The Use of Prediction for Accelerating Upgrade Misses in cc-NUMA Multiprocessors

Manuel E. Acacio, José González[†], José M. García and José Duato[‡]
Universidad de Murcia, Spain. E-mail: {meacacio,jmgarcia}@ditec.um.es
[†]Intel Barcelona Research Center, Intel Labs, Barcelona. E-mail: josex.gonzalez.gonzalez@intel.com
[‡]Universidad Politécnica de Valencia, Spain. E-mail: jduato@gap.upv.es

Abstract

This work is focused on accelerating upgrade misses in cc-NUMA multiprocessors. These misses are caused by store instructions for which a read-only copy of the line is found in the L2 cache. Upgrade misses require a message sent from the missing node to the directory, a directory lookup in order to find the set of sharers, invalidation messages being sent to the sharers and responses to the invalidations being sent back. Therefore, the penalty paid by these misses is not negligible, mainly if we consider that they account for a high percentage of the total miss rate. We propose the use of prediction as a means of providing cc-NUMA multiprocessors with a more efficient support for upgrade misses by directly invalidating sharers from the missing node. Our proposal comprises an effective prediction scheme achieving high hit rates as well as a coherence protocol extended to support the use of prediction. Our work is motivated by two key observations: first, upgrade misses present a repetitive behavior and, second, the total number of sharers being invalidated is small (one, in some cases). Using execution-driven simulations, we show that the use of prediction can significantly accelerate upgrade misses (latency reductions of more than 40% in some cases). These important improvements translate into speed-ups on application performance up to 14%. Finally, these results can be obtained including a predictor with a total size of less than 48 KB in every node.

1 Introduction and Motivation

The user's view of a shared-memory system is elegantly simple: all processors read and modify data in a single shared store. This makes shared-memory multiprocessors preferable to message-passing multicomputers from the user's point of view. Most shared-memory multiprocessors accelerate memory accesses using per-processor caches. Caches are usually transparent to software through a cache coherence protocol. Directory-based coherence protocols (cc-NUMA multiprocessors) offer a scalable performance path beyond snooping-based ones (SMP designs) by allowing a large number of processors to share a single global address space over physically distributed memory. The main difficulty in such designs is to implement the cache coherence protocol in such an efficient way that minimizes L2 miss latencies.

Even with non-blocking caches and out-of-order processors, previous studies have shown that the relatively long L2 miss latency found in cc-NUMA multiprocessors constitutes a serious hurdle to performance [25], and, as recently stated by Hill [12], relaxed consistency models do not reduce this long penalty sufficiently to justify their complexity. Thus, there are compelling reasons to examine transparent hardware optimizations.

In our previous work [2], we proposed an architecture especially designed to reduce the long L2 miss latencies that characterize cc-NUMA designs by means of integrating some key components into the processor die. Our proposal significantly reduced the latency of all the types of L2 misses but one, the so-called *upgrade misses*. Unfortunately, we found several applications for which an important fraction of the L2 misses falls into this category (more than 30% of the total miss rate in several cases). Upgrade misses are caused by a store instruction that finds a read-only copy of the line in the L2 cache. For this kind of misses, the L2 cache already has the valid data and only needs exclusive ownership. The directory must invalidate all the copies of the memory line but the one held by the requesting processor. This way, the directory must first determine the identity of the sharers by accessing directory information. Then, invalidation messages are created and sent. Only when the directory has received all the replies to the invalidations, the requesting node is returned the ownership of the line. This scenario is illustrated in Figure 1, left.

In this work, we focus on reducing the negative impact of upgrade misses by means of avoiding the directory indirection. As shown in Figure 1, right, upgrade misses could be significantly accelerated if instead of going through the directory, sharers were directly invalidated from the requesting node. This way, on an upgrade miss, the faulting node would predict current sharers for the line and would directly send invalidation messages to them. In order to check the prediction, the upgrade miss is also sent to the directory. However, differently from a conventional coherence protocol, now the directory is accessed in parallel with the invalidation process. If all sharers were correctly predicted (which is the case presented in Figure 1), the directory would immediately give the ownership of the line. On the contrary, if any of the real sharers was not included in the list of predicted sharers, the directory would inval-

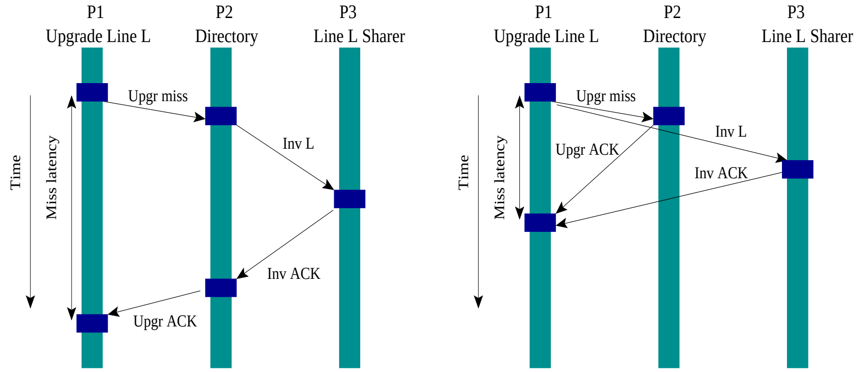

Figure 1: Coherence operations for an upgrade miss in a conventional cc-NUMA (left) and in a cc-NUMA including prediction (right)

idate it before replying to the requesting node. This node assumes the ownership of the line once it has received all the responses from the predicted nodes as well as the acknowledgment from the directory. It is important to note that, correctly predicting the sharers of a certain memory line would make the access to the slow DRAM directory be performed in parallel with the invalidation of the sharers, significantly reducing the number of cycles employed by directories to satisfy upgrade misses. On the other hand, when prediction fails the miss will be satisfied as usual and its latency could be slightly affected as a result of the increase on network and cache controller occupancies to create, send and process invalidation messages and their associated responses.

Our proposal is based on the observation that upgrade misses present a repetitive behavior: the nodes that are invalidated on an upgrade miss for a certain memory line will be invalidated again on future upgrade misses for the same line. Also, as previously pointed [5], we have observed that the number of sharers that must be invalidated after an upgrade miss is very small (one, in some cases), which reveals that a more efficient management of this kind of misses could be effective. Therefore, a well-tuned prediction engine could be employed to capture the repetitive behavior of upgrade misses, significantly accelerating them by removing the access to the directory from the critical path and, thus, improving the performance of cc-NUMA architectures.

Our baseline machine is a conventional sequentially consistent cc-NUMA multiprocessor implementing a write invalidate coherence protocol, such as the state-of-the-art SGI Origin 2000 [18]. Two main elements are developed in order to provide prediction: first, an effective prediction engine able to find out the identity of current sharers in case of an upgrade miss and, second, a coherence protocol designed to support the use of prediction. The proposed prediction scheme is an *address-based* predictor, accessed on an upgrade miss to obtain the identity of up to three potential sharers. As we will later show, this predictor usually obtains very high hit rates. Regarding the new coherence protocol, the main goal is to provide prediction by introducing the minimum number of changes into an already existing coherence protocol.

We observe two main contributions of this work. First, we propose a prediction scheme and extend a four-state MESI coherence protocol to support prediction. The use of prediction can significantly reduce the latency of upgrade misses up to 45%, which translates into speed-ups on application performance up to 14%. Second, we show how these performance advantages could be reached using a prediction scheme with a total size of less than 48 KB in every node.

The rest of the paper is organized as follows. The related work is given in Section 2. Section 3 shows the prediction scheme that we propose. The extended coherence protocol is presented and justified in Section 4. Section 5 shows a detailed performance evaluation of our novel proposal. Finally, Section 6 concludes the work.

2 Related Work

Snooping and directory protocols are the two dominant classes of cache coherence protocols for hardware shared-memory multiprocessors. Snooping systems (such as the Sun UE1000 [4]) use a totally ordered network to directly broadcast coherence transactions to all processors and memory. This way, lower latencies than directory protocols are achieved for upgrade misses (for all sharing misses in general). Unfortunately, the energy consumed by snoop requests, snoop bandwidth limitations and the need to act upon all transactions at every processor, make snooping-based designs extremely challenging, especially in light of aggressive processors with multiple outstanding requests. In contrast, directory protocols transmit coherence transactions over an arbitrary point-to-point network to the corresponding home directories which, in turn, redirect them to the processors caching the line. The consequences are that directory systems (such as the SGI Origin 2000 [18]) can scale to large configurations, but they have higher unloaded latency because of the overheads of directory indirection and message sequencing. Therefore, many research efforts have been focused on studying techniques to reduce the usually long L2 miss latencies that characterize cc-NUMA architectures.

Several works, as [6], [20] and [28], have tried to reduce the latency of upgrade misses by using a multicast scheme to make the invalidation of the sharers. However, the small number of messages sent per invalidation event [9] precludes multicast schemes from obtaining any significant advantage.

The new Compaq AlphaServer GS320 [7] constitutes an example of a cc-NUMA architecture specifically targeted at medium-scale multiprocessing (up to 64 processors). The hierarchical nature of its design and its limited scale make it feasible to use simple interconnects, such as a crossbar switch, to connect the handful of nodes, allowing a more efficient handling of upgrade misses than traditional directory-based multiprocessors by exploiting the extra ordering properties of the switch to eliminate explicit invalidation acknowledgments. On the contrary, our proposal does not require any interconnection network with special ordering.

Apart from reducing the memory overhead entailed by directories in cc-NUMA architectures ([1], [10] and [24]), caching directory information has also been proposed as a technique to minimize directory access time and, consequently, L2 miss latency ([2][22]). Unfortunately, upgrade misses (conversely to other types of misses) could not take significant benefit from finding directory information in a fast directory cache even if it is included into the processor die [2]. The reason is that on an upgrade miss, invalidation messages must be created and sent and the directory must wait until all responses from the sharers have been received to conclude the miss. Now, we try to optimize upgrade misses by doing the access to directory information at the same time that invalidation messages are going to their destinations.

Prediction has a long history in computer architecture and it has proved useful in improving microprocessor performance. Prediction in the context of shared memory was first studied by Mukherjee and Hill, who showed that it is possible to use address-based[1] 2-level predictors at the directories and caches to track and predict coherence messages [21]. Subsequently, Lai and Falfasi modified these predictors to reduce their size and showed how they can be used to accelerate reading of data [16]. Finally, Kaxiras and Young [15] used prediction to reduce access latency in distributed shared-memory systems by attempting to move data from their creation place to their use points as early as possible.

Another related technique to reduce coherence overhead resulting from the invalidation of shared lines was originally presented in [19]. Some heuristics are applied to allow each processor to detect and *self-invalidate* those shared lines that will not be probably accessed in the future. Subsequently, Lai and Falsafi [17] proposed Last-Touch Predictors (LTPs) to improve self-invalidation accuracy. Contrary to our prediction scheme that could be implemented using less than 48 KB, LTPs require much more memory to store their signatures (encoding of program traces used to detect last-touches to every line).

Alternatively, Kaxiras and Goodman [14] and Nilsson and Dahlgren [23] examined a hardware technique to detect and tag loads in the instruction stream that are followed by a store to the same address. The idea was to reduce L2 miss rate by converting a load miss into a coherent write. Note that, this technique allows saving the load miss (which would be converted into the subsequent store miss) but it does not accelerate upgrade misses at all.

Bilir et al. [3] investigated a hybrid protocol that tries to achieve the performance of snooping protocols and the scalability of directory-based ones. The protocol is based on predicting which nodes must receive each coherence transaction. If the prediction hits, the protocol approximates the snooping behavior (although the directory must be accessed in order to verify the prediction). Performance results in terms of execution time were not reported and the design was based on a network with a completely ordered message delivery, which could restrict its scalability. Our work focuses on reducing the latency of upgrade misses by means of predicting the sharers of the memory line. We can take advantage of any of the current and future high-performance point-to-point networks and it could be incorporated into cc-NUMA multiprocessors with minimal changes in the coherence protocol.

3 Predictor Design for Upgrade Misses

The first component of our proposal is an effective prediction scheme that allows each node of a cc-NUMA multiprocessor to *guess* the sharers of a certain memory line once an upgrade miss for the line is triggered.

Figure 2 illustrates the architecture of the prediction scheme that we propose and evaluate in this work. The scheme consists of a *prediction table* which, on an upgrade miss, provides a list of the nodes supposed to have a read-only copy of the memory line and an *Invalidated Lines Table* (*ILT*) used to save the addresses of some of the memory lines for which a *predicted* invalidation was received. The latter is done to ensure the correctness of the coherence protocol, as will be discussed in Section 4.

The prediction table is indexed using the effective memory address that caused the miss. Therefore, this is an example of an *address-based* predictor [14]. As shown in Figure 2, each entry in the prediction table consists of a set of pointers (three in our particular case), each one with its corresponding confidence bits (a 2-bit up-down saturating counter). These pointers are used to encode the identity of the sharers for a particular line (a subset of them when more than three sharers are found). This way, each one of the entries needs $(3 \times log_2 N + 6)$ bits, for a N-node system.

An important design decision is the maximum number of pointers that each entry must provide. Having a small number of pointers could make all the potential sharers for a line not be included. On the other hand, having N 2-bit up-down saturating counters per entry, for a N-node system, would increase the width of the predictor and, consequently, its final size. Earlier studies have shown that most of the time only a few caches have a copy of a line when it is written [5]. We observed that having three pointers per line is enough to store the identity of all the potential sharers for most of the cases. Also, it is important to note that small benefits can be expected from applying prediction to lines that are widely shared. Thus, we think that providing three pointers per entry constitutes a good compromise.

The prediction table is implemented as a non-tagged table and works as follows: initially, all the entries store value 0 into the 2-bit saturating counters. On each upgrade miss, the predictor is probed. The miss is predicted when at least one of the *2-Bit Counter* fields provides confidence (values of 2 or 3). In this case, invalidation messages are

[1] Address-based stands for predictors whose table is accessed using the effective memory address.

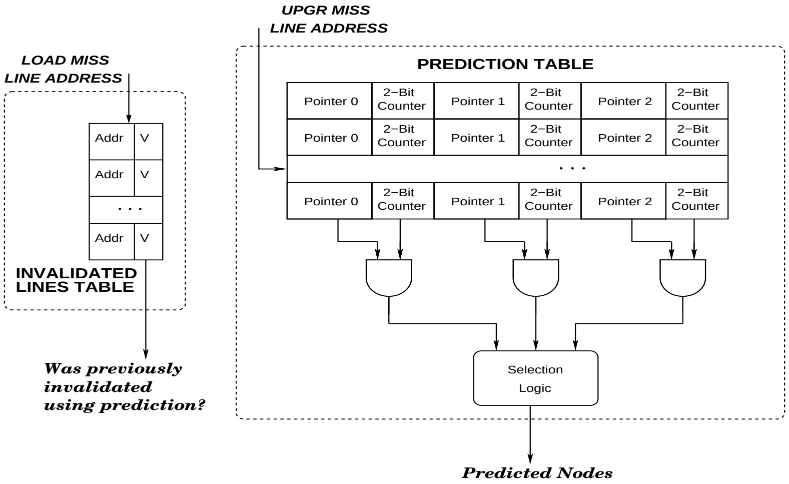

Figure 2: Anatomy of the prediction scheme proposed in this work

sent to the nodes indicated by those pointers whose *2-Bit Counter* fields store values of 2 or 3. On the responses, predictions are verified. The reply message received from the directory for an upgrade miss contains the list of those nodes that actually had a copy of the line. Whereas, the predictor holds a list of three potential sharers. Confidence counters associated with the pointers of those nodes that appear in both lists are incremented. On the other hand, confidence bits of those pointers whose nodes are not present in the list of actual sharers are decremented. Finally, those sharers that do not have an associated pointer are added with a confidence value of 2 as long as free pointers are available (confidence values of 1 or less). An additional optimization is also included as a consequence of the observation made by Gupta and Weber in [9]. The authors found migratory sharing as causing an important percentage of the upgrade misses. Migratory sharing arise when shared data is modified in turn by several processors. We are aware of this fact by updating the predictor each time the read-only copy of a certain memory line is not received from the corresponding home directory but from the owner of the line through a cache-to-cache transfer. This way, the owner of the line is considered a potential sharer.

The *ILT* structure is used to store the address of some of the lines for which a predicted invalidation was received and that have not been referenced since then. This information is originally stored into the L2 cache entry when the line is invalidated (by setting the corresponding bit). Only when this cache entry is subsequently assigned to a different memory line (and the corresponding bit is still set), this information is moved to the *ILT* table. Therefore, each entry at the *ILT* table consists of the address of a memory line and a presence bit indicating whether the entry is currently being used or not.

4 Coherence Protocol Supporting Prediction

Some modifications must be included into the coherence protocol in order to make use of the above prediction scheme. Our starting point is an invalidation-based, four-state MESI coherence protocol similar to the one included in the SGI Origin 2000 [18]. Two main premises guided our design decisions: first, to keep the resulting coherence protocol as close as possible to the original one, avoiding additional race conditions, and second, to assume sequential consistency [12]. As in [5], we use the following terminology for a given memory line:

- The *directory node* is the node in whose main memory the block is allocated (also known as *home node*).
- The *sharer nodes* are those nodes that hold read-only copies of the line.
- The *requesting node* is the node containing the L2 cache that issues an upgrade miss for the line.

When an upgrade miss for a certain memory line occurs, the predictor implemented into the cache controller of the requesting node is accessed. If an entry for the line is not found or it is not confident, the miss is sent to the directory node, where is satisfied as usual. Otherwise, the upgrade miss is *predicted* and the following actions are undertaken:

Requesting Node Operation (1). Sends invalidation messages to the nodes predicted to have read-only copies of the line (sharer nodes). Each message includes a bit identifying the invalidation as *predicted*. In order to verify the prediction, the miss is also sent to the corresponding home directory (a list of the predicted nodes is included into the message).

Directory Node Operation. When a *predicted* upgrade miss for a line in the *Shared* state is received, the directory node checks if all current sharers were predicted and, thus, invalidated. If any (or some) of the sharers was not predicted, the directory invalidates it (or them) as for a non-predicted upgrade miss. Once the non-predicted sharers have been invalidated[2] (if any), the directory updates the entry associated with the line (now the requesting node has the ownership of the line) and sends the reply message giving the exclusive ownership of the line to the requesting node. If a *predicted* upgrade miss is received for a line that is not in the *Shared* state, something preceding the upgrade miss took place. In this case, the miss is converted into non-predicted and processed as usual.

[2] The directory has received the corresponding replies to the invalidations from them.

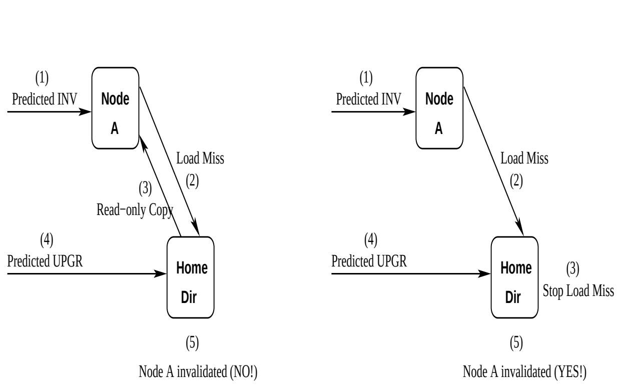

Figure 3: Case in which the first scenario occurs (left) and how is solved (right)

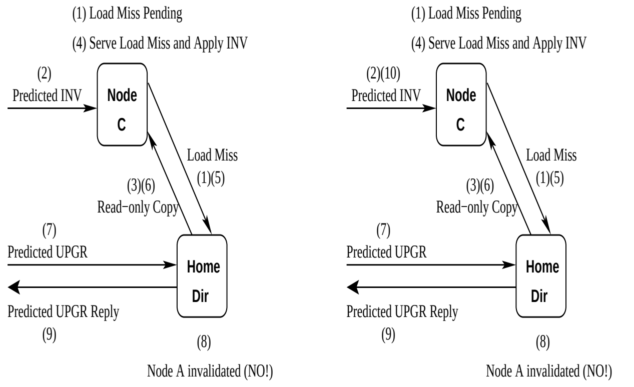

Figure 4: Case in which the second scenario occurs (left) and how is solved (right)

Sharer Node Operation. When a *predicted* invalidation for a certain memory line comes to the L2 cache controller, the line is searched into the L2 cache. One of these situations could occur:

1. The line is not found in the L2 cache. In this case, a *nack* message is returned to the requesting node informing that the prediction for this node missed. Additionally, if the predicted node has a pending load miss for the line, the invalidation is noted down in the *mshr* associated with the miss (as it would be done in the normal case).

2. The line is found to be in the *Exclusive* or *Modified* states. In this case, this node requested a read-write copy of the line and the directory satisfied this miss before the predicted one. Thus, a *nack* must be returned to the requesting node.

3. The line is found to be in the *Shared* state. This is the case of a prediction hit. There are three possible situations that should be considered:

   3.1 The node has not a pending upgrade for the line. In this case, the line is immediately invalidated and an *ack* reply is sent to the requesting node. Besides, the line is marked as having been invalidated by a remote node (not by the directory) by setting a bit (the *Predicted bit*) into the corresponding L2 cache entry (this bit will be cleared in the next miss at that line). Later on, if this L2 cache entry were used to store a different memory line, an entry for the previously invalidated line would be allocated in the *Invalidated Lines Table* (*ILT*) whenever the *Predicted bit* is still set. If all entries in the *ILT* table are being used at this moment, one of them would be used (a kind of replacement) and a load miss would be forced for the memory line associated with the removed entry. An entry for a certain memory line in the *ILT* is freed as soon as a L2 cache miss appears for that line.

   3.2 The node has a pending upgrade for the line but the reply from the directory has not arrived yet. In this case, the line is invalidated and an *ack* reply returned.

   3.3 The node has a pending upgrade for the line and the reply from the directory has already arrived. In this case, this node is becoming the new owner for the line, so the line must not be invalidated and, thus, a *nack* message is sent to the requesting node.

Requesting Node Operation (2). Once the requesting node has received all the *ack*/*nack* replies from the predicted nodes as well as the reply from the directory, the miss *would seem* to have been completed. If the miss was not satisfied as an upgrade miss (due to a different upgrade miss was first served) the requesting node can now assume the exclusive ownership of the line, since it has already been guaranteed by the directory. In case of an upgrade miss we can not be completely sure all shared copies of the line have been invalidated. There are two possible scenarios in which this is not actually true.

The first scenario is illustrated in Figure 3 and arises when one of the *predicted* sharers (say node *A*) requests a read-only copy of the previously invalidated line (due to a load miss) and this request reaches the directory before the *predicted* upgrade miss that caused the invalidation. The directory would satisfy the load miss as usual[3] and, afterwards, on processing the predicted upgrade miss, it would assume that the last copy on node *A* has been invalidated by the requesting node. Fortunately, this situation would be easily avoided if node *A* notifies the directory that the load miss is for a line for which a *predicted* invalidation has previously been received (remember that this information was stored into the L2 entry for the line when the predicted invalidation was applied or into the *ILT* table if the entry was subsequently used). This is done by setting a particular bit into the load miss message. This way, when the directory receives a load miss that has been originated after a *predicted* invalidation for certain line, it will stop the load miss until the corresponding *predicted* upgrade miss has been completed (that is, until node *A* does not appear as holding the line in the sharing code associated with the memory line).

Although very infrequent and strange, a second scenario could still occur (see Figure 4). Assume the case of a pre-

[3] This is true since our coherence protocol does not implement replacement hints for lines in the *Shared* state in order to increase throughput.

	Requesting Node	**Sharer Node**	**Directory Node**
On a predicted Upgrade Miss	1. Send *INVs* to predicted nodes Send *Upgrade Miss* to Directory 2. Get *ack/nack* from predicted nodes Get *Response* from Directory 3. If it actually was an *Upgrade Miss* Invalidate those sharers confirmed by the directory that replied *nack*(if any) Assume the *ownership*	If (line in the *Shared* state) If (*pending upgrade* & have *directory reply*) Send *nack* reply else Send *ack* reply and *invalidate* the line If (!*pending upgrade*) Set *Predicted* bit If (line in the *Modified* state | line in the *Exclusive* state) Send *nack* reply If (line is *not present*) Send *nack* reply If (*pending load miss*) Note down *invalidation*	If (line in the *Shared* state) If (!*predicted all sharers*) Invalidate non-predicted Update sharing information Send response else Process as usual
On other miss type		If (*load miss* & (*Predicted* bit = 1 | has an entry in *ILT*)) Set Predicted bit on load miss If (*Predicted* bit = 1 | has an entry in *ILT*)) Clear bit or free entry If (using L2 cache entry & *Predicted* bit is set) Insert address in *ILT* table	If (*load miss* & *Predicted* bit = 1 & node found as *sharer*) Wait until *Upgrade* has been completed

Table 1: How the coherence protocol works

dicted node (say node *C*) as having a pending load miss. As previously pointed, a *nack* response would be returned to the requesting node and the invalidation would be applied after completing the load miss. However, when the *predicted* invalidation message reaches node *C*, it is unknown whether the directory is going to satisfy the pending load miss before the *predicted* upgrade miss or after it. Thus, the line can not be marked as invalidated as a consequence of a prediction and a problem could arise if the load miss reaches the directory before the *predicted* upgrade miss. In this case, it is possible for the invalidated node to incur into another load miss for the line once the reply from the directory for the former load miss has been received and the line has been invalidated as a consequence of the predicted invalidation. If this new load miss reaches the directory once the predicted upgrade miss has been served, there is no problem. Otherwise, a read-only copy of the line would be given to node *C* and, subsequently, on receiving the predicted upgrade miss, the directory would assume that the copy in node *C* has been invalidated by the *predicted* message previously sent by the requesting node. In our coherence protocol, this situation is solved by the requesting node. It takes notice of those nodes that having been predicted as sharers replied with a *nack* to the predicted invalidation. When the upgrade reply from the directory is received and all the responses for the predicted nodes have been collected, some of the nodes are re-invalidated. In particular, those from which *nack* replies were received but the directory identified them as having a copy of the line. If frequent, this re-send of invalidations could significantly increase the latency of upgrade misses. However, as we will see, this situation has been found to occur very infrequently (less than 1% of the upgrade misses for all the cases and 0% for the majority).

Table 1 summarizes all the situations that could appear as a consequence of applying prediction.

5 Performance Results and Analysis

In this section, we present a detailed performance evaluation of our proposal using extensive execution-driven simulations. First, we present the simulation environment. Next, we analyze the accuracy of the prediction scheme presented in Section 3. Finally, we demonstrate the ability of our prediction-based technique to significantly improve performance.

5.1 Simulation Environment

We have used a modified version of Rice Simulator for ILP Multiprocessors (RSIM), a detailed execution-driven simulator [13]. RSIM models an out-of-order superscalar processor pipeline, a two-level cache hierarchy, a split-transaction bus on each processor node, and an aggressive memory and multiprocessor interconnection network subsystem, including contention at all resources. The modeled system is a 16-node cc-NUMA that implements a full-map, invalidation-based, four-state MESI directory cache-coherent protocol and uses a 2-dimensional mesh to interconnect the system nodes. Second-level caches are assumed to be integrated into the processor chip (as in [11]). Table 2 summarizes the parameters of the simulated system. These parameters have been chosen to be similar to the parameters of current multiprocessors.

Probing and updating the predictors do not add any cycle. Contrary to the uniprocessor/serial-program context where predictors are updated and probed continuously with every dynamic instruction instance, we only update the prediction history and only probe the predictor to retrieve information in the case of an upgrade miss. Thus, as in [14], we believe that the predictors neither constitute a potential bottleneck nor add cycles to the critical path, because their latency can be hidden from the critical path (for example, by speculatively accessing the predictor in parallel with the L2 cache lookup). Prediction messages are created one-per-cycle.

With all these parameters, the resulting no-contention round-trip latency of a load access satisfied at various levels of the memory hierarchy is shown in Table 3.

Table 4 describes the applications we use in this study. In order to evaluate the benefits of our proposals, we have selected several scientific applications covering a variety

16-Node System Parameters	
ILP Processor	
Processor Speed	1 GHz
Max. fetch/retire rate	4
Instruction Window	64
Functional Units	2 integer arithmetic 2 floating point 2 address generation
Memory queue size	32 entries
Cache Parameters	
Cache line size	64 bytes
L1 cache (on-chip, WT)	Direct mapped, 32KB
L1 request ports	2
L1 hit time	2 cycles
L2 cache (off-chip, WB)	4-way associative, 1024KB
L2 request ports	1
L2 hit time	15 cycles, pipelined
Number of MSHRs	8 per cache
Memory Parameters	
Memory access time	70 cycles (70 ns)
Memory interleaving	4-way
Directory Cycle	10 cycles
First coherence message creation time	4 directory cycles
Next coherence messages creation time	2 directory cycles
Internal Bus Parameters	
Bus Speed	1 GHz
Bus width	8 bytes
Network Parameters	
Topology	2-dimensional mesh
Flit size	8 bytes
Non-data message size	16 bytes
Router speed	250 MHz
Arbitration delay	1 router cycle
Router's internal bus width	64 bits
Channel width	32 bits
Channel speed	500 MHz

Table 2: Base system parameters

Round Trip Access	Latency (Cycles)
Secondary Cache	19
Local	118
Remote (1-Hop)	234

Table 3: No-contention round-trip latency of load accesses

of computation and communication patterns for which upgrade misses constitute an important percentage of the total miss rate (more than 25% in all the cases). *MP3D* is from the SPLASH benchmark suite [26], *FFT* and *Ocean* are from SPLASH-2 benchmark suite [27]. *EM3D* is a shared-memory implementation of the Split-C benchmark. *Unstructured* is a computational fluid dynamics application that uses an unstructured mesh. All experimental results reported in this paper are for the parallel phase of these applications. Data placement in our programs is either done explicitly by the programmer or by RSIM which uses a first-touch policy on a cache-line granularity. Thus, initial data-placement is quite effective in terms of reducing traffic in the system.

Program	Size
EM3D	38400 nodes, degree 2, 15% remote, 25 timesteps
FFT	64K Points
MP3D	48000 nodes, 20 timesteps
Ocean	130x130 array, 10^{-7} error tolerance
Unstructured	Mesh.2K, 5 timesteps

Table 4: Applications and input sizes

Using these applications we compare, through extensive simulation runs, the base system (*Base*) with two configurations using two instances of the prediction scheme presented in this work. The first, the *UPT* configuration, includes a prediction table with an entry per each one of the memory lines and an unlimited number of entries in the *ILT* table. Of course, this predictor has a prohibitive cost, but it completely eliminates the aliasing effect caused by the non-tagged nature of the prediction table. The second, the *LPT* configuration, implements a realistic version of the former predictor. In this case, the number of entries available in the prediction table and in the *ILT* table are limited to 16K and 128 respectively, resulting in a total size of less than 48 KB. We think this is a reasonable size given that 1 MB L2 caches are being used. The access to the prediction table is carried out using the 14 bits resulting from computing the XOR between the 14 less significant bits of the line address and its most significant bits. As in [8], we use XOR-based placement to distribute the utilization of the entries in the prediction table and, thus, to reduce conflicts. Finally, due to its small number of entries, the *ILT* table is organized as a totally associative buffer structure.

5.2 Predictor Accuracy

Figure 5 presents accuracy results for the prediction scheme described in Section 2. Over the total number of upgrade misses, it shows the percentage of those for which prediction was applied and one of these situations occurred: (1) all sharers were predicted and invalidated from the requesting node (*Total Hit*), (2) some of the sharers could be predicted but not all (*Partial Hit*) and, finally, (3) other but none of the actual sharers were predicted (*Total Miss*). Also, we show the percentage of upgrade misses that did not make use of prediction (*Not Predict*) and the percentage of predictions that were not served as an upgrade miss by the directory (*Not Inv*). The latter is shown starting from 100% (since it corresponds to misses that are not upgrade misses) and takes place, for example, when two nodes try to simultaneously gain the exclusive ownership of a line in the *Shared* state held in both caches. Only one of them will be serviced as an upgrade miss whereas the other will be subsequently satisfied with a cache-to-cache transfer. However, as derived from Figure 5, this situation appears very infrequently in all the applications.

Observe from Figure 5 that the small size of the predictor used in the *LPT* configuration (less than 48 KB) is enough to virtually obtain the same accuracy results than the unbounded predictor (*UPT*). Its limited size, however, makes some of the entries be used by different memory lines, which slightly increases the number of upgrade misses for which prediction is applied. Whereas *Partial Hit* and *Total Miss* cases suffer a small growth compared to the results obtained for the *UPT* configuration, this does not affect the percentage of *Total Hit* upgrade misses.

As derived from Figure 5, the majority of upgrade misses could be completely predicted for all applications. *Total Hit* case represents more than 85% of the upgrade misses for all the applications but *FFT*, reaching almost 100% in *EM3D*, *MP3D* and *Unstructured*. For these applications, upgrade misses are concentrated on a small set of memory lines, which are frequently cached by the same nodes. For *FFT* only 60% of the upgrade misses could be predicted. The computation in *FFT* is split into two phases. The first performs the fast Fourier transform using the six-step FFT method whereas during second phase the inverse FFT is computed in order to perform a test of the output. For the majority of the memory lines, a single upgrade miss takes place in each one of the phases. This way, prediction is not applied during the first phase and the majority of the predictions are done during the second one, which explains the large percentage of upgrade misses that were not pre-

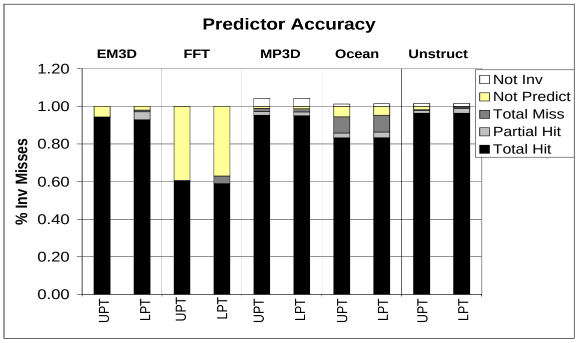

Figure 5: Percentage of upgrade misses predicted

dicted (almost 40%). Finally, note that the *Total Miss* case constitutes a very small percentage which remains lower than the *Not Predicted* case for all the applications but *Ocean*. For *Ocean*, we have found that a small subset of the upgrade misses do not present the repetitive behavior observed for the majority. The initialization of the 2-bit counters of the predictor to 2 makes these upgrade misses be predicted and, consequently, failed. Initializing the 2-bit counters to 1 instead of 2 would eliminate these prediction misses, reducing, as the negative counterpart, the number of prediction hits in favor of the *Not Predicted* case. Thus, due to the small percentage of upgrade misses completely failed (*Total Miss* case) it is preferable to have these counters initialized to 2.

5.3 Performance Analysis

Total Hit case represents those upgrade misses that could be accelerated as a consequence of using prediction. This is also true for those upgrade misses falling into the *Partial Hit* category although in less extension. Finally, *Total Miss* and *Not Inv* cases could have negative consequences on the final performance as a result of the increase on network and cache controller occupancies to create, send and process invalidation messages and their associated responses. This section analyzes the consequences that using prediction has on application performance.

Correctly predicting the sharers of a certain memory line on an upgrade miss prevents the directory from invalidating the sharers and, consequently, significantly reduces the number of cycles that the directory must dedicate to process the upgrade miss. Figure 6 illustrates the normalized average latency for upgrade misses split into network latency, directory latency and miscellaneous latency (buses, cache accesses...), for the base, *UPT* and *LPT* configurations. Network latency comprises the number of cycles the upgrade miss spends on the interconnection network. Directory latency represents the cycles needed by the directory to satisfy the miss. For *UPT* and *LPT* configurations, directory latency also includes the number of cycles between the missing node receives the reply to an upgrade miss from the directory and the response for the last predicted invalidation has been processed. Normalized average latencies are computed dividing the average latencies for each one of the configurations by the average latencies for the base case. Table 5 shows the average number of invalidations per upgrade miss sent by the directory (for the base system) and the average number of invalidations sent by the requesting node when an upgrade miss is predicted.

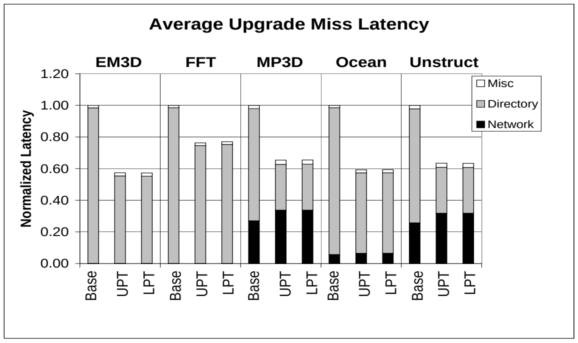

Figure 6: Normalized average *upgrade miss* latency

Application	*Base*	*UPT*	*LPT*
EM3D	1.96	1.77	2.03
FFT	1.00	1.00	1.01
MP3D	1.07	1.83	1.84
Ocean	1.13	1.25	1.26
Unstructured	1.40	1.74	1.76

Table 5: Invalidations per upgrade miss (for *Base* case) and nodes included per prediction (for *UPT* and *LPT* cases)

As observed in Figure 6, the use of the *LPT* prediction scheme significantly reduces the latency of upgrade misses for *EM3D* (45%), *MP3D* (35%), *Ocean* (41%), *Unstructured* (38%) and, in less extension, for *FFT* (25%). These improvements almost coincide with those obtained with the unrealistic predictor (*UPT*). Sending invalidations from the requesting node on an upgrade miss accelerates the invalidation process and, consequently, the component of the latency caused by the directory is significantly decreased, as shown in Figure 6. Now the invalidation of the sharers occurs in parallel with the access to the directory whereas, in a traditional cc-NUMA multiprocessor, it can not start until the directory has determined the identity of the sharers. Besides, as shown in Table 5, the regularity exhibited by upgrade misses allows our prediction scheme to obtain the high accuracy shown in Figure 5 without having to send an excessive number of unnecessary invalidation messages, which could degrade performance. Only for *Ocean* and *Unstructured* we have found that the average number of invalidation messages sent on a predicted upgrade miss is slightly greater than the average number of sharers. This is the reason for the small increase on the network component of the average upgrade miss latency observed in Figure 6 for these applications.

Application	**Load misses**	**Store misses**	
		Upgrade misses	*Rest store misses*
EM3D	66.25%	33.75%	0.00%
FFT	54.52%	45.47%	0.01%
MP3D	49.78%	47.99%	2.23%
Ocean	51.51%	40.36%	8.13%
Unstructured	37.10%	29.93%	32.97%

Table 6: Classification of the L2 misses according to the instruction that caused them

Normalized average latencies for load and store misses are also shown in Figure 7. Table 6 presents the fraction of L2 misses caused by load and store instructions. Store misses are in turn split into upgrade miss and rest of store

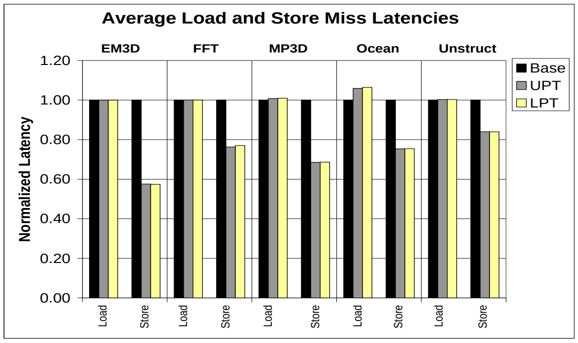

Figure 7: Normalized average load/store miss latency

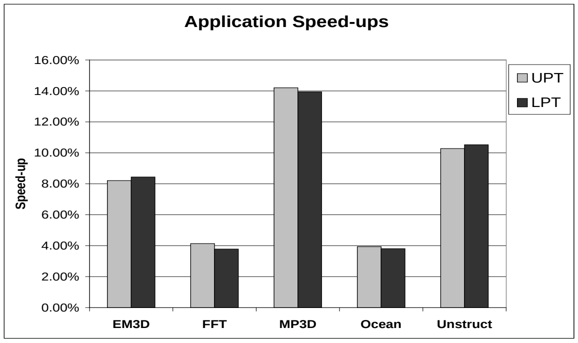

Figure 8: Application speed-ups

miss categories. As shown in Figure 7, the important benefits found for upgrade misses also lead to reductions in the average latency of store instructions. For *EM3D* and *FFT*, each store miss finds a read-only copy of line in the local L2 cache causing an upgrade miss (as derived from Table 6). Thus, reductions reported for upgrade misses are translated into store miss latency reductions (45% for *EM3D* and 25% for *FFT*). For *MP3D*, *Ocean* and *Unstructured* part of the store misses do not cause an upgrade miss and, thus, prediction can not be applied to them. As shown in Table 6, this percentage is small for *MP3D* and *Ocean* and latency reductions close to the ones reported for upgrade misses are obtained (31% for MP3D and 25% for Ocean). On the other hand, 32.97% of the store misses found in *Unstructured* do not cause an upgrade miss and latency reductions of 16% are obtained for store misses. Note also that, for all the applications but *Ocean*, our proposal does not increase the latency of load misses. *Ocean* is the application for which a greater number of predicted upgrade misses require some of the invalidation messages to be re-tried as a consequence of the scenario previously presented in Section 4. While the percentage of predicted upgrade misses for which invalidations must be re-tried (over the total number of predictions) has been observed to be less than 0.34% for *EM3D*, *FFT*, *MP3D* and *Unstructured*, it reaches 0.78% for *Ocean*. For this application, we have found that upgrade misses are closely followed by load misses from other nodes which, in case of a predicted upgrade miss with invalidations re-tried, are slightly delayed, causing the performance degradation for load misses shown in Figure 7.

The ultimate metric for application performance is the execution time. Figure 8 shows the speed-ups in execution time for *UPT* and *LPT* configurations with respect to the base system. As it can be observed, important speed-ups are derived from the use of prediction in *MP3D* (14%), *Unstructured* (11%) and *EM3D* (9%). Although upgrade misses were significantly accelerated in *Ocean* and they constitute an important fraction of the total L2 miss rate, the degradation observed for average load miss latency affects the final performance and a speed-up of 4% is obtained. Finally, the important fraction of upgrade misses that could not be predicted in *FFT* limits the speed-up found for this application to 4%.

6 Conclusions

Upgrade misses account for an important fraction of the total miss rate. This type of misses are caused by a store instruction for which the local L2 cache holds a read-only copy of the memory line and the directory is accessed in order to gain exclusive ownership of the line. This requires a message sent from the missing node to the directory, a directory lookup in order to find the set of sharers, invalidation messages being sent to the sharers and responses to the invalidations being sent back. Only when the directory has received all the replies to the invalidations, the requesting node is returned the ownership of the line and, consequently, the penalty paid by these misses is not negligible.

In this work, we propose the use of prediction to significantly accelerate upgrade misses. On an upgrade miss, the faulting node predicts current sharers for the line and directly sends invalidation messages to them. In order to check the prediction, the upgrade miss is also sent to the directory. However, differently from a conventional coherence protocol, now the access to the directory is performed in parallel with the invalidation process. If all sharers were correctly predicted, the directory would immediately give the ownership of the line. On the contrary, if any of the sharers was not sent the corresponding invalidation message, the directory would invalidate it before responding the requesting node. The requesting node assumes the ownership of the line once it has received all the responses from the predicted nodes as well as the acknowledgment from the directory. Our proposal is based on the observation that upgrade misses present a repetitive behavior. Additionally, the number of sharers that must be invalidated on an upgrade miss has been previously reported to be very small (one, in some cases), which reveals that a more efficient treatment of this kind of misses could be rewarding.

The prediction-based technique proposed in this work consists of two main components. The first is an address-based prediction engine able to find out the identity of up to three potential sharers in case of an upgrade miss. The second component is a four-state MESI coherence protocol, similar to the one used in the SGI Origin 2000, properly extended (with minimal changes) to support the use of prediction.

In order to evaluate the effects of our proposal, we executed several shared-memory applications on top of the RSIM detailed execution-driven simulator. First, we analyzed the accuracy of the proposed predictor and found that for a great percentage of the upgrade misses (which reaches almost 100% for some applications) the set of sharers was successfully predicted. Then, we analyzed our proposal in terms of its impact on application performance. The high percentage of upgrade misses correctly predicted translated into latency reductions on average of more than 40% in some cases. These benefits caused important reductions on the average latency of store instructions whereas load latency remained virtually unaffected. Finally, application execution times were accelerated up to 14%. These results can be obtained including a predictor with a total size of less than 48 KB in every node.

As part of our future work, we are performing a comparison between our technique and some other related schemes, as those presented in [14], [17] and [23].

Acknowledgments

This research has been carried out using the resources of the Centre de Computació i Comunicacions de Catalunya (CESCA-CEPBA) as well as the SGI Origin 2000 of the Universitat de Valencia. This work has been supported in part by the Spanish CICYT under grant TIC2000-1151-C07-03. José Duato is supported in part by a fellowship from the Fundación Séneca (Comunidad Autónoma de Murcia, Spain).

References

[1] M. E. Acacio, J. González, J. M. García and J. Duato. "A New Scalable Directory Architecture for Large-Scale Multiprocessors". *Proc. of the 7th Int'l Symposium on High Performance Computer Architecture*, pp. 97–106, January 2001.

[2] M. E. Acacio, J. González, J. M. García and J. Duato. "A Novel Approach to Reduce L2 Miss Latency in Shared-Memory Multiprocessors". *Proc. of the 16th Int'l Parallel and Distributed Processing Symposium*, April 2002.

[3] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill and D. A. Wood. "Multicast Snooping: A New Coherence Method Using a Multicast Address Network". *Proc. of the 26th Int'l Symposium on Computer Architecture*, pp. 294–304, May 1999.

[4] A. Charlesworth. "Extending the SMP Envelope". *IEEE Micro*, 18(1):39–49, Jan/Feb 1998.

[5] D. E. Culler, J. P. Singh and A. Gupta. *"Parallel Computer Architecture: A Hardware/Software Approach"*. Morgan Kaufmann Publishers, Inc., 1999.

[6] D. Dai and D. K. Panda. "Reducing Cache Invalidation Overheads in Wormhole Routed DSMs Using Multidestination Message Passing". *Proc. of International Conference on Parallel Processing*, 1:138–145, August 1996.

[7] K. Gharachorloo, M. Sharma, S. Steely and S. V. Doren. "Architecture and Design of AlphaServer GS320". *Proc. of International Conference on Architectural Support for Programming Language and Operating Systems*, pp. 13–24, November 2000.

[8] A. González, M. Valero, N. Topham and J. M. Parcerisa. "Eliminating Cache Conflict Misses through XOR-Based Placement Functions". *Proc. of the Int'l Conference on Supercomputing*, pp. 76–83, 1997.

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

[10] A. Gupta, W.-D. Weber and T. Mowry. "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes". *Proc. Int'l Conference on Parallel Processing*, pp. 312–321, August 1990.

[11] L. Gwennap. "Alpha 21364 to Ease Memory Bottleneck". *Microprocessor Report*, pp. 12–15, October 1998.

[12] M. D. Hill. "Multiprocessors Should Support Simple Memory-Consistency Models". *IEEE Computer*, 31(8):28–34, August 1998.

[13] C. J. Hughes, V. S. Pai, P. Ranganathan and S. V. Adve. "RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors". *IEEE Computer*, 35(2):40–49, February 2002.

[14] S. Kaxiras and J. R. Goodman. "Improving CC-NUMA Performance Using Instruction-Based Prediction". *Proc. of the 5th Int'l High Performance Computer Architecture*, pp. 161–170, January 1999.

[15] S. Kaxiras and C. Young. "Coherence Communication Prediction in Shared-Memory Multiprocessors". *Proc. of the 6th Int'l High Performance Computer Architecture*, pp. 156–167, January 2000.

[16] A. C. Lai and B. Falsafi. "Memory Sharing Predictor: The Key to a Speculative DSM". *Proc. of the 26th Int'l Symposium on Computer Architecture*, pp. 162–171, May 1999.

[17] A. C. Lai and B. Falsafi. "Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction". *Proc. of the 27th Int'l Symposium on Computer Architecture*, pp. 139–148, May 2000.

[18] J. Laudon and D. Lenoski. "The SGI Origin: A ccNUMA Highly Scalable Server". *Proc. of the 24th Int'l Symposium on Computer Architecture*, pp. 241–251, June 1997.

[19] A. R. Lebeck and D. A. Wood. "Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors". *Proc. of the 22nd Int'l Symposium on Computer Architecture*, pp. 48–59, June 1995.

[20] M. P. Malumbres, J. Duato and J. Torrellas. "An Efficient Implementation of Tree-based Multicast Routing for Distributed Shared-Memory Multiprocessors". *Proc. of the 8th Int'l Symposium on Parallel and Distributed Processing*, pp. 186–189, 1996.

[21] S. S. Mukherjee and M. D. Hill. "Using Prediction to Accelerate Coherence Protocols". *Proc. of the 25th Int'l Symposium on Computer Architecture*, pp. 179–190, July 1998.

[22] A. K. Nanda, A.-T. Nguyen, M. M. Michael and D. J. Joseph. "High-Throughput Coherence Controllers". *Proc. of the 6th Int'l High Performance Computer Architecture*, pp. 145–155, January 2000.

[23] J. Nilsson and F. Dahlgren. "Reducing Ownership Overhead for Load-Store Sequences in Cache-Coherent Multiprocessors". *Proc. of the 14th Int'l Parallel and Distributed Processing Symposium*, pp. 684–692, May 2000.

[24] B. O'Krafka and A. Newton. "An Empirical Evaluation of Two Memory-Efficient Directory Methods". *Proc. of the 17th Int'l Symposium on Computer Architecture*, pp. 138–147, May 1990.

[25] V. S. Pai, P. Ranganathan, H. Abdel-Shafi and S. Adve. "The Impact of Exploiting Instruction-Level Parallelism on Shared-Memory Multiprocessors". *IEEE Transactions on Computers*, 48(2):218–226, February 1999.

[26] J. Singh, W.-D. Weber and A. Gupta. "SPLASH: Stanford Parallel Applications for Shared-Memory". *Computer Architecture News*, 20:5–44, March 1992.

[27] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". *Proc. of the 22nd Int'l Symposium on Computer Architecture*, pp. 24–36, June 1995.

[28] Z. Zhou, W. Shi and Z. Tang. "A Novel Multicast Scheme to Reduce Cache Invalidation Overheads in DSM Systems". *Proc. of the 19th IEEE Int'l Performance, Computing and Communications Conference*, 2000.