A Direct Coherence Protocol for Many-Core Chip Multiprocessors

Alberto Ros, Manuel E. Acacio and José M. García
Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30100 Murcia (Spain)
email: {a.ros,meacacio,jmgarcia}@ditec.um.es

Abstract—Future many-core CMP designs that will integrate tens of processor cores on-chip will be constrained by area and power. Area constraints make impractical the use of a bus or a crossbar as the on-chip interconnection network, and tiled CMPs organized around a direct interconnection network will probably be the architecture of choice. Power constraints make impractical to rely on broadcasts (as, for example, Token-CMP does) or any other brute-force method for keeping cache coherence, and directory-based cache coherence protocols are currently being employed. Unfortunately, directory protocols introduce indirection to access directory information, which negatively impacts performance. In this work, we present DiCo-CMP, a novel cache coherence protocol especially suited to future many-core tiled CMP architectures. In DiCo-CMP the task of storing up-to-date sharing information and ensuring ordered accesses for every memory block is assigned to the cache that must provide the block on a miss. Therefore, DiCo-CMP reduces the miss latency compared to a directory protocol by sending requests directly to the cache that provides the block in a cache miss. These latency reductions result in improvements in execution time of up to 6% on average over a directory protocol. In comparison with Token-CMP, our protocol only sends one request message for each cache miss, as such is able to reduce network traffic by 43%.

Index Terms—Many-core CMP, cache coherence protocol, direct coherence, indirection problem, on-chip network traffic.

1 Introduction

The huge number of transistors that are currently offered in a single die has made major microprocessor vendors to shift towards multi-core architectures in which several processor cores are integrated on a single chip, leading to Chip-multiprocessors or CMPs [33].

Most current CMPs (for example, the dual-core IBM Power6 [23] and the eight-core Sun UltraSPARC T2 [38]) have a relatively small number of cores, every one with at least one level of private cache. These cores are typically connected through an on-chip shared bus or crossbar. However, the interesting new opportunity is now that the number of cores is expected to be doubled every 18 months [9], making undesirable elements that could compromise the scalability of these designs. One of such elements is the interconnection network. As shown in [22], the area required by a shared bus or a crossbar as the number of cores grows has to be increased to the point of becoming impractical. Tiled CMP architectures [43], which are designed as arrays of identical or close-to-identical building blocks (tiles) connected over a point-to-point unordered network, are a scalable alternative to these small-scale CMP designs and they help in keeping complexity manageable. In this work, we focus on tiled CMPs with private L1 caches and shared L2 caches. Therefore, some accesses to the shared cache will be sent to the local slice while the rest will be serviced by remote slices (L2 NUCA architecture [21]). Figure 1 shows the organization of a tile (left) and a 16-tile CMP (right).

On the other hand, most CMP systems provide programmers with the intuitive shared-memory model,

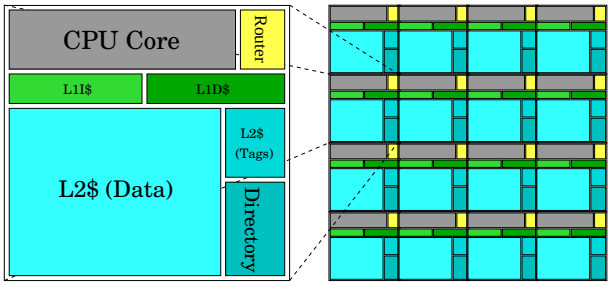


Fig. 1. Organization of a tile and a 4×4 tiled CMP.

which requires efficient support for cache coherence. Although a great deal of attention was devoted to cache coherence protocols in the last decades in the context of shared-memory multiprocessors, the technological parameters and constraints entailed by CMPs demand new solutions to the cache coherence problem [9].

Directory-based cache coherence protocols have been typically employed in systems with point-to-point unordered networks (as tiled CMPs are). Unfortunately, these protocols introduce indirection to obtain coherence information from the directory (commonly on chip as a directory cache), thus increasing cache miss latencies. Moreover, the number of cache misses suffering from indirection increases with tiled CMPs. This is because the directory information is commonly distributed among the tiles of the CMP through a physical address mapping [19], [38], [43], i.e., the tile wherein the directory information of a block resides (the *home* tile) is calculated by taking $log_2 n$ bits from the block address, where n is the number of tiles. Since this mapping distributes directory information among tiles in a round-robin fashion

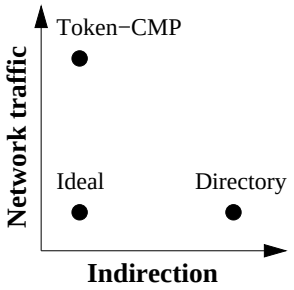


Fig. 2. Trade-off between indirection and network traffic.

without considering the cores requesting each block, the probability of accessing a remote tile increases.

An alternative approach that avoids indirection is Token-CMP [32]. Token-CMP is based on broadcasting requests to all last-level private caches. In this way, caches can directly provide data when they receive a request (no indirection occurs). Unfortunately, the use of broadcasting increases network traffic and, therefore, power consumption in the interconnection network, which has been previously reported to constitute a significant fraction (approaching 50% in some cases) of the overall chip power [25], [40]. Figure 2 shows the trade-off between Token-CMP and directory protocols [28]. An ideal protocol for tiled CMPs would avoid the indirection of the directory protocols without relying on broadcasting requests.

In this work, we present direct coherence, a cache coherence protocol that meets the advantages of directory and token protocols and avoids their problems. In direct coherence, the task of storing up-to-date sharing information and ensuring ordered accesses for every memory block is assigned to the cache that provides the block on a cache miss (the *owner* cache in a MOESI protocol). In this way, indirection is avoided by directly sending the requests to the owner cache instead of to the home tile, where coherence information resides in a directory protocol. Particularly, we describe an implementation of direct coherence for tiled CMPs, named as *DiCo-CMP*. In DiCo-CMP, the identity of the owner caches is speculatively recorded in a small structure called *L1 coherence cache* associated to each core. To achieve accurate owner predictions, this structure can be updated whenever the owner tile changes through control messages called *hints*. Additionally, since the owner cache can change on write misses, another structure called *L2 coherence cache* keeps up-to-date information about the identity of the owner cache and it is accessed each time a request fails to locate that cache.

In this way, DiCo-CMP reduces the latency of cache misses compared to a directory protocol by sending coherence messages directly from the requesting caches to those that must observe them, as it would be done in Token-CMP, and reduces network traffic compared to Token-CMP by sending just one request message on every cache miss, which also translates into improvements in execution time. Detailed simulations show that DiCo-CMP achieves improvements in total execution time of 6% on average over a directory protocol and of 3% on average over Token-CMP. Moreover, our proposal reduces network traffic up to 43% on average compared to Token-CMP, and consequently, the total power consumed in the interconnection network.

A first implementation of direct coherence was presented for distributed shared-memory multiprocessors in [35]. Later on, a preliminary version of direct coherence optimized for tiled CMPs (DiCo-CMP) was presented in [36]. Here, we extend the later work with the following contributions:

- A new proposal for updating the L1 coherence cache that employs address signatures to filter some useless hint messages. Address signatures allow us to significantly reduce the storage required by the original hints mechanism (from $4KB$ and scalability of $O(n)$ to $0.25KB$ for any number of cores) with a slight increase in network traffic. Additionally, this scheme constitutes our best alternative in terms of execution time.
- A more extensive evaluation that includes multimedia applications from the ALPBench suite [24].

The rest of the work is organized as follows. In Section 2 we present a review of the related work and the base protocols used for the evaluation. Section 3 describes DiCo-CMP. The different ways of updating the L1 coherence cache including the use of address signatures are described in Section 4. Section 5 studies the area and power requirements of DiCo-CMP. In Section 6, we introduce the methodology employed in the evaluation. Section 7 shows the performance results obtained by our proposal and, finally, Section 8 concludes the paper.

2 Related work

In this paper, we compare DiCo-CMP against two cache coherence protocols aimed to be used in CMPs: an implementation of a directory protocol for CMPs and Token-CMP. The next two subsections give some details regarding these two cache coherence protocols. First of all we comment on some of the related works.

In the shared-memory multiprocessors domain, Acacio *et al.* propose to avoid the indirection for cache-to-cache transfer misses [1] and upgrade misses [2] separately by predicting the current holders of every cache block. Predictions must be verified by the corresponding directory controller, thus increasing the complexity of the protocol on mis-predictions. Hossain *et al.* propose different optimizations for each sharing pattern considering a chip multiprocessor architecture [17]. Particularly, they accelerate the producer-consumer pattern by converting 3-hop read misses into 2-hop read misses. Again, communication between the cache providing the data block and the directory is necessary, thus introducing more complexity in the protocol. In contrast, our proposal is applicable to all types of misses (reads, writes and upgrades) and just the identity of the owner tile is predicted. Moreover, the fact that the directory information

is stored along with the owner of the block simplifies the protocol. Finally, differently from the techniques proposed by Acacio *et al.*, we avoid predicting the current holders of a block by storing the up-to-date directory information in the owner tile.

Also in the context of shared-memory multiprocessors, Cheng *et al.* [13] have proposed converting 3-hop read misses into 2-hop read misses for memory blocks that exhibit the producer-consumer sharing pattern by using extra hardware to detect when a block is being accessed according to this pattern. In contrast, our proposal obtains 2-hop misses for read, write and upgrade misses without the need of detecting sharing patterns.

Enright *et al.* propose Virtual Tree Coherence (VTC) [16]. In this mechanism, that uses coarse-grain coherence tracking [10], the sharers of a memory region are connected by means of a virtual tree. Since the root of the virtual tree serves as the ordering point in place of the home tile, and the root tile is one of the sharers of the region, the indirection can be avoided for some misses. In contrast, direct coherence protocols keep the coherence information at block granularity and the ordering point always has the valid copy of the block, which leads to less network traffic and lower levels of indirection.

Huh *et al.* [19] propose to allow replication in a NUCA cache to reduce the access time to a shared multibanked cache. More recently, Beckmann *et al.* [6] present ASR that replicates cache blocks only when it is estimated that the benefits of replication (lower L2 hit latency) exceeds its costs (more L2 misses). In contrast, our protocol reduces miss latencies by avoiding the access to the L2 cache when it is not necessary, and no replication is performed. DiCo-CMP could be also used in conjunction with techniques that try to make the best use of the limited on-chip cache storage.

Martin *et al.* present a technique that allows snooping-based protocols to utilize unordered networks by adding logical timing to coherence requests and reordering them on destiny to establish a total order [30]. Likewise, Agarwal *et al.* propose In-Network Snoop Ordering (INSO) [3] to allow snooping over unordered networks. The Intel QPI (Quick Path Interface) [20] also achieves 2-hop misses by broadcasting requests, but removes most responses by introducing a new cache state (F). Since direct coherence protocols do not rely on broadcasting requests, they generate less traffic and, therefore, less power consumption when compared to snooping-based protocols.

Martin *et al.* propose to use destination-set prediction to reduce the bandwidth required by a snoopy protocol [28]. Differently from DiCo-CMP, this proposal is based on a totally-ordered interconnect (a crossbar switch), which does not scale with the number of nodes. Destination-set prediction is also used by Token-M in shared-memory multiprocessors with unordered networks [27]. However, on mis-predictions, requests are solved by resorting on broadcasting after a time-out period. Differently, in DiCo-CMP mis-predictions are re-sent immediately to the owner cache, thus reducing latency and network traffic.

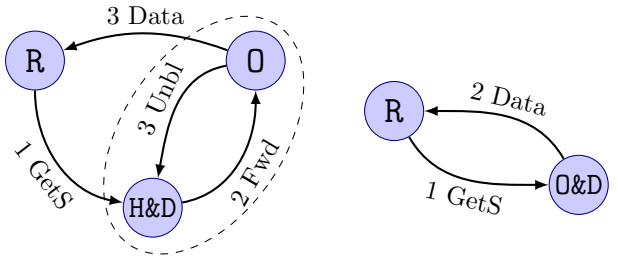
Finally, some authors evaluated the use of hints with different objectives [7], [18]. In these works the authors try to keep updated directory information to find out where a valid copy of the block can be obtained in case of a read miss. In contrast, we use the hints as a policy to update the location of the owner cache which servers as ordering point and stores up-to-date directory information. On the other hand, the use of signatures has been recently proposed for disambiguating addresses across threads in transactional memory [12], [42]. In contrast, we use signatures to keep information that improves the efficiency of the hints mechanism.

2.1 Directory-CMP

Directory-based coherence protocols [11] have been widely used in shared-memory multiprocessors. Now, several chip multiprocessors, like Piranha [5], also use directory protocols to keep cache coherence. In this paper, we compare our proposal against a directory protocol similar to the intra-chip coherence protocol used in Piranha, which is based on MOESI states. In this implementation, on-chip directory caches are used for accelerating the accesses to directory information for blocks stored in the L1 caches. Moreover, the protocol implements a migratory-sharing optimization [39], in which a cache holding a modified cache block invalidates its copy when responding with the block, thus granting the requesting processor read/write access to the block (even when only read permission was requested). This optimization has been shown to improve substantially the performance of many applications.

2.2 Token-CMP

Token coherence [29] is a framework for designing coherence protocols whose main asset is that it decouples the correctness substrate from several different performance policies. Token coherence protocols can avoid both the need of a totally ordered network and the introduction of additional indirection caused by the directory in the common case of cache-to-cache transfers. Token coherence protocols keep cache coherence by assigning T tokens to every memory block, where one of the T is the owner token. Then, a processor can read a block only if it holds at least one token for that block and has valid data. On the other hand, a processor can write a block only if it holds all T tokens for that block and has valid data. Token coherence avoids starvation by issuing a persistent request when a processor detects potential starvation. In this paper, we compare our coherence protocol against Token-CMP [32], which is a performance policy aimed to achieve low-latency cache-to-cache transfer misses. Token-CMP targets CMP systems, and uses a distributed arbitration scheme for persistent requests, which are issued after a single retry to optimize the access to contended blocks. Again, the migratory-sharing optimization is implemented.



(a) Directory protocols. (b) Direct coherence protocols.

Fig. 3. How cache-to-cache transfer misses are performed in directory and direct coherence protocols. R=Requester; H=Home; D=Directory; O=Owner.

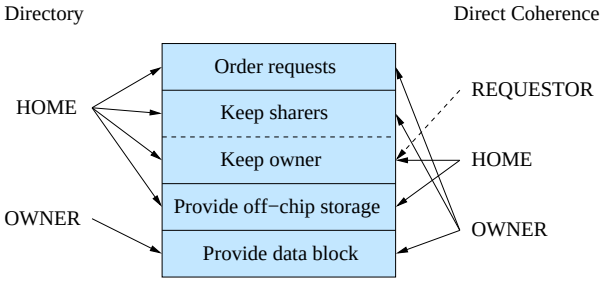


Fig. 4. Tasks performed in cache coherence protocols.

3 DiCo-CMP

In this section, we describe DiCo-CMP in detail. First, we explain how direct coherence avoids indirection for most cache misses by changing the distribution of the roles involved in cache coherence maintenance. We also study the changes in the structure of the tiles necessary to implement DiCo-CMP. Then, we describe the cache coherence protocol for tiled CMPs and, finally, we study how to avoid the starvation issues that could arise.

3.1 Direct coherence basis

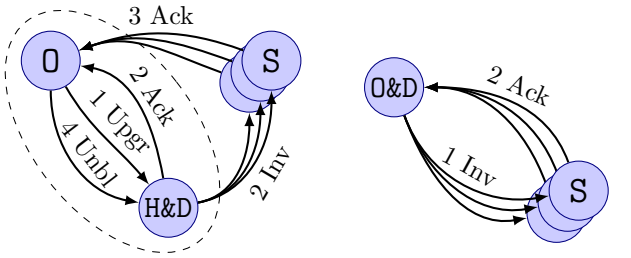
Directory protocols introduce indirection in the critical path of cache misses. Figure 3(a) shows a cache miss suffering from indirection in a directory protocol, a cache-to-cache transfer for a read miss. When a cache miss takes place it is necessary to access the home tile to obtain the directory information and serialize the requests before performing any coherence action (*1 GetS*). In case of a cache-to-cache transfer miss, the request is subsequently forwarded to the owner cache (*2 Fwd*), where the block is provided (*3 Data*). As it can be observed, the miss is performed in three hops. Moreover, requests for the same block cannot be processed by the directory until it receives the unblock message (*3 Unbl*).

To avoid this indirection problem, we propose to directly send the request to the provider of the block, i.e., the owner cache. This is the main motivation behind direct coherence. To allow that, direct coherence stores the sharing information along with the owner block, and it also assigns the task of keeping cache coherence and ensuring ordered accesses for every memory block to the tile that stores that block. As shown in Figure 3(b), DiCo-CMP sends the request to the owner cache (*1 GetS*) instead of to the home tile. In this way, data is provided by the owner cache in just two hops (*2 Data*).

Therefore, direct coherence requires a re-distribution of the roles involved in solving a cache miss. Next, we describe the tasks performed in cache coherence protocols and the component responsible for each task in both directory and direct coherence protocols, which are illustrated in Figure 4:

- *Order requests*: Cache coherence maintenance requires to serialize the requests issued by different cores for the same block. In snooping-based cache coherence protocols, the requests are ordered by the shared interconnection network (usually, a bus). However, since tiled CMP architectures implement an unordered network, this serialization of the requests must be carried out by another component. Directory protocols assign this task to the home tile of each memory block. In direct coherence protocols, this task is performed by the owner cache.
- *Keep coherence information*: Coherence information is used to track blocks stored in private caches. In protocols that include the *O* state, like MOESI protocols, coherence information also identifies the owner cache. In particular, *sharing information* is used to invalidate all cached blocks on write misses, while *owner information* is used to know the identity of the provider of the block on every miss. Directory protocols store coherence information at the home tile, where cache coherence is maintained. Instead, direct coherence requires that sharing information be stored in the owner cache for keeping coherence there, while owner information is stored in two different components. First, the requesting cores need to know the owner cache to send the requests to it. Processors can easily keep the identity of the owner cache, e.g., by recording the last core that invalidated their copy. However, this information can become stale and, therefore, it is only used for avoiding indirection (dashed arrow in Figure 4). Then, the responsible for tracking the up-to-date identity of the owner cache is the home tile which must be notified on every ownership change.
- *Provide the data block*: If the valid copy of the block resides on chip, data is always provided by the owner cache, since it always holds a valid copy.
- *Provide off-chip storage*: When the valid copy of a requested block is not stored on chip, an off-chip access is required to obtain the block. In both protocols, the home tile is responsible for detecting that the owner copy of the block is not stored on chip, sending the off-chip request and receiving the data block.

Another example of the advantages of DiCo-CMP is shown in Figure 5. This diagram represents an upgrade that takes place in a tile whose L1 cache is the owner one, which happens frequently in common applications



(a) Directory protocols. (b) Direct coherence protocols.

Fig. 5. How upgrades are performed in directory and direct coherence protocols. O=Owner; H=Home; D=Directory; S=Sharers.

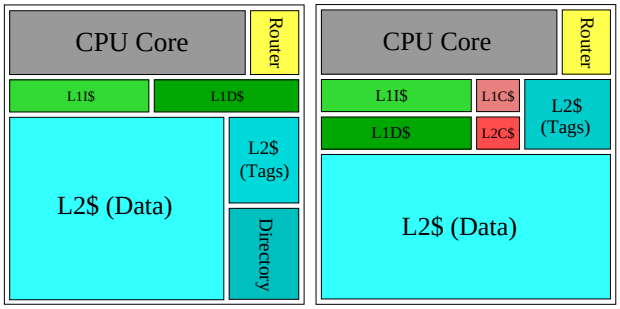


Fig. 6. Organization of a tile for a directory protocol (left) and for direct coherence (right).

(e.g., in the producer-consumer pattern). In a directory protocol, upgrades are resolved by sending the request to the directory (*1 Upgr*), which replies with the number of acknowledgements that must be received before the block can be accessed (*2 Ack*), and sends invalidation messages to all sharers (*2 Inv*). Sharers confirm their invalidation to the requester (*3 Ack*). Once all the acknowledgements have been received by the requester, the block can be modified and the directory is unblocked (*4 Unbl*). In contrast, in DiCo-CMP only invalidation messages (*1 Inv*) and acknowledgements (*2 Ack*) are required, thus solving the miss with just two hops.

Additionally, by keeping together the owner block and the directory information, the control messages between them are not necessary, thus saving some network traffic (two messages in Figure 3 and three in Figure 5). Moreover, this allows the O&D node to solve misses without using transient states, thus reducing the number of states of the cache controller and making the implementation simpler. Finally, the elimination of transient states at the directory reduces the waiting time for the subsequent requests and, therefore, average miss latency.

3.2 Changes to the structure of the tiles of a CMP

The new distribution of roles that characterizes direct coherence protocols requires some modifications in the structure of the tiles that build the CMP. Firstly, the identity of the sharers for every block is stored in the corresponding owner cache, instead of in the home tile, to allow caches to keep coherence for the memory blocks that they hold in owner state. Therefore, DiCo-CMP extends the tags' part of the L1 caches with a sharing code field, e.g., a bit-vector (L2 caches already include this field in directory protocols). In contrast, DiCo-CMP does not need the directory structure in the home tile that traditional directory protocols require.

Additionally, DiCo-CMP needs two extra hardware structures that are used to record the identity of the owner cache for a certain set of blocks:

- *L1 coherence cache* (L1C$): The pointers stored in this structure are used by the requesting core to avoid indirection by directly sending local requests to the corresponding owner cache. Therefore, this structure is located close to each processor's core. DiCo-CMP can update this information in several ways based on network usage (see Section 4).
- *L2 coherence cache* (L2C$): Since the owner cache can change on write misses, this structure must track the owner cache for each block allocated in any L1 cache. This structure is accessed each time a request fails to locate the owner cache. Therefore, its information must be updated whenever the owner cache changes through control messages, which must be processed by the L2C$ in the very same order in which they were generated (see Section 3.3.3).

Figure 6 shows a tile design for directory protocols (left) and for direct coherence protocols (right). A comparison among the extra storage required by the protocols considered in this work can be found in Section 5.

3.3 Description of the cache coherence protocol

3.3.1 Requesting processor

When a processor issues a request that misses in its private L1 cache, it directly sends the request to the owner cache in order to avoid indirection. The identity of the potential owner cache is obtained from the L1C$, which is accessed at the time that the cache miss in detected. If there is a hit in the L1C$, the request is sent to the owner cache. Otherwise, the request is sent to the home tile, where the L2C$ will be accessed to get the identity of the current owner cache.

3.3.2 Request received by a cache that is not the owner

When a request is received by a cache that is not the current owner of the block, it simply re-sends the request. L1 caches re-send requests to the home tile. On the other hand, if the request is received by the home tile and there is a hit in the L2C$, it is sent to the current owner cache. In absence of race conditions the request will reach the owner cache. Finally, if there is a miss in the L2C$ and the L2 cache is not the owner of the block, main memory is accessed to get the block. In this case, the block is allocated in the requesting L1 cache, which gets the ownership of the block, but not in the L2 cache (as occurs in the directory protocol since we assume non-inclusive caches). In addition, it is necessary to allocate a new entry in the L2C$ pointing to the current L1 owner cache.

The appearance of owner mis-predictions could impact on the minimum number of virtual networks that ensure the absence of deadlock at the interconnect. However, the longest message dependency chain (without cycles) in direct coherence is the same as in directory protocols (five messages) and, therefore, the minimum number of virtual networks required is the same (five in both protocols) according to [15].

3.3.3 Request received by the owner cache

Every time a request reaches the owner cache, it is necessary to check whether this cache is currently processing a request from a different processor for the same block (a previous write waiting for acknowledgements). In this case, the block is in a busy or transient state, and the request must wait until all the acknowledgements are received.

If the block is not in a transient state, the miss can be immediately resolved. If the owner is the L2 cache all requests (reads and writes) are resolved by deallocating the block from the L2 cache and allocating it in the private L1 cache of the requester. Again, the identity of the new owner cache must be stored in the L2C$.

When the owner is an L1 cache, read misses are completed by sending a copy of the block to the requester and adding it to the sharing code field. Since our protocol is also optimized for the migratory-sharing pattern, read misses for migratory blocks invalidate the copy in the owner cache and send the exclusive data to the L1 cache of the requesting processor.

For write misses, the owner cache sends invalidation messages to all the caches that hold a copy of the block and, then, it sends the data block to the requester. Acknowledgement messages are collected at the requesting cache. If the miss is an upgrade the owner cache checks the sharing code field to know whether the requester still holds a copy of the block (note that a previous write miss from a different processor could have invalidated its copy and in this case the owner cache should also provide a valid copy of the block). As shown in Figure 5, upgrade misses that take place in the owner cache just need to send invalidations and receive acknowledgements (two hops in the critical path).

Finally, since the L2C$ must store up-to-date information regarding the owner cache, every time that the owner cache changes, the old owner cache also sends a control message to the L2C$ indicating the identity of the new owner. These messages must be processed by the L2C$ in the very same order in which they were generated. Otherwise, the L2C$ could fail to store the identity of the current owner. To enforce this constraint, once the L2C$ processes the message reporting an ownership change from the old owner, it sends a confirmation response to the new owner. Until this confirmation message is received by the new owner, it could access the data block (if already received), but cannot give the ownership to another cache. Since these two control messages are not in the critical path the cache miss, they do not introduce extra latency.

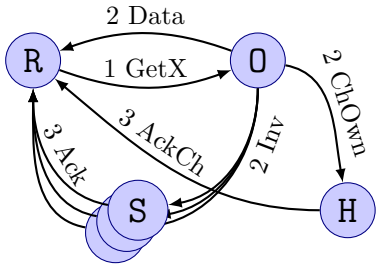


Fig. 7. Example of ownership change upon write misses. R=Requester; O=Owner; S=Sharers; H=Home.

As an example, Figure 7 illustrates a write miss for a shared block. It assumes that the requester has valid and correct information about the identity of the current owner tile in the L1C$ and, therefore, it is able to send directly the request to the owner tile (*1 GetX*). Then, the owner tile must perform the following tasks. First, it sends the data block to the requester (*2 Data*). Second, it sends invalidation messages to all the sharers (*2 Inv*), and it also invalidates its own copy. The information about the sharers is obtained from the sharing code stored along with every owner block. Third, it sends the message informing about the ownership change to the home tile (*2 ChOwn*). All tiles that receive an invalidation message respond with an acknowledgement message to the requester once they have invalidated their local copies (*3 Ack*). When the data and all the acknowledgements arrive to the requesting processor the write operation can be performed. However, if another write request arrives to the tile that previously suffered the miss (*R*), it cannot be handled until the acknowledgement to the ownership change issued by the home tile (*3 AckCh*) is received.

3.3.4 Replacements

In our particular implementation, when an owner block is evicted from an L1 cache, it must be allocated at the L2 cache along with its sharing code. The replacement is performed just by sending a writeback message to the home tile, as happens in Token-CMP. Then, the L2C$ deallocates its entry for this block because the owner cache is now the L2 cache. Replacements for blocks in shared state are performed transparently, i.e., no coherence messages are needed.

Finally, no coherence actions must be performed in case of an L1C$ replacement. However, when an L2C$ entry is evicted, the protocol should ask the owner cache to invalidate all the copies from the L1 caches. Luckily, as happens to the directory cache in directory protocols, an L2C$ with the same number of entries and associativity than the L1 cache could be enough to completely remove this kind of replacements [37].

3.4 Preventing starvation

Directory protocols avoid starvation by queuing requests in FIFO order at the directory buffers. Differently in

DiCo-CMP, write misses can change the cache that keeps coherence for a particular block and, therefore, some requests can take some extra time until this cache is finally found. If a memory block is repeatedly written by different cores, a request could take some time to find the owner cache ready to process it, even when it is sent by the L2C$. Hence, some cores could be completing their requests while other requests remain starved.

DiCo-CMP detects and avoids starvation by using a simple mechanism. In particular, each time a request accesses the L2C$ a counter is increased. The request is considered starved when this counter reaches a certain value (e.g, three accesses to the L2C$, in this work). When the L2C$ detects a starved request, it re-sends the request to the owner cache, but it records the address of the requested block. If the starved request reaches the owner cache, the miss is resolved, and the L2C$ is notified, ending the starvation situation. If the starved request does not reach the owner tile, ownership is moving from a cache to another one, and a message notifying the change has been issued. When the L2C$ receives this message, it detects the block as suffering from starvation, and the acknowledgement message is not sent. This ensures that the identity of the owner cache does not change until the starved request completes.

4 Updating the L1 coherence cache

DiCo-CMP uses the L1C$ to avoid indirection by keeping pointers that identify the owner cache of certain blocks. Several policies can be used to update the value of these pointers. A first option is to record the information about the last core that invalidated or provided each block, i.e., the last processor that wrote the block. When a block is invalidated from an L1 cache, the L1C$ records the identity of the processor causing the invalidation. In case of a read miss, the identity of the provider of the block is also stored. Additionally, when an owner block is evicted from an L1 cache, some control messages are sent to the sharers to inform about the new location of the owner cache, the home tile. We call this policy the *Base* policy.

Unfortunately, in most cases this information is not enough to obtain accurate owner predictions and it must be enhanced by sending some *hints*. Hints are control messages that inform the L1C$ about owner changes. Since sending hints to all cores on each change is not efficient in terms of network traffic, it is necessary to keep track of those cores that need to receive hints for each memory block.

In our previous work of DiCo-CMP [36], we proposed a *frequent sharers* mechanism to send hints. This mechanism requires the addition of a new field to each cache entry. This field keeps a bit-vector that identifies the requesting cores (or frequent sharers) for each owner block. When the owner changes, hints are sent to these cores to update their L1C$s. Moreover, the frequent sharers vector is also sent along with the data message. Since

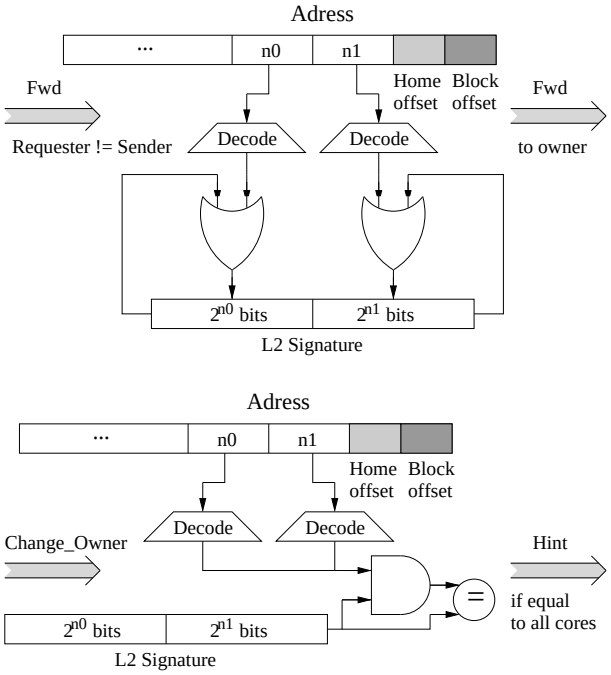


Fig. 8. Organization of the address signature mechanism proposed to send hints.

we choose not to store the frequent sharer information at the L2 cache level in order to keep storage requirements low, this field is reset whenever there is an L1 cache eviction of an owner block. We call this policy *Hints FS*. This mechanism is not very suitable for large-scale CMPs since the area required by the bit-vector could become prohibitive. In addition, it does not filter hint messages for those blocks in which the *Base* mechanism works well, thus consuming precious network bandwidth.

Therefore, in this work, we propose to use *address signatures* to design a scalable hints mechanism in terms of area requirements. We call this policy *Hints AS*. Address signatures encode a set of addresses into a register of fixed size, following the principles of hash-encoding with allowable errors as described in [8]. An address signature stores a superset of the addresses that have been encoded in it, so it only can claim that a particular address has not been included in it. Therefore, the disadvantage of address signatures is that false positives can happen. However, this is not a correctness issue for the hints mechanism but maybe a performance issue, due to a potential increase in network traffic.

Essentially, each home tile includes an address signature (*L2 Signature*) that encodes a certain set of addresses. In order to filter some useless hints we only store the addresses for those cache misses mis-predicting the owner tile, i.e., the home tile receives a request from a core that is not the requester one (Figure 8, top). In this way, when the home tile is informed about the ownership change for a particular block, it checks the signature and broadcast hints to all cores if the address is present (Figure 8, bottom). Note that when invalidation messages are required it is not necessary to send hints to the cores that receive them.

Since this scheme only uses one signature for all cores, and hints are broadcast to them in case the address is found, some cores will receive hints for blocks that they are not actually requesting, thus overloading the L1C$. To avoid this effect, we add another address signature (L1 Signature) to each core. On each cache miss, the address of the block is encoded in the signature. Then, when a hint is received, it is only stored in the L1C$ if the address is found in the signature.

Particularly, addresses are encoded using a double-bit-select signature implementation [42], as Figure 8 shows. The signature is divided into two sets. The $log_2(b) - 1$ less-significant bits ($n1$) are decoded and ORed with the first set, being b the size in bits of the signature. The $log_2(b) - 1$ subsequent less-significant bits ($n0$) are decoded and ORed with the second set. An address belongs to the signature if the corresponding bit is present in both sets.

When we refer to the less-significant bits we do not take into consideration the block offset. For the L2 signature, we neither take the home offset, as illustrated in Figure 8. This offset comes from assigning an address to a home tile according to the less-significant bits ($log_2 n$).

5 Area and Power Considerations

In this section, we compare the memory overhead and the extra structures needed by the three protocols considered in this work: Token-CMP, Directory, and DiCo-CMP. Moreover, we discuss how frequently these structures are accessed to demonstrate that our proposal will not have significant impact on the power consumed by these structures and, therefore, significant reductions in total power consumption can be expected as a result of the savings in terms of network traffic that DiCo-CMP entails (see Section 7.3).

Token-CMP needs to keep the token count for any block stored both in the L1 and L2 caches. This information only requires $\lceil log_2(n+1) \rceil$ bits (the owner-token bit and the non-owner token count), where n is the number of processing cores. These additional bits are stored in the tags' part of both cache levels.

Directory protocols store the on-chip directory information either in the L2 tags when the L2 cache holds a copy of the block or in a distributed directory cache when the block is stored in any of the L1 caches but not in the L2 cache. In our implementation, the number of entries of a directory bank is the same as the number of entries of an L1 cache, since this size is enough to always find the directory information for on-chip misses, i.e., without incurring in directory misses [37]. The directory must be accessed on each cache miss.

DiCo-CMP stores the directory information for blocks held in any L1 or L2 cache in the owner cache (L1 or L2). Moreover, it uses two structures that store a pointer to the owner cache, the L1 and L2 coherence caches. The L1C$ is accessed only when it is known that there is a cache miss in order to keep power consumption low. The L2C$ is necessary for locating the owner cache whenever the information in the L1C$ is not correct. This structure is only accessed for misses affected by indirection (about 22% of the cache misses as shown in Section 7.1). As happens with the on-chip directory cache in the directory protocol, the L2C$ does not require more entries than the number of entries of the L1 caches. Differently from a directory cache, just one pointer is stored in each entry. In this way, the L2C$ required by DiCo-CMP has smaller size than the directory cache employed in the directory-CMP protocol. The use of hints improves performance at the cost of increasing both storage requirements and network traffic. In particular, the *frequent sharers* mechanism requires to store the frequent sharers in the tags of the L1 caches –$O(n)$–. On the other hand, the *address signature* mechanism only uses two signatures per tile (1024 bits, each one) and, therefore, the storage requirements are reduced and scales with the number of cores.

TABLE 1
Memory overhead introduced by coherence information (per tile) in a 4x4 tiled CMP.

	Structure	Entry size	Entries	Total size	Overhead
Data	L1 cache	tag + 64 bytes	2K	134.25KB	
	L2 cache	tag + 64 bytes	16K	1070KB	
Token-CMP	L1$ tokens	5 bits	2K	1.25KB	**0.93%**
	L2$ tokens	5 bits	16K	10KB	
Directory	L2$ dir. inf.	2 bytes	16K	32KB	**3.59%**
	Dir. cache	tag + 2 bytes	2K	11.25KB	
DiCo-Base	L1$ dir. inf.	2 bytes	2K	4KB	**4.19%**
	L2$ dir. inf.	2 bytes	16K	32KB	
	L1C$	tag + 4 bits	2K	7.25KB	
	L2C$	tag + 4 bits	2K	7.25KB	
DiCo-Hints FS	L1$ freq. sh.	2 bytes	2K	4KB	**+0.34%**
DiCo-Hints AS	L1 signature	128 bytes	1	0.125KB	**+0.02%**
	L2 signature	128 bytes	1	0.125KB	

For the particular configuration of this work (a 4×4 tiled CMP with 128KB L1 private caches), the number of bits required for storing the sharing code is 16 (2 bytes), whereas just $log_2 16 = 4$ bits are needed for storing a single pointer. Table 1 summarizes the structures, the size, and the memory overhead respect to the size of the data caches required by Token-CMP, directory-CMP and the different implementations of DiCo-CMP evaluated in this work. Note that the table concentrates on the structures used for keeping coherence information and, therefore, does not account for the extra structures required by Token-CMP and DiCo-CMP to avoid starvation. In general, we can see that direct coherence has an overhead close to a directory protocol.

6 Simulation Environment

We evaluate our proposal with full-system simulation using Virtutech Simics [26] extended with Multifacet GEMS 1.3 [31]. GEMS provides a detailed memory system timing model which accounts for all protocol messages and state transitions. In order to model precisely the interconnection network, and thus, obtain more accurate results, we have replaced the original (not very detailed) network simulator offered by GEMS with the SiCoSys detailed interconnection network simulator [34].

TABLE 2
System parameters.

Memory Parameters (GEMS)	
Processor frequency	3 GHz
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I & D caches	128KB, 4 ways
L1 cache hit time	4 cycles
Shared unified L2 cache	1MB/tile, 4 ways
L2 cache hit time	6 + 9 cycles (tag + data)
L1 & L2 Coherence cache	512 sets, 4 ways, 1 cycle
L1 & L2 Signatures	1024 bits, double-bit-select
Memory access time	160 cycles
Network Parameters (SiCoSys)	
Network frequency	1.5 GHz
Topology	4x4 Mesh
Switching technique	Wormhole
Routing technique	Deterministic X-Y
Data and control message size	4 flits and 1 flit
Flit size	144 bits (18 bytes)
Routing time	1 cycle
Switch time	1 cycle
Link latency (one hop)	2 cycles
Link bandwidth	1 flit/cycle

SiCoSys allows to take into account most of the VLSI implementation details with high precision but with much lower computational effort than hardware-level simulators. In addition, we have extended SiCoSys to allow us to simulate multicast networks.

The simulated system is a tiled CMP organized as a 4×4 array of replicated tiles. Since we consider CMPs with a relatively large number of cores, each tile contains an in-order processor core, thus offering better performance/power ratio than a small number of complex cores would obtain. Table 2 shows the values of the main parameters of the system evaluated in this work.

Finally, we have used a varied selection of twelve scientific and multimedia applications for the evaluation. *Barnes* (16K bodies, 4 time steps), *Cholesky* (tk15.O), *FFT* (64K complex doubles), *Ocean* (258x258 ocean), *Radix* (1M keys, 1024 radix), *Raytrace* (teapot), *Volrend* (head) and *Water-NSQ* (512 molecules, 4 time steps) are from the SPLASH-2 benchmark suite [41]. *RaytraceOpt* improves the original *Raytrace* application by removing a lock acquisition for a ray ID which is not used for any actual computation. *Unstructured* (Mesh.2K, 5 time steps) is a computational fluid dynamics application. *MPGdec* (525_tens_040.m2v) and *MPGenc* (output of *MPGdec*), are multimedia applications from the APLBench suite [24]. We account for the variability in multithreaded workloads [4] by doing multiple simulation runs for each benchmark in each configuration and injecting random perturbations in memory systems timing for each run. The experimental results reported in this paper correspond to the parallel phase of each program.

7 Evaluation results and analysis

We compare the different implementations of DiCo-CMP proposed in Section 4 (*base*, *hints FS*, and *hints AS*) with both Token-CMP and the directory protocol described in Section 2. In addition, to find out the potential of DiCo-CMP, we have implemented an *oracle* policy in which the identity of the current owner cache is provided on every cache miss.

7.1 Impact on the number of hops needed to solve cache misses

In general, DiCo-CMP reduces the number of hops needed to solve a miss by avoiding the indirection introduced by the access to the home tile. However, some misses can increase the number of hops compared to a directory protocol due to owner mis-predictions. In order to study how DiCo-CMP impacts on the number of hops needed to solve cache misses, we classify each miss in one of the following categories:

- *2-hop misses*: Misses belonging to this category does not suffer from indirection since the number of hops in the critical path of the miss is two. In directory protocols, misses fall into this category either when the home tile of the requested block can provide the copy of the block or when the miss takes place in the home tile, and in both cases it is not necessary to invalidate blocks from other tiles. Token-CMP solves all misses that do not require persistent requests in two hops. Finally, DiCo-CMP solves cache misses using two hops either when the request is directly sent to the current owner cache and invalidations are not required, or when the miss takes place in the tile where the owner block resides (upgrades). In all protocols, when the miss takes place in the home tile and this tile holds the owner block in the L2 cache, the miss is solved without generating network traffic (0-hop miss). These misses are also included in this category because they do not introduce indirection.
- *3-hop misses*: A miss belongs to this category when three hops in the critical path are necessary to solve it. This never happens in Token-CMP.
- *>3-hop misses*: We include in this category misses that need more than three hops in the critical path to be solved. This only happens in DiCo-CMP when the identity of the owner cache is mis-predicted, or in Token-CMP when persistent requests are required to solve the miss.
- *Memory misses*: Misses that require off-chip access since the owner block is not stored on chip fall into this category.

Figure 9 shows the percentage of cache misses that fall into each category. As commented in the introduction, in tiled CMP architectures that implement a directory protocol it is not very frequent that the requester be at the home tile of the block because the distribution of blocks among tiles is performed in a round-robin fashion. However, the fact that sometimes the block is found in the L2 cache in owner state due to L1 cache evictions, decreases the number of misses with indirection. In this way, the first bar in Figure 9 shows that most applications have an important fraction of misses

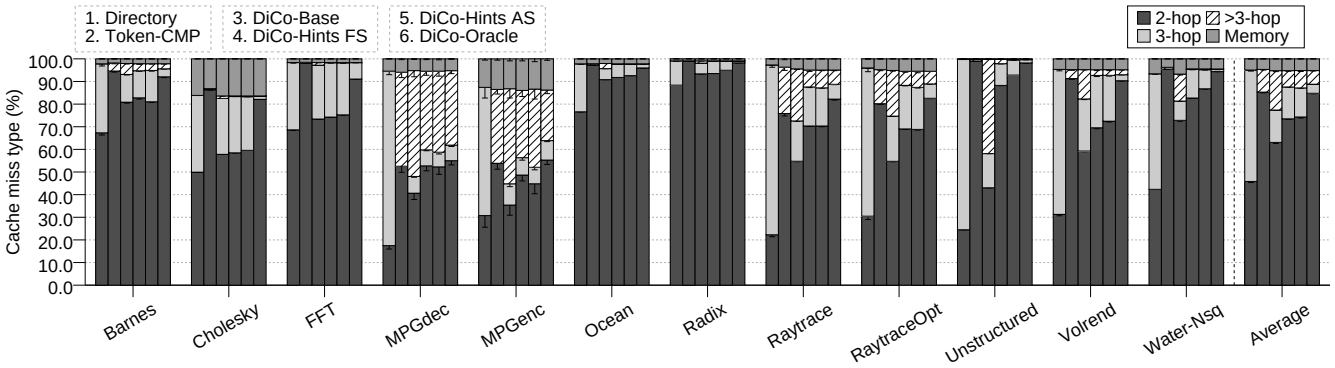


Fig. 9. How each miss type is solved for the applications evaluated in this work.

suffering from indirection, like *MPGdec*, *MPGenc*, *Raytrace*, *RaytraceOpt*, *Unstructured*, and *Volrend*, and other applications in which most of the misses are solved in two hops, like *Barnes*, *FFT*, *Ocean*, and *Radix*. Obviously, DiCo-CMP will have more impact for the applications that suffer more indirection, although this impact will also depend on the cache miss rate of each application. We also can observe that Token-CMP solves most of the misses (85% on average) using two hops (second bar).

As shown in the third bar of Figure 9, *DiCo-Base* increases the percentage of cache misses without indirection compared to a directory protocol (from 46% to 63% on average). On the other hand, 17% of cache misses are solved needing more than three hops. This fact is due to owner mis-predictions that arise for two reasons: (1) staled owner information was found in the L1C$ or (2) the owner cache is changing or busy due to race conditions and the request is sent back to the home tile. The first case can be removed with a precise hints mechanism, as clearly happens in *Unstructured*. In the second case, the extra number of hops entailed by DiCo-CMP is equivalent to the cycles that the requests wait at the home tile until they are processed in the base directory protocol and, consequently, it does not suppose extra miss latency. This kind of *>3-hop* misses mainly appears in applications with high levels of contention, like *MPGdec*, *MPGenc* and *Raytrace*, and they also occur in Token-CMP.

The two hints mechanisms implemented for DiCo-CMP, *DiCo-Hints FS* and *DiCo-Hints AS* (fourth and fifth bars, respectively), increase the percentage of misses solved in two hops with respect to *DiCo-Base* in 11% and 12% on average, respectively. The main advantage of *DiCo-Hints AS* is its low storage overhead. Although for some applications the use of hints slightly increases the percentage of two-hop misses, like in *Barnes*, *Cholesky*, and *FFT*, for others, especially *Unstructured*, hints significantly help to achieve accurate predictions. Hints are mainly useful for applications in which the migratory-sharing pattern is common since writes (or migratory reads) for blocks following this pattern do not send invalidations because the owner cache has the only valid copy of the block. Therefore, in *DiCo-Base*, the cores requesting migratory blocks do not update the pointer stored in their L1C$.

The *DiCo-Oracle* implementation (last bar) gives us the potential of DiCo-CMP. Therefore, the results are similar to the ones obtained by Token-CMP. In both cases, the misses falling into the *>3-hop* category are for contended blocks. Although *DiCo-Hints AS* does not obtain the same percentage of 2-hop misses than *DiCo-Oracle* (10% less on average), it has similar percentage of *>3-hop* misses which makes this solution perform close to the oracle case, as we will see in Section 7.4.

7.2 Impact on cache miss latencies

The avoidance of indirection shown by DiCo-CMP reduces the average cache miss latency. In addition, DiCo-CMP removes the transient states at the directory by putting together the provider of the block and the directory information. This fact reduces the time that the requests are waiting at the directory to be processed, which results in even more latency reductions. Figure 10 shows L1 cache miss latency for the applications evaluated in this work normalized with respect to Token-CMP, which is shown as a horizontal line. It does not consider the overlapping of the misses, and latency is calculated considering each miss individually. Latency is broken down into four segments to understand better in what way DiCo-CMP reduces the cache miss latency:

- *Finding*: It is the time elapsed between the issue of a request and the arrival of the request to the serialization point, i.e., the home tile for directory protocols and the owner cache for DiCo-CMP.
- *Waiting*: In directory protocols, it is the time spent waiting at the home tile because another request for the same block is being processed. In DiCo-CMP, this segment represents the period elapsed between the first time that the owner cache is found and the time when the owner cache processes the request.
- *Memory*: It is the time spent getting the data block from main memory when it is required.
- *Solving*: It is the time elapsed between the request leaves the serialization point and the block is accessed by the requesting processor. This period includes the need of forwarding the request in a directory protocol, and the issue of data, invalidation and acknowledgement messages in both protocols.

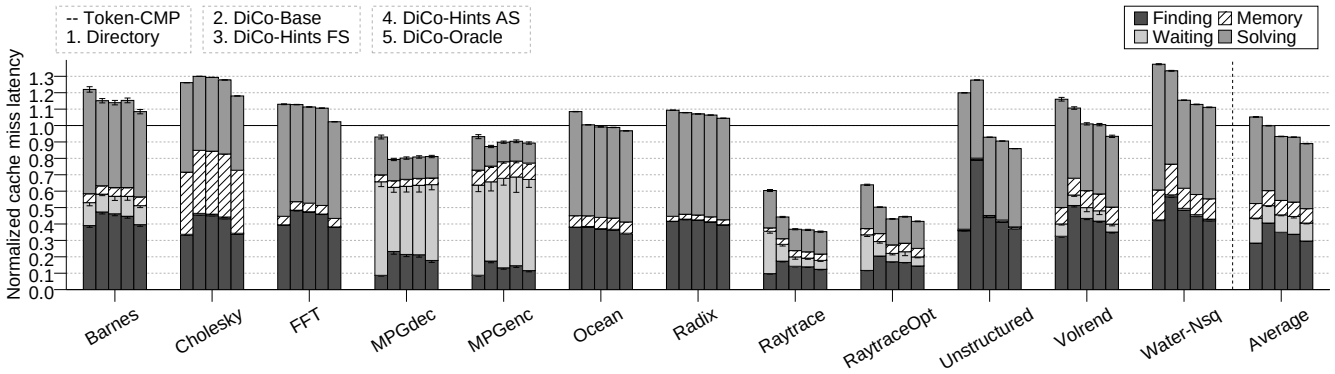


Fig. 10. Normalized L1 cache miss latency for the applications evaluated in this work.

We do not consider this classification for Token-CMP because this protocol does not follow a particular order to solve cache misses.

In general, we can see that all the polices implemented for DiCo-CMP reduce the average cache miss latency compared to a directory protocol (first bar). Both *DiCo-Hints FS* and *DiCo-Hints AS* (third and fourth bars, respectively) also reduce the latency compared to Token-CMP. In particular *DiCo-Hints AS* obtains reductions of 12% and 7% on average over a directory protocol and Token-CMP, respectively. Moreover, its average latency is close to the obtained for the oracle case (last bar).

Looking at the different segments into which cache miss latency is split, we can observe that, in general, the finding time is shorter for the directory protocol. This is because in a directory protocol this period always comprises a single hop. However, DiCo-CMP can take several hops until the owner cache is found. As we can see, the more accurate are the owner predictions, the shorter is this segment. For example, *Unstructured* has a lot of mis-predictions (*>3-hop* misses) when the base policy is considered, which doubles the finding time compared to a directory protocol. Nevertheless, hints significantly helps to reduce this extra latency. Note that for some other applications the increase in the number of *>3-hop* misses is due to race conditions, which do not increase the latency of the misses. Finally, for other applications like *FFT*, *Ocean*, *Radix*, and *Raytrace*, the finding time in the oracle case is a bit shorter than in a directory protocol. This is because sometimes the owner cache is closer to the requesting core than the home tile.

Regarding the waiting time, we can observe that only some applications (*Barnes*, *MPGdec*, *MPGenc*, *Raytrace*, *RaytraceOpt* and *Volrend*) have requests waiting at the home tile during a meaningful time. Since DiCo-CMP removes transient states at the directory, this waiting is shortened for some of these applications. This waiting time is usually caused by contended locks and, therefore, reductions in these requests result in a faster acquisition of locks and, finally, in reductions in the number of memory requests, as happens in *Raytrace* and *RaytraceOpt*.

The memory time does not vary significantly for the evaluated protocols. However, the solving time is always reduced when DiCo-CMP is implemented because for most misses it requires just one hop, in contrast to the two hops (forwarding and data) needed for a directory protocol. This time is not affected by the policy employed because the policy only tries to find the owner cache as soon as possible.

7.3 Impact on network traffic

Figure 11 compares the network traffic generated by the protocols considered in this work. In particular, each bar plots the number of bytes transmitted through the interconnection network (the total number of bytes transmitted by all the switches of the interconnect) normalized with respect to Token-CMP. As stated in Section 6, we assume an interconnection network with multicast support. We can see that Token-CMP obtains the highest traffic levels because it broadcasts requests on every cache miss. Network traffic can be dramatically reduced when the directory protocol is employed (51% on average). This is because requests are only sent to the home tile, which in turn sends coherence messages just to the L1 caches that must receive them.

In Figure 11 (top), we show the network traffic split into three categories: data, control and hints traffic. For some applications data traffic is reduced due to the decrease of cache misses that DiCo-CMP can entail, as commented in the previous section. Compared to the directory protocol, DiCo-CMP meaningfully saves the traffic generated by control messages. This saving is originated by the elimination of control messages between the home tile and the owner cache that DiCo-CMP entails. This reduction allows *DiCo-Base* to reduce network traffic by 6% compared to the directory protocol. In contrast, DiCo-CMP introduces hint messages for some configurations in order to achieve more accurate owner predictions. The hints that appear for *DiCo-Base* come as consequence of evictions of owner blocks, as explained in Section 4. In general, hints increase network traffic, especially in *Unstructured* in which they are crucial to obtain good performance. However, this traffic is always lower than the reached by Token-CMP because hints are only sent (if necessary) when the owner cache changes. *DiCo-Hints AS* requires more traffic than *DiCo-Hints FS* at the cost of

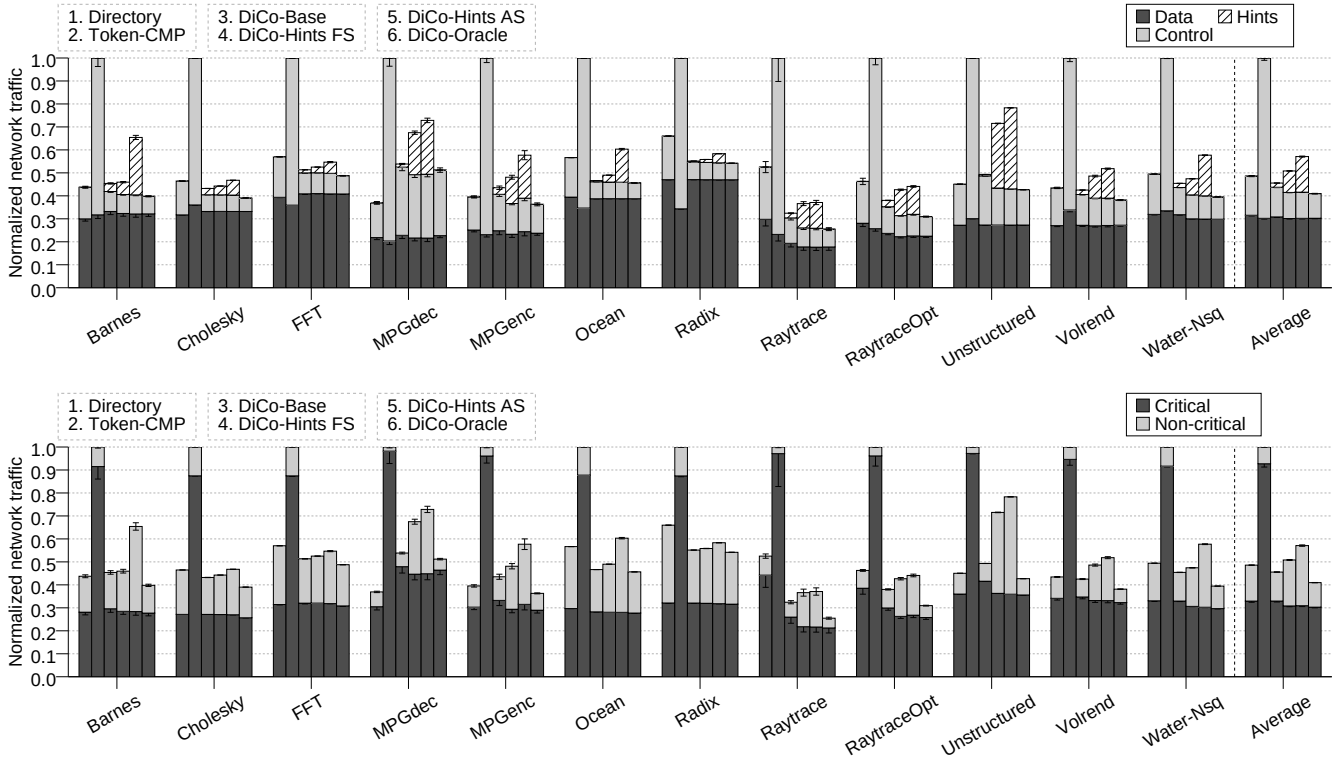


Fig. 11. Normalized network traffic for the applications evaluated in this work.

reducing considerably the storage requirements for the hints mechanism. In general, we can observe that *DiCo-Hints AS* increases network traffic by 14% compared to the directory protocol, although it still reduces the traffic compared to Token-CMP up to 43%.

Figure 11 (bottom) shows the network traffic split into critical and non-critical messages. This classification is important to know how each protocol can be optimized under heterogeneous networks [14], in which non-critical messages can be sent through low-power wires to save power consumption. In Token-CMP all broadcast requests are considered critical because it is unknown which ones are going to be actually in the critical path. Directory highly reduces the amount of critical traffic with respect to Token-CMP. We can also observe that DiCo-CMP reduces even more this kind of traffic because hints are out of the critical path of the miss. Therefore, under heterogeneous networks, DiCo-CMP can save more power consumption and even other more aggressive hints policies, as broadcasting hints on every owner change, can be implemented with small overhead in terms of power.

7.4 Impact on execution time

The ability of avoiding indirection and the low network traffic requirements that DiCo-CMP shows translates into applications' execution time. Figure 12 plots the average execution times that are obtained for the applications evaluated in this paper. All the results have been normalized with respect to those observed for the directory protocol.

In general, we can see from Figure 12 that Token-CMP improves the execution times of the directory protocol by 2% on average. As already discussed, Token-CMP avoids indirection by broadcasting requests to all caches and, consequently, the network contention can become critical, even when a network with multicast support is assumed. For applications with high level of contention, like *MPGdec*, *MPGenc* and *Raytrace*, Token-CMP has to issue persistent requests for a significant number of cache misses which make it get worse performance.

DiCo-CMP does not rely on broadcasting but requests are just sent to the potential owner cache. It is clear that the performance achieved by DiCo-CMP will depend on its ability to find the current owner cache. We observe improvements in execution time for *DiCo-Base* of 3% compared to the directory protocol, obtaining similar performance than Token-CMP. On the other hand, the use of hints increases the fraction of two-hop misses which translates into increased gains in terms of execution time. Both *DiCo-Hints FS* and *DiCo-Hints AS* improve execution time by 6% over a directory protocol and 3% over Token-CMP. We also can see that the *DiCo-Hints AS* policy obtains average results very close to the unimplementable *DiCo-oracle* policy. Particularly, for the *MPGdec* application, the hints policies achieve lower execution time than the oracle one. This is because, in case of contention, always predicting the owner node can be unfair for the misses that take place in the tiles farther to the owner cache, thus introducing more starved requests.

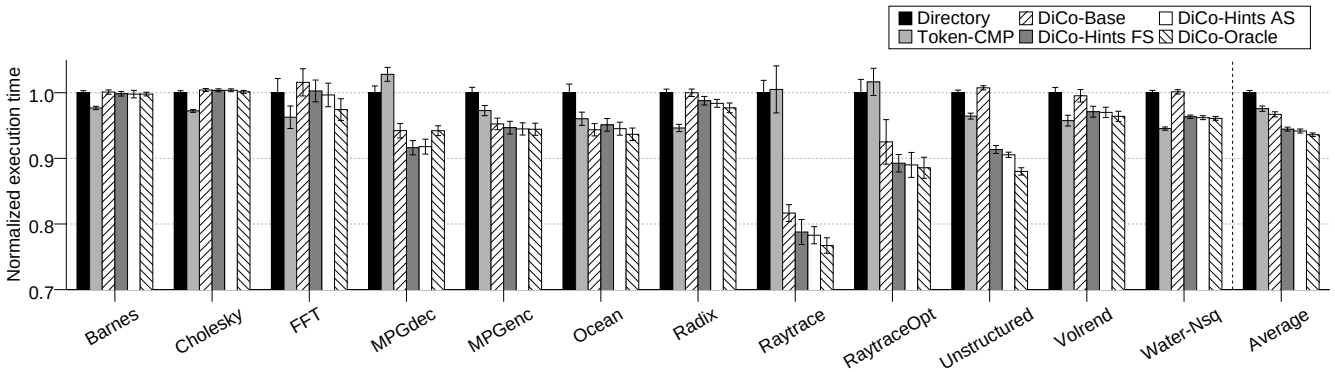


Fig. 12. Normalized execution time for the applications evaluated in this work.

8 Conclusions

Tiled CMP architectures have recently emerged as a scalable alternative to current small-scale CMP designs, and will be probably the architecture of choice for future many-core CMPs. Differences in the technological parameters and constraints entailed by CMPs with respect to traditional shared-memory multiprocessors demand new solutions to the cache coherence problem.

In this work, we present direct coherence, a cache coherence protocol for tiled CMP architectures that meets the advantages of directory and token protocols and avoids their problems. In direct coherence, the task of storing up-to-date sharing information and ensuring ordered accesses for every memory block is assigned to the owner cache. Compared to a directory protocol, our proposal avoids the indirection that the access to the directory entails by directly sending the requests to the owner cache and, therefore, it also reduces the cache miss latency. Compared to Token-CMP, our proposal reduces network traffic by sending just one request message on every cache miss. We name the implementation of direct coherence for CMPs as DiCo-CMP, and we evaluate different policies of this implementation.

In this way, we show that *DiCo-Hints AS*, the best policy for DiCo-CMP, is able to reduce the indirection compared to a directory protocol from 54% to 26% on average. Both this reduction in misses with indirection and the decrease in the waiting time for some applications that DiCo-CMP achieves result in reductions of 12% in the latency of cache misses, on average. Finally, the improvements obtained for the cache miss latencies lead to improvements of 6% in execution time. Moreover, *DiCo-Hints AS* achieves network traffic reductions of 43% compared to Token-CMP by sending just one request message per miss, and consequently, the total power consumed in the interconnection network will be also reduced. The savings allow *DiCo-Hints AS* achieve improvements of 3% in terms of execution time compared to Token-CMP. Additionally, we show that the structures and complexity required by DiCo-CMP are comparable to those used in a directory protocol, which confirms that our proposal is a viable alternative to current cache coherence protocols for tiled CMPs.

Acknowledgments

The authors would like to thank the anonymous reviewers for their detailed comments and valuable suggestions, which have helped to make the paper sounder. This work has been jointly supported by Spanish Ministry of Ciencia e Innovación under grant "TIN2006-15516-C04-03" and European Comission FEDER funds under grant "Consolider Ingenio-2010 CSD2006-00046". Alberto Ros is supported by a research grant from Spanish MEC under the FPU national plan (AP2004-3735).

References

[1] M. E. Acacio, J. González, J. M. García, and J. Duato. Owner prediction for accelerating cache-to-cache transfer misses in cc-NUMA multiprocessors. In *SC2002 High Performance Networking and Computing*, pages 1–12, Nov. 2002.

[2] M. E. Acacio, J. González, J. M. García, and J. Duato. The use of prediction for accelerating upgrade misses in cc-NUMA multiprocessors. In *11th PACT*, pages 155–164, Sept. 2002.

[3] N. Agarwal, L.-S. Peh, and N. K. Jha. In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects. In *15th HPCA*, pages 67–78, Feb. 2009.

[4] A. R. Alameldeen and D. A. Wood. Variability in architectural simulations of multi-threaded workloads. In *9th HPCA*, pages 7–18, Feb. 2003.

[5] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A scalable architecture based on single-chip multiprocessing. In *27th ISCA*, pages 12–14, June 2000.

[6] B. M. Beckmann, M. R. Marty, and D. A. Wood. ASR: Adaptive selective replication for CMP caches. In *39th MICRO*, pages 443–454, Dec. 2006.

[7] M. Björkman, F. Dahlgren, and P. Stenström. Using hints to reduce the read miss penalty for flat COMA protocols. In *28th Int'l Conference on System Sciences*, pages 242–251, Jan. 1995.

[8] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, July 1970.

[9] K. D. Bosschere, W. Luk, X. Martorell, N. Navarro, M. O'Boyle, D. Pnevmatikatos, A. Ramírez, P. Sainrat, A. Seznec, P. Stenström, and O. Temam. High-performance embedded architecture and compilation roadmap. *Transactions on HiPEAC I*, pages 5–29, Jan. 2007.

[10] J. F. Cantin, J. E. Smith, M. H. Lipasti, A. Moshovos, and B. Falsafi. Coarse-grain coherence tracking: RegionScout and region coherence arrays. *IEEE Micro*, 26(1):70–79, Jan. 2006.

[11] L. M. Censier and P. Feautrier. A new solution to coherence problems in multicache systems. *IEEE Transactions on Computers*, 27(12):1112–1118, Dec. 1978.

[12] L. Ceze, J. Tuck, C. Cascaval, and J. Torrellas. Bulk disambiguation of speculative threads in multiprocessors. In *33rd ISCA*, pages 227–238, June 2006.

[13] L. Cheng, J. B. Carter, and D. Dai. An adaptive cache coherence protocol optimized for producer-consumer sharing. In *13th HPCA*, pages 328–339, Feb. 2007.

[14] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter. Interconnect-aware coherence protocols for chip multiprocessors. In *33rd ISCA*, pages 339–351, June 2006.
[15] J. Duato, S. Yalamanchili, and L. M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers, Inc., 2002.
[16] N. D. Enright-Jerger, L.-S. Peh, and M. H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast tree for scalable cache coherence. In *41th MICRO*, pages 35–46, Nov. 2008.
[17] H. Hossain, S. Dwarkadas, and M. C. Huang. Improving support for locality and fine-grain sharing in chip multiprocessors. In *17th PACT*, pages 155–165, Oct. 2008.
[18] H.-C. Hsiao and C.-T. King. Boosting the performance of now-based shared memory multiprocessors through directory hints. In *20th ICDCS*, pages 602–609, Apr. 2000.
[19] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA substrate for flexible CMP cache sharing. In *19th ICS*, pages 31–40, June 2005.
[20] D. Kanter. The common system interface: Intel's future interconnect. *Real World Technologies*, Aug. 2007.
[21] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th ASPLOS*, pages 211–222, Oct. 2002.
[22] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *32nd ISCA*, pages 408–419, June 2005.
[23] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51(6):639–662, Nov. 2007.
[24] M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The ALPBench benchmark suite for complex multimedia applications. In *Int'l Symp. on Workload Characterization*, pages 34–45, Oct. 2005.
[25] N. Magen, A. Kolodny, U. Weiser, and N. Shamir. Interconnect-power dissipation in a microprocessor. In *Int'l workshop on System Level Interconnect Prediction*, pages 7–13, Feb. 2004.
[26] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.
[27] M. M. Martin. *Token Coherence*. PhD thesis, University of Wisconsin-Madison, Dec. 2003.
[28] M. M. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors. In *30th ISCA*, pages 206–217, June 2003.
[29] M. M. Martin, M. D. Hill, and D. A. Wood. Token coherence: Decoupling performance and correctness. In *30th ISCA*, pages 182–193, June 2003.
[30] M. M. Martin, D. J. Sorin, A. Ailamaki, A. R. Alameldeen, R. M. Dickson, C. J. Mauer, K. E. Moore, M. Plakal, M. D. Hill, and D. A. Wood. Timestamp snooping: An approach for extending SMPs. In *9th ASPLOS*, pages 25–36, Nov. 2000.
[31] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, 33(4):92–99, Sept. 2005.
[32] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. Martin, and D. A. Wood. Improving multiple-cmp systems using token coherence. In *11th HPCA*, pages 328–339, Feb. 2005.
[33] K. Olukotun, B. A. Nayfeh, L. Hammond, K. G. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *7th ASPLOS*, pages 2–11, Oct. 1996.
[34] V. Puente, J. A. Gregorio, and R. Beivide. SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems. In *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 15–22, Jan. 2002.
[35] A. Ros, M. E. Acacio, and J. M. García. Direct coherence: Bringing together performance and scalability in shared-memory multiprocessors. In *14th HiPC*, pages 147–160, Dec. 2007.
[36] A. Ros, M. E. Acacio, and J. M. García. DiCo-CMP: Efficient cache coherency in tiled CMP architectures. In *22nd IPDPS*, pages 1–11, Apr. 2008.
[37] A. Ros, M. E. Acacio, and J. M. García. Scalable directory organization for tiled CMP architectures. In *Int'l Conference on Computer Design (CDES)*, pages 112–118, July 2008.
[38] M. Shah, J. Barreh, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Saha, D. Sheahan, L. Spracklen, and A. Wynn. UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC. In *IEEE Asian Solid-State Circuits Conference*, pages 22–25, Nov. 2007.
[39] P. Stenström, M. Brorsson, and L. Sandberg. An adaptive cache coherence protocol optimized for migratory sharing. In *20st ISCA*, pages 109–118, May 1993.
[40] H. Wang, L.-S. Peh, and S. Malik. Power-driven design of router microarchitectures in on-chip networks. In *36th MICRO*, pages 105–111, Dec. 2003.
[41] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *22nd ISCA*, pages 24–36, June 1995.
[42] L. Yen, J. Bobba, M. R. Marty, K. E. Moore, H. Volos, M. D. Hill, M. M. Swift, and D. A. Wood. LogTM-SE: Decoupling hardware transactional memory from caches. In *13th HPCA*, pages 261–272, Feb. 2007.
[43] M. Zhang and K. Asanović. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *32nd ISCA*, pages 336–345, June 2005.

Alberto Ros received the MS degree in Computer Science from the Universidad de Murcia, Spain, in 2004. In 2005, he joined the Computer Engineering Department (DITEC) at the same university as a PhD student with a fellowship from the Spanish government, receiving the PhD degree in computer science in 2009. He is working on designing and evaluating scalable coherence protocols for shared-memory multiprocessors. His research interests include cache coherence protocols, memory hierarchy designs, and scalable cc-NUMA and chip multiprocessor architectures.

Manuel E. Acacio received the MS and PhD degrees in computer science from the Universidad de Murcia, Spain, in 1998 and 2003, respectively. He joined the Computer Engineering Department, Universidad de Murcia, in 1998, where he is currently an Associate Professor of computer architecture and technology. His research interests include prediction and speculation in multiprocessor memory systems, multiprocessor-on-a-chip architectures, power-aware cache-coherence protocol design, fault tolerance, and hardware transactional memory systems.

José M. García received a MS degree in Electrical Engineering and a PhD degree in Computer Engineering in 1987 and 1991 respectively, both from the Technical University of Valencia (Spain). Prof. García is currently serving as Dean of the School of Computer Science at the University of Murcia (Spain). From 1995 to 1997 he served as Vice-Dean of the School of Computer Science, and also as Director of the Computer Engineering Department from 1998 to 2004. He is professor in the Department of Computer Engineering, and also Head of the Research Group on Parallel Computer Architecture.

He has developed several courses on Computer Structure, Peripheral Devices, Computer Architecture, Parallel Computer Architecture and Multicomputer Design. He specializes in computer architecture, parallel processing and interconnection networks. His current research interests lie in high-performance coherence protocols and fault tolerance for Chip Multiprocessors (CMPs), and parallel aplications for GPUs. He has published more than 110 refereed papers in different journals and conferences in these fields.

Prof. García is member of HiPEAC, the European Network of Excellence on High Performance and Embedded Architecture and Compilation. He is also member of several international associations such as the IEEE and ACM, and also member of some European associations (Euromicro and ATI).