean—has been provided to us from Stanford University. MP3D uses 10K particles for 10 time steps. Cholesky was run using the `bcsstk14` benchmark matrix. Water uses 288 molecules for 4 time steps, and Ocean works on a 128x128 grid with tolerance 10^{-7}. All statistics are gathered in the parallel sections.

Table 1: Benchmark programs.

Benchmark	Description
MP3D	3-D particle-based wind-tunnel simulator
Water	Water molecular dynamics simulation
Cholesky	Cholesky factorization of a sparse matrix
Ocean	Ocean basin simulator

5. Experimental Results

We first study the relative performance of the various protocols by looking at the effects on the execution time in Section 5.1. Then in Section 5.2 we focus on the efficiency with which the protocol extensions using hints reduce the average number of network hops for read-miss transactions. Finally in Section 5.3 we study to what extent the COMA protocols with hints affect the network traffic.

2. While a separate request and reply mesh is a customary solution to avoid deadlock [14], we simplify our models by using infinite network buffers in each network switch.

5.1 Effects on the Execution Time

The experimental evaluation considers five systems. We first compare the performance of the CC-NUMA protocol according to Section 2.1 (denoted NUM) with the flat COMA protocol according to Section 2.2 with no hints (denoted COM). The execution times for the four applications on top of these systems appear in Figure 8 normalized to the execution time of COM. For each application, we consider three SLC cache sizes: 4, 16, and 64 Kbytes. To see the fraction of the execution time that stems from handling cache misses and synchronizations, we have decomposed each execution-time bar into three sections: The bottommost section is the busy time, the middle section is the read-miss penalty, and the topmost section is the time spent waiting for a lock to be granted. (The time waiting for writes to perform is eliminated because we assume release consistency [9].)

Comparing CC-NUMA with COMA with no hints with 4 Kbyte caches first, we see that the execution time for CC-NUMA is between 13% (MP3D) and 100% (Ocean) longer than COMA which stems from the relative number of cold, coherence, and replacement misses in the applications. Whereas all replacement misses can be handled locally in the COMA model (we assume infinite AMs), most replacement misses result in global read-miss transactions in CC-NUMA. In Table 2, the miss rates for each application decomposed into cold, coherence and replacement misses are shown. In Ocean and Cholesky, the replacement miss component dominates the total miss rate which explains why COMA performs significantly better than CC-NUMA for these applications. By contrast, the difference in performance between CC-NUMA and COMA is smaller for MP3D where coherence misses dominate. As we consider larger SLCs, the difference between CC-NUMA and COMA vanishes as we can see in Figure 8 for the 64 Kbyte SLC systems. These results are consistent with [17].

Table 2: Cold, coherence, and replacement miss rate components for 4, 16, and 64 Kbyte SLCs.

Appl.	Cold miss rate	Coh. miss rate	Repl. m. rate (4 Kb)	Repl. m. rate (16 Kb)	Repl. m. rate (64 Kb)
MP3D	1.6%	8.9%	6.7%	6.5%	1.8%
Water	0.01%	0.2%	0.8%	0.3%	0.1%
Chol.	0.8%	0.2%	4.5%	0.8%	0.2%
Ocean	0.03%	0.6%	11%	5.8%	1.9%

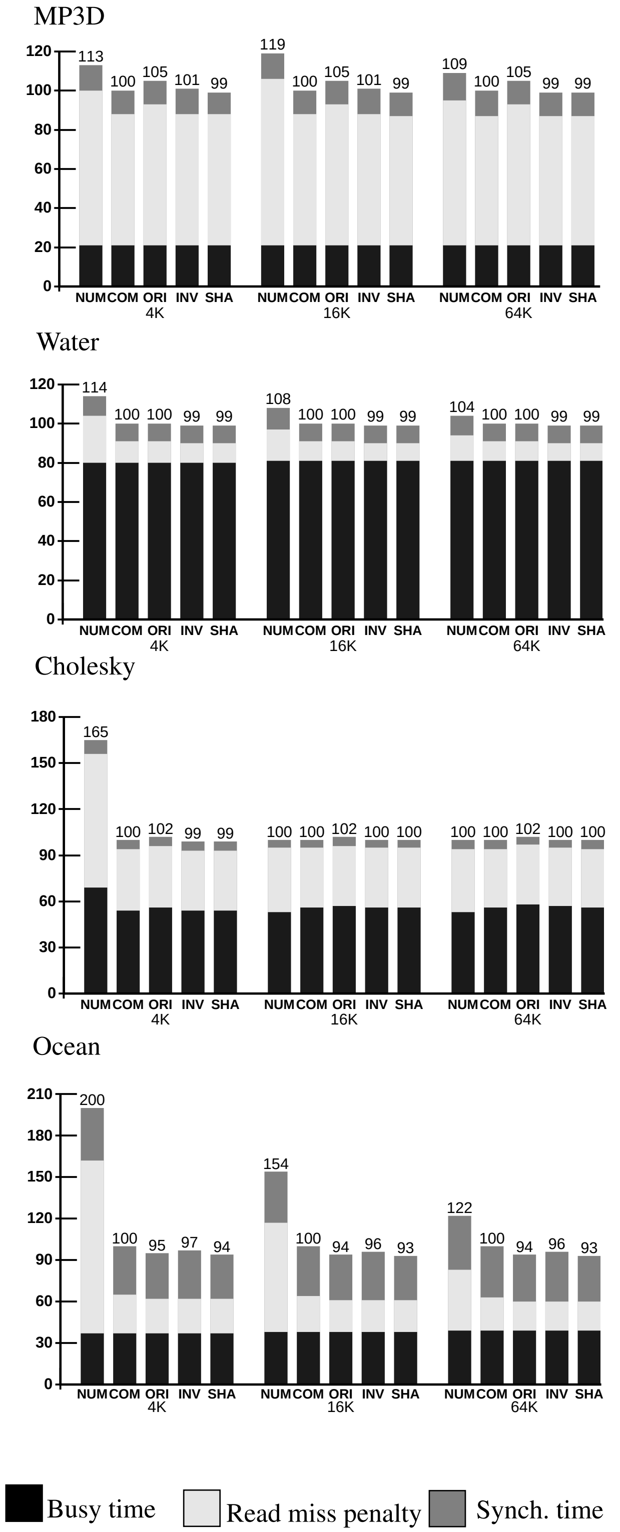


Figure 8: Execution times relative to Flat COMA with no hints for all simulated systems at various SLC cache sizes (4, 16, and 64 Kbytes).

We next consider the three COMA protocols using hints. ORI refers to the original COMA protocol that involves an extra network hop if the hint turns out to be wrong in Section 2.3 and that uses the shared hint heuristic according to Section 3.2. We do not consider the invalid hint heuristic for this protocol because of the low success rate and because of the devastating effect wrong hints have on the performance of this protocol. We also simulate the new COMA protocol in Section 3 using invalid hints (INV) and shared hints (SHA).

Starting with the original protocol using hints (ORI) and for systems with 64 Kbyte second-level caches, we see in Figure 8 that it does better than the COMA protocol with no hints (COM) only for Ocean, where the execution time is 6% shorter. Ocean contains producer-consumer data as a result of nearest neighbor communication in the multigrid solver. Therefore, shared hints have a high success rate. For the other applications, ORI exhibits mixed results; while the execution time for Water is virtually unaffected, ORI does worse than COM for MP3D and Cholesky. The fact that the execution time is between 2 and 5% longer for these applications as compared to COMA with no hints suggests that the success rate is less than 50%. Migratory data dominates in MP3D and Cholesky. The fact that ORI shows poor performance indicates that data does not migrate among nodes in the same order which would be beneficial for shared hints.

Continuing with the new protocols using shared (SHA) and invalid (INV) hints, we see that they perform somewhat better than the COMA protocol with no hints in some cases, but never worse. This is due to the fact that they do not introduce extra latency for unsuccessful hints. Both hint heuristics do best for Ocean because of its producer-consumer data dominance, although SHA does best. Although it is difficult to make out from Figure 8, SHA manages to cut the read-miss penalty in COMA with no hints by 14%. The reason why SHA is better than INV is that SHA also can shortcut misses to migratory data when the latest supplier of the data block on a miss is the same as the current one. This happens in Ocean for the barrier counter that is read and written by all processors in turn. Due to the deterministic order of how locks are granted by the queue-based lock mechanism, barrier counters tend to migrate among nodes in the same order, giving SHA an advantage over INV. This also explains why we see a shorter synchronization stall time for SHA than INV in Ocean.

Overall, although the new protocol with shared hints performs better than the original protocol with shared hints for all applications, the performance improvement over COMA without hints is significant for Ocean only. In the next two sections, we will analyze in detail the reasons for the modest improvements of hints.

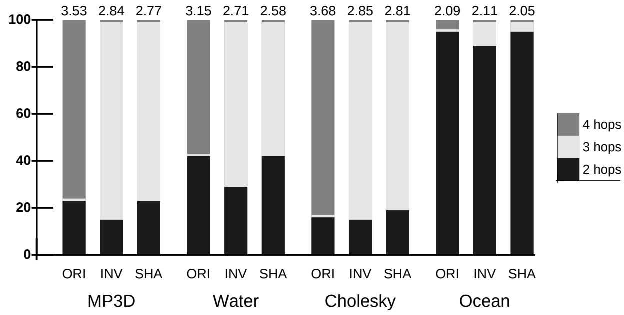


Figure 9: Distribution of 2, 3, and 4 hop read-miss transactions in the protcols using hints.

5.2 Effects on the Number of Network Hops

To analyze how successfully the new protocols using hints reduce the number of network hops, we counted the average number of hops needed to carry out each read-miss transaction using hints.

Depending on the location of Local, Home, Master, and Remote with respect to each other, read-miss transactions could be serviced in two network hops with the original COMA protocol with no hints. To separate out the effects of how successfully hints can cut the number of network hops, we charge a network hop for all requests in a read-miss transaction even if the source and the destination is the same node. For example, if Local and Home refer to the same node, we charge a network hop for the global read-miss request (*GRd*) in Figure 3. Therefore, the average number of hops for coherence misses in the original COMA protocol with no hints is three.

In Figure 9, we show the average number of network hops needed for misses using hints on top of each bar for the three COMA protocols. Considering the original protocol using hints (ORI) first, we note that it requires 3.53, 3.15, and 3.68 hops for MP3D, Water, and Cholesky, respectively; more hops are needed than in the protocol without hints. To understand why this is the case, we also show in Figure 9 the fraction of misses using hints that require two, three, and four hops. While we also record the miss transactions that have to be retried due to other pending coherence actions on a block, such transactions contribute marginally to the statistics. From the distribution, we clearly see that a majority of the misses in ORI need four hops. These four-hop miss transactions stem from unsuccessful hints. Because of the apparent low successful rate of hints, ORI does worse than the COMA protocol with no hints for MP3D and Cholesky. The read-miss penalty in Water is a small fraction of the overall execution time; hence unsuccessful hints have a marginal effect.

Looking at the new protocols using hints (SHA and INV), we see that the average number of hops is lower than three for all applications. Specifically, virtually all four-hop misses have been wiped out because a miss request exploiting hints is sent to the potential master as well as to Home in these protocols, simultaneously. Unfortunately, because of the low success rate of hints, most misses still need three hops in all applications except Ocean which is why we see modest improvements in the execution times. Another important observation is that shared hints do consistently better than invalid hints for all applications; the fraction of two-hop misses is higher in ORI and SHA than in INV.

5.3 Effects on Network Traffic

One negative effect of the new hint protocols as compared to the original hint protocol is the extra message needed for each read-miss transaction. These messages increase traffic and could increase the contention which as a secondary effect could affect the read-miss penalty. To study whether this is a significant effect, we first measured the average bandwidth needed, measured in Mbytes per second, for each application which we show in Figure 10 for the COMA protocol with no hints. We see that MP3D requires more than twice the bandwidth of the other applications. It appears that MP3D and Cholesky are the only applications where the network could saturate.

In Figure 11 we show the total network traffic for each application and for the different protocols with hints, normalized to the traffic of the COMA with no hints assuming 4 Kbyte SLCs. Whereas the original protocol using hints requires (ORI) at most 6% more traffic than the protocol with no hints, the new protocols using hints do not require significantly more traffic. The only case where the traffic gets significantly higher is for the Water application under the INV protocol. Fortunately, Water needs considerably less bandwidth than the other applications as we see in Figure 10 so the extra traffic is not an important issue.

Comparing the traffic of ORI and SHA, we see that the difference is typically less than 10%, indicating that the traffic increase due to extra messages sent in SHA protocols is negligible compared to the traffic caused by other coherence messages. Finally, if we compare the traffic

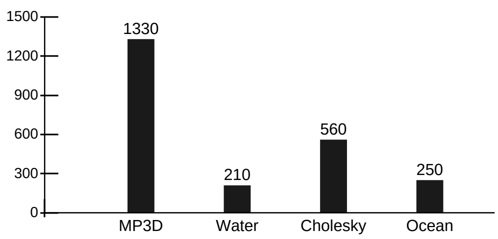


Figure 10: Bandwidth need for the COMA protocol with no hints.

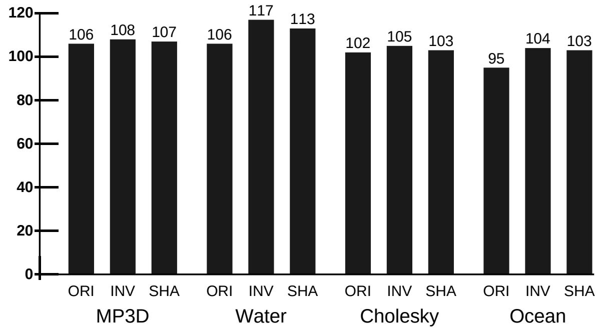


Figure 11: Network traffic relative to the COMA protocol with no hints (100%).

caused by INV and SHA, we see that SHA does better thanks to its higher success rate for hints.

6. Concluding Remarks

Flat COMA protocols manage to remove global read-miss transactions for replacement cache misses because of the attraction memories which act as huge main-memory caches. Unfortunately, other miss types such as coherence misses may involve as many as three network traversals because the directory must be interrogated before a copy can be located.

In this paper, we have studied how COMA protocols can use hints to find the node that keeps a copy without interrogating the directory. This can cut the number of network traversals by one. The first contribution is a new protocol for using hints that does not introduce extra network traversals if the hint is wrong. Secondly, we propose a new hint heuristic that considers the last node that provided the copy as the one that is going to provide a copy when the next attraction-memory read miss is encountered.

We evaluate these new protocols and compare their performances with previously proposed COMA protocols using detailed architectural simulations and four applications. Our new protocol with the enhanced hint heuristic performs better than previous COMA protocols, and the read-miss penalty is improved by 14% for one out of the four applications. For the other three applications, however, the improvement is marginal. The reasons for this are the low fraction of read misses that can use hints and the low success rate of the hint heuristic. While the hint heuristics seem successful for producer-consumer data, they perform poorly under migratory sharing which seems especially hard to deal with. In addition, since a protocol that exploits hints is more tricky and because it also needs some extra state in terms of storage for the identity of a potential holder of a copy, we feel that the improvement in performance that hints can provide does not justify the cost.

Acknowledgments

We want to thank our colleague Håkan Grahn and the anonymous reviewers for numerous comments on an earlier version of this paper.

References

[1] Agarwal, A. *et al*. "The MIT Alewife: A Large-Scale Distributed-Memory Multiprocessor," in *Proceedings of the 1st Workshop on Scalable Shared-Memory Multiprocessors.* Kluwer Acedemics, 1991.

[2] Boyle, J. et al. "*Portable Programs for Parallel Processors*". Holt, Rinehart, and Winston Inc. 1987.

[3] Brorsson, M., Dahlgren, F., Nilsson, H., and Stenström, P. "The CacheMire Test Bench - A Flexible and Effective Approach for Simulation of Multiprocessors," in *Proceedings of the 26th Annual Simulation Symposium*, pp. 41-49, March 1993.

[4] Burkhardt III, H. et al. "*Overview of the KSR1 Computer System*." Technical Report KSR-TR-9202001, Kendall Square Research, Boston, February 1992.

[5] Dahlgren, F., Dubois, M., and Stenström, P. "Fixed and Adaptive Sequential Prefetching in Shared-Memory Multiprocessors," in *Proceedings of 1993 International Conference on Parallel Processing*, Vol. I, pp. 56-63, 1993.

[6] Dahlgren, F., Dubois, M., and Stenström, P. "Combined Performance Gains of Simple Cache Protocol Extensions," in *Proceedings of the 21th Annual International Symposium on Computer Architecture*, pp. 187-197, April 1994.

[7] Dahlgren, F. and Stenström, P. "Using Write Caches to Improve Performance of Cache Coherence Protocols in Shared-Memory Multiprocessors," accepted for publication in *Journal of Parallel and Distributed Computing*. June 1994.

[8] Gharachorloo, K., Gupta, A., and Hennessy, J. "Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 15-26, May 1990.

[9] Gharachorloo, K., Gupta, A., and Hennessy, J. "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors," in *Proceedings of ASPLOS IV*, pp. 245-257, April 1991.

[10] Grahn, H., Dubois, M., and Stenström, P. "Implementation and Evaluation of Updata-Based Cache Protocols Under Relaxed Memory Consistency Models," accepted for publication in *Future Generation Computer Systems,* July 1994.

[11] Gupta, A. and Weber, W-D. "Cache Invalidation Patterns in Shared-Memory Multiprocessors," in *IEEE Transaction on Computers,* 41(7), pp. 794-810, July 1992.

[12] Gupta, A., Joe, T., and Stenström, P. "*Performance Limitations of Cache-Coherent NUMA and COMA Multiprocessors and the Flat-COMA Solution*," Technical report, Stanford University, CSL-TR-92-524, October 1992.

[13] Hagersten, E., Landin, A., and Haridi, S. "DDM - A Cache-Only Memory Architecture," In *IEEE Computer*, Vol. 25, No. 9, pp. 44-54, September 1992.

[14] Lenoski, D. *et al.* "The Stanford Dash Multiprocessor," in *IEEE Computer*, Vol. 25, No. 3, pp. 63-79, March 1992.

[15] Mowry, T. and Gupta, A. "Tolerating Latency through Software-Controlled Prefetching in Scalable Shared-Memory Multiprocessors," in *Journal of Parallel and Distributed Computing*, pp. 87-106, June 1991.

[16] Singh, J.P., Weber, W.-D., and Gupta, A. "SPLASH: Stanford Parallel Applications for Shared-Memory," in *Computer Architecture News*, 20(1):5-44, March 1992.

[17] Stenström, P., Joe, T., and Gupta, A. "Comparative Performance Evaluation of Cache-Coherent NUMA and COMA Architectures," in *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 80-91, May 1992.