Journal of Systems Architecture 56 (2010) 77–87

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

# A scalable organization for distributed directories

Alberto Ros *, Manuel E. Acacio, José M. García

*Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, 30100 Murcia, Spain*

A R T I C L E I N F O

*Article history:*
Received 11 March 2009
Received in revised form 15 July 2009
Accepted 16 November 2009
Available online 24 November 2009

*Keywords:*
Chip multiprocessors
Many-core CMPs
Cache coherence
Distributed directory
Scalability
Duplicate tags
Implicit replacements

A B S T R A C T

Although directory-based cache-coherence protocols are the best choice when designing chip multiprocessors with tens of cores on-chip, the memory overhead introduced by the directory structure may not scale gracefully with the number of cores. Many approaches aimed at improving the scalability of directories have been proposed. However, they do not bring perfect scalability and usually reduce the directory memory overhead by compressing coherence information, which in turn results in extra unnecessary coherence messages and, therefore, wasted energy and some performance degradation. In this work, we present a distributed directory organization based on duplicate tags for tiled CMP architectures whose size is independent on the number of tiles of the system up to a certain number of tiles. We demonstrate that this number of tiles corresponds to the number of sets in the private caches. Additionally, we show that the area overhead of the proposed directory structure is 0.56% with respect to the on-chip data caches. Moreover, the proposed directory structure keeps the same information than a non-scalable full-map directory. Finally, we propose a mechanism that takes advantage of this directory organization to remove the network traffic caused by replacements. This mechanism reduces total traffic by 15% for a 16-core configuration compared to a traditional directory-based protocol.

## 1. Introduction

Nowadays, the most efficient way of organizing the increasing number of transistors per chip is to integrate multiple processor cores in the same chip. Recent examples of these chip multiprocessors (CMP) are, among others, the 2-core IBM Power6 [14] and the 8-core Sun T2 [26]. These CMPs typically connect the cores through an on-chip shared bus or crossbar. However, the area required by these interconnects as the number of cores grows has to be increased to the point of becoming impractical.

CMP architectures that integrate tens of processor cores (usually known as many-core CMPs) are expected for the near future, after Intel unveiled recently the 80-core Polaris prototype [4]. Particularly, tiled CMP architectures [30,34], which are designed as arrays of identical or close-to-identical building blocks (tiles) connected over a point-to-point network, are a scalable alternative to current small-scale CMP designs and will help in keeping complexity manageable. In most current proposals, each tile contains at least one level of cache memory that is private to the local core (the L1 in this work), and the first level of shared cache (commonly, the L2 cache) is physically distributed between the tiles of the system, as shown in Fig. 1. Private caches are kept coherent in hardware by using a cache-coherence protocol.

In CMP architectures, the cache-coherence protocol is a key component since it can add requirements of area and power consumption to the final design and, therefore, it could restrict severely its scalability. When the CMP is comprised of a large number of cores, the best way of keeping cache coherence is by implementing a directory-based protocol, since protocols based on broadcasting requests are not power-efficient due to the tremendous number of messages that they would generate.

Directory-based cache-coherence protocols reduce power consumption compared to broadcast-based protocols because they keep track of the sharers of each block in a directory structure. In a traditional directory organization, each directory entry stores the sharers for each memory block through a simple bit-vector or full-map sharing code, i.e., one bit per private cache. Since the area requirements of this structure grow linearly with the number of cores in the CMP, many approaches aimed at improving its scalability have been proposed [1,3,7,9,20]. However, they do not bring perfect scalability and usually reduce the directory memory overhead by compressing the coherence information, which in turn results in extra unnecessary coherence messages, wasted energy, and some performance degradation. Another alternative to the full-map scheme that also keeps precise sharing information is to have a directory structure that stores duplicate tags of the blocks held in the private caches. This scheme has been recently used both in cc-NUMA machines as Everest [21] and in CMPs as Piranha [5] or Sun UltraSPARC T2 [29].

* Corresponding author.
*E-mail addresses:* a.ros@ditec.um.es (A. Ros), meacacio@ditec.um.es (M.E. Acacio), jmgarcia@ditec.um.es (J.M. García).

1383-7621/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2009.11.006

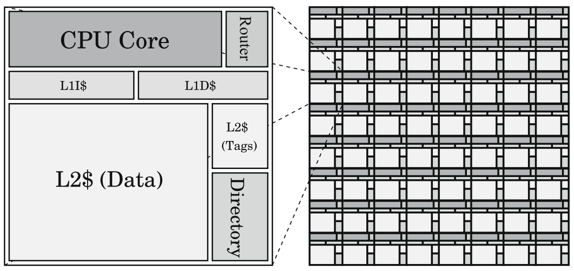


**Fig. 1.** Organization of a tile and a $8 \times 8$ tiled CMP.

In tiled CMPs, the directory structure is split into banks which are distributed across the tiles. Each directory bank tracks a particular range of memory blocks. Up to now, most tiled CMP proposals assume a straightforward implementation for the directory structure based on the use of a full-map sharing code. This directory organization does not scale, since its size grows linearly with the number of tiles of the system. Moreover, since the directory must be stored on-chip to allow for short cache miss latencies and CMP designs are constrained by area, the directory area should represent a small fraction of the total chip.

In this work, we show that a directory organization based on duplicate tags, which are distributed among the tiles of a tiled CMP by following a particular granularity of interleaving can scale up to a certain number of cores, while still storing precise coherence information. In particular, we show that the size of each directory bank does not depend on the number of tiles. In the proposed directory organization, each directory entry has associated a unique entry of a private cache in the system. A directory entry stores the tag of the block allocated in its corresponding entry of the private cache, a valid bit and an ownership bit. If the ownership bit is enabled the cache is known to be the owner of the block.

The size of each directory bank in the proposed organization is $c \times (l_t + 2)$, where $c$ is the number of entries of the last-level private cache if the private caches are inclusive or the aggregate number of entries of all private caches if they are non-inclusive, and $l_t$ is the size of the tag field. To ensure that each directory entry is associated with just one entry of some private cache, and vice versa, the directory interleaving must be defined by taking some bits of the memory address that fulfill the following condition: $bits\_home \subseteq bits\_private\_cache\_set$, i.e., the bits that define the home tile must be a subset of the bits used for indexing private caches. We also have measured the area overhead of the proposed directory organization using the CACTI tool [31], obtaining an overhead of just 0.56% when compared to the on-chip data caches considered in this work.

Designing large-scale CMPs is not straightforward, and tiled CMPs are aimed at simplifying the development of these multiprocessors by duplicating identical or close-to-identical building blocks. Additionally, this allows processor vendors to support families of products with varying computational power, and thus, cost. The proposed scalable distributed directory organization will allow vendors to use the same building block for designing tiled CMPs with different number of tiles.

Additionally, this directory organization allows us to modify the coherence protocol in order to remove extra messages caused by replacements. Since each cache entry is associated to a directory entry (the same way too in case of associative caches) the requesting tile does not have to inform the directory about replacements, because the directory knows which block is being replaced when the request for a new block arrives to it. We name this technique as *implicit replacements*. We have found that this mechanism leads to average reductions of 15% and 13% for 16 and 32-core configurations respectively compared to a non-scalable traditional directory-based protocol that employs unlimited directory caches and informs the directory about replacements only in case of evictions of dirty blocks. Moreover, compared to a directory organization based on duplicate tags that also needs to inform the directory about evictions of clean blocks, the implicit replacement mechanism saves 35% of coherence messages on average for a 16-core configuration and 32% when 32 cores are simulated. These reductions in network traffic are expected to result in significant savings in terms of power consumption. These results are slightly better for a smaller number of cores because the working set accessed by each core is greater in that case and, therefore, less replacements take place.

A preliminary and partial version of this work was presented in Ref. [25]. Here, we significantly extend that work with a more formal description of the directory organization and the mapping function, and an analysis of the scalability limits and how those limits can be addressed. Moreover, we extend the evaluation process, including multimedia applications in addition to the suite of scientific benchmarks already considered, evaluating of 32-tile CMP in addition to the 16-tile CMPs evaluated in the preliminary work, implementing other cache-coherence protocols that use compressing sharing codes, and including a detailed simulator for modeling the interconnection network (SiCoSys [23]). Finally, we also study the area required by the different directory structures by using the CACTI [31] tool.

The rest of the paper is organized as follows. A background on directory organizations for both cc-NUMA and CMP systems is given in Section 2. Section 3 describes the scalable distributed directory organization. The implicit replacements mechanism is presented in Section 4. Section 5 introduces the methodology employed in the evaluation. Section 6 shows the area requirements of the directory organization and the savings in network traffic obtained with the implicit replacements mechanism. Section 7 studies the limitations of the presented directory organization and how to avoid them. And finally, Section 8 concludes the paper.

## 2. Background on directory organizations

Directory-based cache-coherence protocols have been used for long in shared-memory multiprocessors. Unfortunately, the size of the directory structure does not scale with the number of nodes of the system. It has a complexity order of $O(nm)$ when track of the sharers of every memory block $(m)$ is kept through a full-map sharing code ($n$ bits, where $n$ is the number of cores). In this work, we study a directory organization for tiled CMPs that addresses this problem. Firstly, we review some of the previous proposals to reduce directory storage for cc-NUMA architectures which can be also used in CMPs, and then, we discuss the directory organizations implemented in current CMP proposals and architectures.

### *2.1. cc-NUMA architectures*

Some proposals reduce the width of directory entries by using compressed sharing codes instead of full-map. For example, *coarse vector* is based on using each bit of the sharing code for a group of K processors (a bit is set if at least one of the processors in the group holds the memory block). Another compressed sharing codes are *tristate* [3] (also called superset scheme), *Gray-tristate* [20] or *binary tree with subtrees* [2]. Other authors propose to have a limited number of pointers per entry, which are chosen for covering the common case [7,28], and overflow situations are handled by broadcasting invalidation messages or eliminating one of the existing copies.

Other proposals try to reduce the directory height by combining several entries into a single one (*directory entry combining*) [27]. An alternative way is to organize the directory structure as a cache (*sparse directory*) [22,9], or include this information in the tags of private caches [24], thus reducing the height of the directory down to the height of the private caches. All these proposals are based on the observation that only a small fraction of the memory blocks can be stored in the private caches at a particular moment of time.

Unfortunately, these techniques result in extra coherence messages being sent or in increased cache miss rates, reducing the directory memory overhead at the expense of performance and/or power (as a consequence of an increase in network traffic).

The idea of having duplicate tags has also been used for distributed shared-memory multiprocessors as, for example, by Nanda et al. in Everest [21]. In Everest, the directory structure or complete and concise remote (CCR) directory keeps the state information (tag and state) of the memory blocks belonging to the local home that are cached in the remote nodes. In this way, CCR directory contains the same amount than memory as a sparse directory and keeps the same information than a full-map directory. However, the number of entries in the CCR directory grows linearly with the number of cores in the system.

On the other hand, other authors studied the directory interleaving to reduce the size of a distributed directory that stores a linked list of pointers to the sharers of each cache block [13]. An interleaving consisting in taking the less significant bits of the memory address allows each directory bank to have the same number of entries than the number of entries of the last-level private cache. Unfortunately, the access to the full list of pointers requires extra latency, and the size of the pointers is not completely scalable –$O(\log_2 n)$–.

## 2.2. CMP architectures

Some current small-scale CMPs keep cache coherence by implementing a snooping-based protocol, such as the IBM POWER6 architecture [14]. However, this architecture also employs directory states to filter some unnecessary coherence messages.

Other CMPs that implement a directory-based cache-coherence protocol use duplicate (or shadow) tags to keep the coherence information, such as the Piranha [5] or Sun UltraSPARC T2 [29] architectures. In this case, each directory entry has fixed size and is comprised of a tag and a state field. The number of directory entries required to keep track of all blocks stored in the private caches corresponds to the sum of the entries of all private caches. In Piranha, this directory structure is centralized and, therefore, it increases with the number of cores since each core includes a private cache. Moreover, all cache misses must access this centralized directory structure, which would mean a significant bottleneck for many-core CMPs. The Sun UltraSPARC T2 architecture distributes the directory among the L2 cache banks leading to an organization similar to the studied in this work, but this architecture still uses a non-scalable crossbar as the interconnection network.

On the contrary, large-scale tiled CMPs require a distributed directory organization for scalability reasons. Essentially, each tile includes at least one level of private cache and a slice of the total directory. Each memory block is mapped to a home tile which is responsible for keeping coherence information for that block. The identity of the home tile of a block is commonly known from the address bits ($\log_2 n$ bits, where $n$ is the number of tiles). We consider that a scalable directory organization for these systems is achieved when the size of each directory slice does not vary with the number of tiles. Obviously, the number of directory slices increases proportionally with the number of tiles, but also the number of data caches. Therefore, under this assumption the overhead introduced by the directory information does not increase with respect to data as the number of tiles grows.

The two most popular ways of organizing a distributed directory in tiled CMPs are (1) the use of directory caches [19] or (2) the inclusion of a bit-vector in the first level of shared caches [10]. The first technique can result in a high directory miss rate (up to 70%, as recently reported in several studies [19,12]). The second technique avoids directory misses by using the same number of entries as the shared cache. However, this scheme can only be used when the inclusion property between private and shared caches is enforced, i.e., the shared cache must allocate an entry for each block in a private cache. In the other case, a directory cache is also needed for those blocks not allocated in the shared cache, thus introducing scalability problems and the appearance of directory misses.

Unfortunately, the inclusion property between private and shared caches could also restrict system scalability. When the number of cores grows and, therefore, the number of private caches, more pressure could be put over a particular slice of the shared cache, resulting in a larger amount of replacements. Additionally, the inclusion property forces all copies of a block to be invalidated from the private caches when the block is replaced from the shared cache, increasing the miss rate of private caches and, consequently, degrading performance. On the other hand, the size of the directory entries either does not scale with the number of cores (e.g., bit-vector) or does not keep precise sharing information (e.g., coarse vector [9] or limited pointers [7]).

Recently, in Ref. [19] different directory organizations have been studied for tiled CMPs which demonstrate that the organization for the directory is a crucial aspect when designing large-scale CMPs.

## 3. Scalable directory organization

In this section we show that a distributed directory organization based on duplicate tags can scale up to a certain number of cores depending on the system parameters. Moreover, the described directory organization keeps precise information about all blocks stored in private caches, i.e., directory misses only take place when the block is not stored in any private cache and, therefore, no extra coherence actions are needed as consequence of directory misses.

To guarantee the scalability of the directory it is necessary to keep fixed both the size of each directory entry (directory width) and the number of entries per slice (directory height). The use of duplicate tags as directory entries makes scalable the directory width, since each directory entry is comprised of a tag and a state field. On the other hand, the total number of directory entries required to track all blocks stored in the private caches should be the same as the number of entries of all private caches. This rule is always fulfilled for centralized directories, but not when the directory is distributed.

As previously discussed, tiled CMPs split the directory structure into banks which are distributed across the tiles. Each directory bank tracks a particular range of memory blocks. If all blocks stored in the private caches map to the same bank, the directory of this bank can overflow, thus requiring more entries to keep all the directory information. In this case, the minimal number of entries required to store all duplicate tags increases linearly with the number of tiles.

In the first subsection, we discuss how the granularity of the directory interleaving can affect the maximum number of entries required by each directory bank and, therefore, the scalability of the directory. Then, we define the conditions that are necessary for the directory structure to scale with the number of cores. We also describe the structure of the directory and how precise sharing information can be obtained from it. Finally, we comment on the

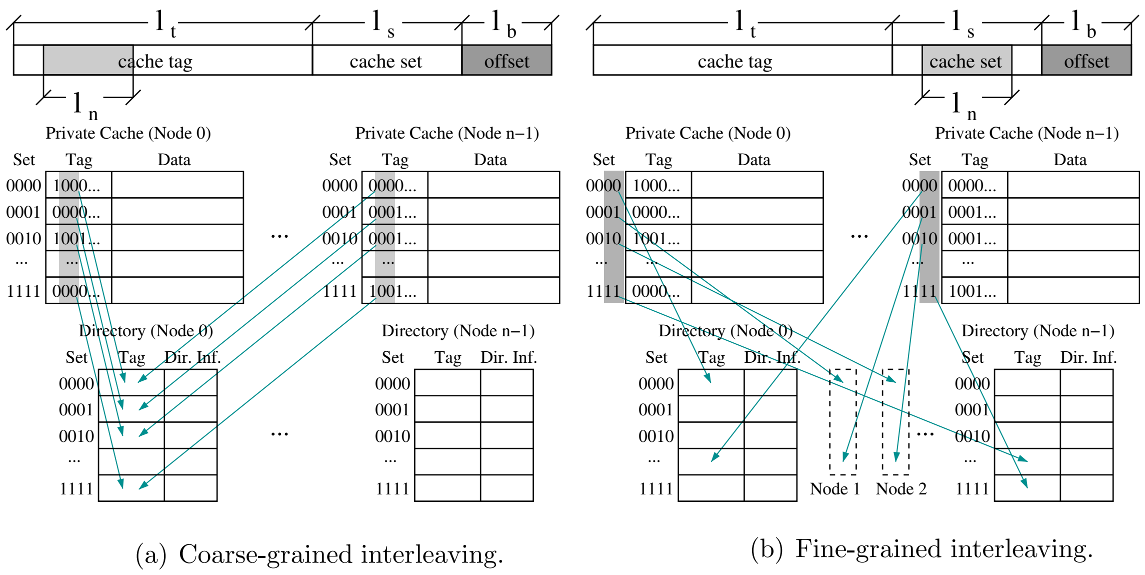


**Fig. 2.** Granularity of directory interleaving and its effect on directory size. (a) Coarse-grained interleaving. (b) Fine-grained interleaving.

requirements of a cache-coherence protocol that implements this directory organization.

## 3.1. Granularity of directory interleaving

One important decision when designing the memory hierarchy of a tiled CMP is the granularity of the directory interleaving. Each memory block must map to a particular tile, which is the home tile for that block. This tile is responsible for keeping cache coherence for that block and, therefore, must store the directory information necessary to perform that task. On the other hand, if the tiled CMP includes an on-chip cache which is logically shared by all cores (but obviously distributed among the tiles), it is also necessary to define an interleaving for that cache. Cache and directory interleavings may be different. However, this decision incurs in extra coherence messages between the tile where the directory information is stored and the tile where data can reside, thus making the cache-coherence protocol less efficient and more complex. Therefore, it is desirable that both the shared cache and directory have the same interleaving.

The directory can be easily distributed among the tiles of the CMP by taking $\log_2 n (l_n)$ bits of the block address, where $n$ is the number of tiles of the system (physical address mapping). These bits denote the tile where the directory information for each block can be found. The position of these bits defines the granularity of the interleaving, and as shown in Fig. 2, the number of entries required by each directory bank to be able to keep the sharing information of all cached blocks belonging to it.

In Fig. 2, we can observe two alternative ways of distributing the blocks among the tiles of the CMP and their consequences. Looking at the address of a memory block we can distinguish three main fields from the point of view of a private cache: the block *offset* $(l_b)$ which depends on the size of blocks stored in cache, the *cache set* $(l_s)$ in which the block must be stored and the *cache tag* $(l_t)$ used to identify a block stored in a cache.

If the $l_n$ bits chosen to define the home tile belong to the cache tag field, huge continuous memory regions map to the same tile (coarse-grained interleaving). Under this assumption, all the blocks stored in the private caches could map to the same directory bank. This situation is shown in Fig. 2a. Assuming that the number of sets and the associativity of the private caches are $s$ and $a$, respectively, the number of entries required by each directory to keep the information of the cached blocks mapped to it is $n \times s \times a$ (or $n \times c$, where $c$ is the number of entries of each last level private cache). In particular, each directory must have $s$ sets of $n \times a$ ways each.

Otherwise, if the $l_n$ bits belong to the cache set field, memory is split in a great amount of small regions that map to different tiles in a round-robin fashion (fine-grained interleaving), as shown in Fig. 2b. Under this assumption, each entry of each L1 cache maps to only one entry of the directory. Therefore, the number of entries required by each directory bank will be $s \times a$. In particular, each directory bank must have $s/n$ sets of $n \times a$ ways each one. The size required by this structure is $c$, which scales with the number of tiles of the system. The proposed directory organization uses an interleaving where the $l_n$ bits belong to the cache set fields.

## 3.2. Conditions required for ensuring directory scalability

A distributed directory organization with the same number of entries as the private caches needs a function that maps private cache entries to directory entries so that (1) a particular memory block always has its duplicate tag in the same directory bank (the home one) regardless of the cache wherein the block is stored and (2) the function is injective, i.e., one-to-one. The first rule guaranties that sharing information for a particular block can always be found in its home bank. The second one ensures that the number of directory entries corresponds to the number of cache entries, thus achieving the scalability of the directory.

To describe the aforementioned function, let's first consider systems with just one level of direct mapped private caches. Each cache entry can be uniquely defined by a tuple of $(core, set)$ and, in the same way, each directory entry can be defined by the tuple $(home, set)$. Due to the first rule, the bits used to select the home cannot be taken from the bits used to identify the core since, in that case, a block can map to any tile depending on the cache where it is stored. Therefore, the bits used to select the home must be a subset of the bits used to select the cache set, i.e., $bits\_home \subseteq bits\_private\_cache\_set$. This mapping rule guarantees a scalable distributed directory organization. However it also has a restriction. More specifically, it can only be used when the number

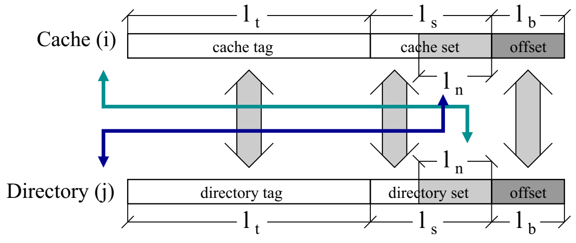


**Fig. 3.** Mapping between cache entries and directory entries.

of sets of the private cache is greater or equal than the number of tiles of the system ($num\_tiles \leqslant num\_private\_cache\_sets$). Later on, this restriction will be discussed more thoroughly.

When we consider associative caches, each cache entry can be defined by the tuple $(core, set, way)$, and each directory entry by the $(home, set, way)$ one, assuming that both structures have the same associativity. Again, due to the first rule, the bits used to select the home cannot be taken neither from the bits identifying the core nor from the bits identifying the way, so that the rule that guaranties scalability is respected.

Finally, regarding private caches organized in several cache levels, if the cache levels are inclusive the scalability is achieved in the same way as with just the last (larger) cache level. However, when the inclusion policy is not ensured, each home tile should have as many directory banks as caches in the private hierarchy, each one with the same number of entries as each cache. In this case, the scalability can be achieved when the number of tiles is not greater than the number of sets of the small cache in the hierarchy.

By using this mapping function we can see that the associativity required for each directory bank is the same as the one used for the private caches. Therefore, the associativity of the directory can be reduced from $n \times a$, as discussed in the previous section, to $a$ by taking the number of the tile in which the block is cached as part of the set bits. In this way, the number of sets grows from $s/n$ to $s$. In conclusion, each directory bank requires the same number of sets and associativity as a private cache.

Fig. 3 shows the mapping function for scalable distributed directory organizations. Home tiles are chosen by taking $l_n$ bits of the ones used to select the set in the private cache (e.g., the less significant ones). Moreover, the set in a directory bank is obtained from both the remaining bits of the cache set and the number of the tile where the block is stored. Likewise, $l_n$ bits of the directory set are used to identify the tile that holds the copy in its private cache. Although in the scheme the $l_n$ bits are the less significant ones of the set field, they can be any set of bits of that field.

When the number of tiles $n$ is greater than the number of sets of the L1 cache $s$, the number of entries required by the directory is $n \times a$, but this is not the common case. In any case, the number of entries required by this directory organization is $\max\{s, n\} \times a$, that is to say, the number of entries completely scales for values of $s$ greater than $n$.

## 3.3. Directory structure

In the previous sections we have described how the number of entries of the directory can scale with the number of tiles. However, the size of the entries commonly used to keep the directory information does not scale with the number of tiles (e.g., $n$ for a full-map sharing code, or $p \times \log_2 n$ when $p$ pointers are used to locate the cached copies). One way to keep constant the size of the directory entries is storing duplicate tags. Particularly, our proposal for a scalable directory stores in each entry the tag of the block plus two extra bits. The first bit is the valid or presence bit. If this bit is set the block is known to be stored in the cache entry associated with this directory entry. Remember that each directory entry is associated with only one cache entry (injective function). This bit is used to locate all the copies of the block on a write miss. The second bit is the ownership bit and when it is set the cache entry is known to have the ownership of the block. This bit is used to enable the implementation of a MOESI protocol, and it identifies the cache that must provide the data block on a cache miss.

Since we only store the tag of the block and two more bits in each directory entry, and the tag bits keep invariant with the number of tiles, the size of the directory keeps constant as the number of cores of the CMP increases. The total size of each directory bank is $c \times (l_t + 2)$.

Considering the mapping function presented in the previous section, the coherence information for a particular block can be obtained from the home directory bank as shown in Fig. 4. A block is stored in a particular private cache whether there is a hit in the directory bank for the corresponding set and the valid bit is enabled. The corresponding set is calculated by replacing the $l_n$ bits that identify the home directory with the $l_n$ bits that identify the tile which contains that cache. If the ownership bit is also enabled, that cache is the owner of the block. Therefore, the identifier of the set for a block depends on the cache where it is stored. By searching this information in the corresponding $n$ directory entries (one per each private cache) the complete directory information is obtained. This search can be performed in parallel to accelerate the access to the directory information.

Updating the directory information only requires to modify few bits. On a write miss, invalidation messages are sent to the sharers and their corresponding presence bits are disabled. The directory entry for the new owner of the block is with the tag of the block and both the valid and the ownership bits are enabled. Adding a new sharer only requires writing the tag of the block in the corresponding directory entry and setting the valid bit.

## 3.4. Cache-coherence protocol

The cache-coherence protocol required by this directory organization is similar to the required by a directory-based protocol that uses directory caches with a non-scalable full-map sharing code in each entry. However, the use of a limited number of duplicate tags requires some extra modifications.

As happens in Piranha [5] and Everest [21], replacements of shared blocks must be notified to the home tile. These notifications are necessary to deallocate an old directory entry, in order to allow the new block to use that entry. Notifications of these evictions make the cache-coherence protocol more complex. Moreover, to avoid race conditions replacements are usually performed in three phases, a fact that entails extra network traffic.

In systems with unordered networks, as tiled CMPs are, the request for a block can reach the home tile before the replacement caused by that block. If the directory set for that block has valid information in all the ways, the request must wait until the replacement deallocates one of the entries. This can result in extra cache miss latency. To avoid these issues and to remove the network traffic generated by replacements, we propose the implicit replacements mechanism described in the following section.

## 4. Implicit replacements

The proposed directory organization allows us to slightly modify the coherence protocol in order to remove the messages caused by replacements. This is achieved by performing the replacements in an implicit way along with the requests which cause them.

There are two main observations that allow our proposal for scalable directory organization to support implicit replacements.

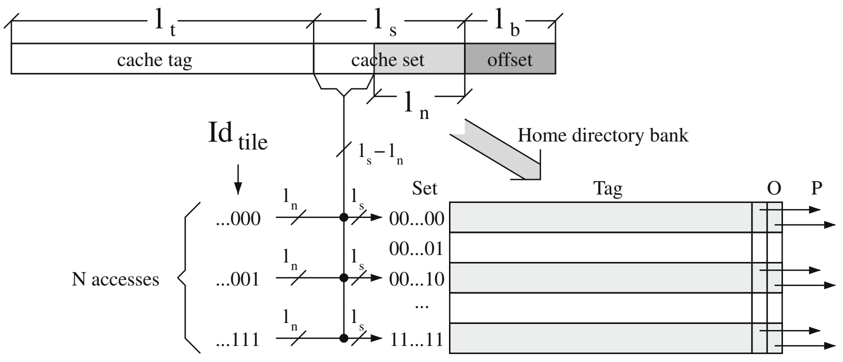


**Fig. 4.** Finding coherence information (*P* = presence bit; *O* = ownership bit).

Firstly, since we employ the bits used to select the set of private caches to associate cache entries to directory entries we ensure that the evicted and the requested blocks map to the same home tile and, therefore, to the same directory bank. Note that if a coarse-grained interleaving was chosen, these blocks could map to different directory banks (depending on the value of the tag). Secondly, each cache entry is associated with only one directory entry (the same way too), and vice versa. In this way, both the directory and the requesting cache know the address of the evicted block and it is not necessary to attach it to the request messages. Therefore, the size of coherence messages does not change considerably. It is only necessary the addition of a field informing about the way within the set where the requested block is going to be stored (2 bits in our case), which is the same way where its duplicate tag is stored in the directory.

In Fig. 5a, we can see how a replacement is usually performed. When a block must be stored in cache and the corresponding set is full, the less recently used block must be evicted (we assume a pseudo LRU eviction policy). In current directory protocols evictions of shared blocks are usually performed transparently without informing the directory. However, as previously discussed, when the directory is organized with duplicate tags, replacements must be notified even for blocks in shared state. Moreover, evictions of dirty blocks must writeback data to the next cache level. For simplicity, these writebacks are performed in a three-phase transaction as illustrated in Fig. 5a (replacement). First, the cache asks the home tile permission to writeback the block (*1 Put*). Then the home tile confirms the transaction ((*2 Ack*), and finally the block is sent to the next cache level ((*3 WrB*). Fig. 5a (Request) shows how a cache-to-cache transfer miss is solved. Requests are sent to the home tile to get the directory information (*1 Get*), and then are forwarded to the owner cache (*2 Fwd*) where the data is provided (*3 Data*), or the data is directly provided from the L2 cache in the home tile. Finally, the requesting cache informs the home tile that another cache miss for this memory block can be processed (*4 Unbl*).

Fig. 5b shows how implicit replacements are performed along with the requests that cause them. On each cache miss an MSHR (Miss Status Hold Register) entry is allocated with the information about the request. However, in our proposal we also need to store the address of the evicted block (if any), so that our MSHR has two address fields instead of one. If there is an evicted block we also allocate a new entry for it in the MSHR indicating the state of that block. Moreover, the way where the new block will be stored is specified in the request message (*1 Get/Put*). When this message reaches the home tile, another two MSHR entries must be allocated, as usual. One of the entries stores both addresses. Note that storing the address of the evicted block can be replaced with a pointer to the MSHR entry where this address is stored, thus reducing its size. The acknowledge of the replacement is forwarded along with the request (*2 Fwd/Ack*). When the data arrives to the requesting cache (*3 Data/Ack*) both MSHRs are deallocated and the writeback is performed along with the unblock message (*4 Unbl/WrB*), thus allowing the directory to process the subsequent requests for both blocks. Another advantage of this protocol is that it avoids the race conditions caused by replacements that were discussed earlier, because now the replacement is implicit into the request and, therefore, both messages reach the home tile at the same time.

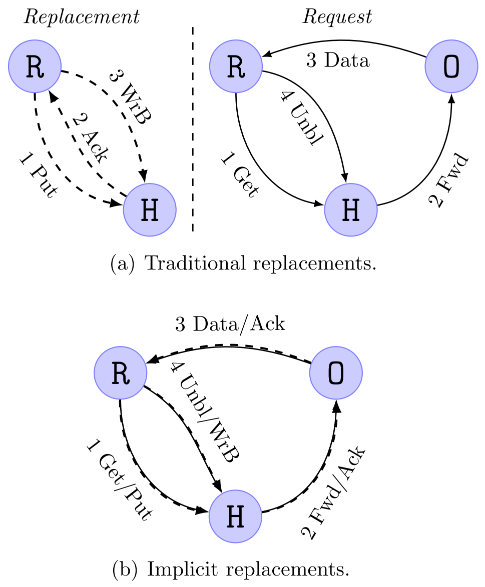


**Fig. 5.** Differences between the proposed coherence protocol and a traditional coherence protocol. (a) Traditional replacements. (b) Implicit replacements.

While the described mechanism is employed for evictions of dirty blocks, evictions of shared blocks are avoided in an easier way since we know the way within the cache set where the block is going to be stored. When the request arrives to the home tile, the directory information for the new block will be stored in the same way as in the cache. Therefore, the tag of the new block replaces the tag of the evicted block, performing the notification without requiring extra coherence messages.

## 5. Simulation environment

We evaluate our proposals with full-system simulation using Virtutech Simics [17] extended with Multifacet GEMS 1.3 [18]. GEMS provides a detailed memory system timing model which accounts for all protocol messages and state transitions. Since the network modeled by GEMS 1.3 is not very precise, we have extended it with SiCoSys [23], a detailed interconnection network simulator that allows to take into account most of the VLSI implementation details with high precision.

We simulate tiled CMP systems with one level of private cache and a shared cache distributed across the tiles. Since we consider tiled CMP designs built from a relatively large number of cores, each tile contains an in-order processor core, thus offering better performance/Watt ratio than a small number of complex cores would obtain. In particular, to show that our proposals scale with the number of tiles we present simulation results for systems with both 16 and 32 tiles.

Table 1 shows the values of the main parameters used for the evaluation. Cache latencies have been calculated using the CACTI 5.3 tool [31] for 45 nm technology. We also have used CACTI to estimate the area of the different directory structures. In this study, we assume that the length of the physical address is 40 bits like, for example, in the SUN T2 architecture [29]. Moreover, we have implemented all the protocols evaluated in Section 6 by using the language for implementing cache-coherence protocols (SLICC) included in GEMS. These implementations have been exhaustively checked using a tester program provided by GEMS that checks all race conditions to raise any incoherence.

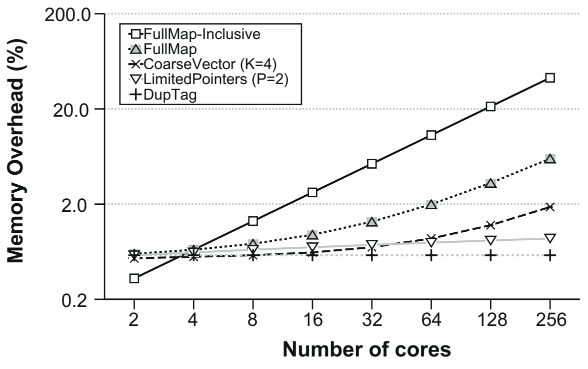
The ten applications used in our simulations cover a variety of computation and communication patterns. *Barnes* (8192 bodies, 4 time steps), *FFT* (256K points), *Ocean* ($130 \times 130$ ocean), *Radix* (1M keys, 1024 radix), *Raytrace* (teapot), *Volrend* (head) and *Water-Nsq* (512 molecules, 4 time steps) are scientific applications from the SPLASH-2 benchmark suite [33]. *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 [15]. We account for the variability in multithreaded workloads 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.

## 6. Evaluation results

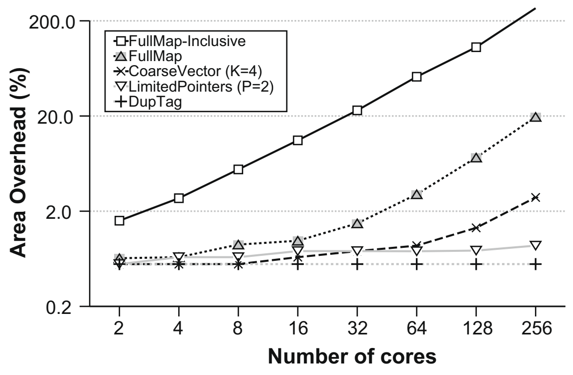
This section analyzes the area requirements of the proposed directory organization and the network traffic that can be saved by the implicit replacements mechanism. Area requirements have been calculated using the CACTI tool, whilst network traffic has been measured using the GEMS simulator enhanced with SiCoSys.

**Table 1**
System parameters.

| | |
|---|---|
| *Memory parameters (GEMS)* | |
| Processor frequency | 4 GHz |
| Cache hierarchy | Non-inclusive |
| Cache block size | 64 bytes |
| Split L1 I & D caches | 64KB, 4-way |
| L1 cache hit time | 2 (tag) + 2 (data) cycles |
| Shared unified L2 cache | 512KB/tile, 8-way |
| L2 cache hit time | 2 (tag) + 4 (data) cycles |
| Directory cache hit time | 2 Cycles |
| Memory access time | 200 Cycles |
| *Network parameters (SICOSYS)* | |
| Network frequency | 2 GHz |
| Topology | 2-Dimensional mesh |
| Switching technique | Wormhole |
| Routing technique | Deterministic X-Y |
| Data and control message size | 4 Flits and 1 flit |
| Routing time | 1 Cycle |
| Switch time | 1 Cycle |
| Link latency (one hop) | 2 Cycles |
| Link bandwidth | 1 Flit/cycle |



(a) Overhead in terms of bits.



(b) Overhead in terms of area ($mm^2$).

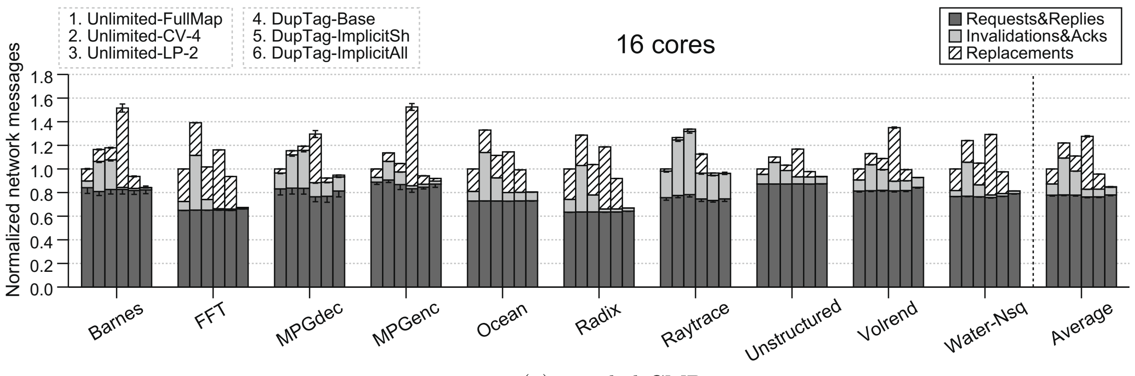
**Fig. 6.** Directory memory overhead as a function of the number of tiles. (a) Overhead in terms of bits. (b) Overhead in terms of area ($mm^2$).

### 6.1. Directory memory overhead

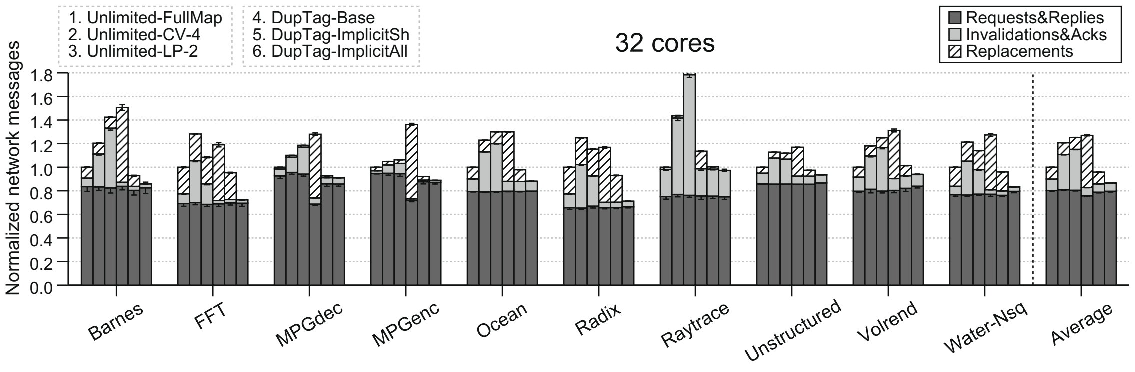
In this section, we study the directory memory overhead of our proposed organization compared to some of the schemes described in Section 2. Fig. 6 shows this overhead as a function of the number of tiles in the system. The directory organizations shown in the graphs are *FullMap-Inclusive*, *FullMap*, *CoarseVector* ($K = 4$), *limited pointers* ($P = 2$), and finally the organization presented in this work (*DupTag*). The characteristics of all these schemes are described below. The overhead of the directory structure has been calculated with respect to both the L1 and the L2 caches taking into account the size of the tag field. Fig. 6a shows the directory memory overhead in terms of number of bits added by the coherence information (and the tags that the storage of the coherence information entails). Fig. 6b shows the directory area overhead in terms of $mm^2$. For both graphs, we assume a tiled CMP with the parameters described in the previous section.

*FullMap-Inclusive* is currently used in some proposed tiled CMPs in which the L1 and the L2 are inclusive (the L2 contains all blocks held in the private L1 caches). The directory is stored in the tags' part of the L2 caches, thus removing the need of an extra directory cache. As discussed in Section 2.2, enforcing the inclusion property between private and shared caches may not be scalable. Moreover, the use of a full-map sharing code and the fact that it is introduced one sharing code field per L2 cache entry makes the overhead of this scheme increase linearly with the number of cores.

Another approach aimed at reducing the directory storage is to employ on-chip directory caches. For this evaluation, we assume directory caches with the same number of entries (same number



(a) 16-tiled CMP.



(b)32-tiledCMP.

**Fig. 7.** Reductions in the number of coherence messages. (a) 16-tiled CMP. (b) 32-tiled CMP.

of sets and associativity) as a private L1 cache. Note that due to the limited number of entries used in each directory cache, it could be needed to re-use an existing entry to store directory information for a new block. This implies invalidating all copies of a block when its directory information is evicted from the directory cache. However, this option can increase the miss rate of private caches. Another option is keeping an off-chip directory with the evicted, but this scheme results in extra storage, or broadcasting requests to all cores when a directory entry for a particular block is not found in the directory cache, but this approach results in extra network usage. As discussed in Section 3.1, for a fine-grained interleaving and an associativity of $a \times n$, all necessary directory information can be stored, but in this case the associativity of the directory cache is also not scalable.

In any case, when these directory caches store a full-map sharing code (*FullMap* case), the area overhead grows up to 20% for 256 tiles, which is prohibitive. Compressed sharing codes can reduce this overhead by losing accuracy. In *CoarseVector* ($K = 4$) the sharing code is compressed by using one bit per each group of four tiles. The bit is set if at least one of the four tiles holds a copy of the block. Although the area of the directory structure is reduced, it still increases with the number of tiles. In *limited pointers* ($P = 2$) only two pointers are used to identify the caches that share each memory block. When the number of sharers is greater than two, writes are performed by broadcasting invalidation messages (a broadcast bit is also required per entry). The area required by this organization is $2 \times \log_2 n$, which scales better than the former sharing codes. However, differently from the proposed organization, compressed sharing codes fall into extra coherence messages since they do not store precise information about all the caches that hold the blocks.

Finally, we can see that by combining a fine-grained interleaving and duplicate tags (*DupTag* case) we can achieve a completely scalable directory organization which keeps on-chip all the information necessary to keep cache coherence and, therefore, neither extra invalidation messages nor off-chip directory structures are required. The area overhead of this directory organization is 0.56% when compared to the area taken by L1 and L2 caches.

## 6.2. Reductions in number of coherence messages

In this section, we evaluate the results in terms of number of messages generated by the cache-coherence protocol. The reduction in number of messages translates into savings in 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 [16,32]. Regarding performance (execution time), all the protocols achieve similar results.

Fig. 7 shows the number of coherence messages generated by several directory organizations. This number has been normalized with respect to a directory-based protocol that uses unlimited on-chip directory caches with a full-map sharing code (*Unlimited-FullMap* case). *Unlimited-CV-4* and *Unlimited-LP-2* represent configurations with unlimited directory caches that store a coarse vector with groups of four tiles and a limited pointer scheme with two pointers, respectively. Note that by simulating unlimited directory caches we do not account for the extra invalidation messages that are necessary when a directory entry has to be evicted. Finally, we show the directory organization based on duplicate tags (*DupTag-Base*), and the optimizations entailed by the implicit replacement mechanism. *DupTag-ImplicitSh* only removes evictions of shared blocks. In *DupTag-ImplicitAll*, all evictions are performed along with request messages.

First, we can observe that the number of invalidation messages generated by the protocols with compressed sharing codes is greater than the number of invalidation messages generated by a protocol with a precise sharing code. Although the coarse vector sharing code entails less messages than the limited pointer one when 16 cores are considered, it is more traffic-efficient for a 32-core configuration. The *DupTag* configurations slightly reduce the number of invalidation messages compared to *Unlimited-FullMap*. This is because replacements of shared blocks are not notified for the *Unlimited-FullMap* scheme and, therefore, unnecessary invalidation messages, which find the block evicted from cache, are issued for some misses.

On the other hand, the implicit replacements mechanism removes the coherence messages caused by evictions. In the three first configurations shown in Fig. 7, evictions of shared blocks are performed without informing the directory structure. However, as discussed in Section 3.4, an organization based on duplicate tags requires informing the directory in case of these evictions, which increases network traffic. *DupTag-ImplicitSh* performs evictions of shared blocks in an implicit way, which reduces significantly the traffic caused by replacements. *DupTag-ImplicitAll* is more aggressive and removes all replacement messages by also merging evictions of private or owned blocks with the request that causes the eviction.

In general, the directory organization based on duplicate tags is the one with lowest invalidation messages and the implicit replacements mechanism is able to remove the coherence messages caused by L1 replacements. Average reductions of 15% are obtained for a 16-core configuration compared to *Unlimited-FullMap*. Moreover, if we compare the implicit replacements mechanism with the *DupTag-Base* configuration we can save 35% of coherence messages on average. If we consider 32 cores, the savings are slightly lower (13% compared to *Unlimited-FullMap* and 32% compared to *DupTag-Base*) because the input size of the applications is the same as having 16 cores, but it is distributed among more cores. Since each core has a smaller working set, the number of L1 replacements is reduced.

## 7. Managing scalability limits and locality issues

The directory organization presented in this work has two main limitations, which are discussed in this section. First, the scalability is limited to configurations for which the number of private cache sets is greater than the number of tiles. Second, we assume that memory blocks are distributed among the tiles in a round-robin fashion with a fine-grained interleaving. This distribution does not consider the locality of the accesses to the shared cache, and may result in longer latencies for L2 cache accesses.

### 7.1. Scalability limits

We will first focus on the scalability limitations. When the number of cores grows up to the point where there are more cores than cache sets, the number of sets required by each directory bank to allow it to store all duplicate tags must be equal to the number of cores, instead of the number of cache sets. More specifically, the number of entries required by each directory bank is $\max(c, n \times a)$, where $c$ is the number of entries of a private cache, $n$ is the number of tiles in the system, and $a$ is the associativity of private caches.

Nowadays tiled CMPs could be designed with this scalable directory. One example is the Tilera tile64 [6], which is a 64-tile CMP with 8KB direct mapped L1 instruction and data caches and 64KB 2-way associative L2 caches per tile. Both caches store blocks of 64 bytes. As we can see in Fig. 8a, cache coherence could be kept by using this directory organization, while still being scalable for up to 128 tiles if L2 caches are shared. If we consider private L2 caches (as Tilera tile64 does) the scalability limit is 512 tiles. Obviously, with private L2 caches, the directory structure requires more area, since the amount of blocks that can be allocated on private caches is higher. Considering the system that we have simulated in this work (Fig. 8b), the directory organization can scale up to 256 tiles (or 1024 in case of implementing private L2 caches). The *PrivateL2-NI* line in Fig. 8a represents the scalability of the directory when L1 and L2 private caches do not enforce inclusion.

Because it is expected that the number of tiles will increase while the size of the L1 caches will keep more or less constant, this directory organization could have only limited scalability. This may be partly remedied for future systems if they include several levels of inclusive private caches. Fortunately, it is expected that the next generation of commodity multiprocessors include two or more levels of inclusive private caches on-chip and a shared last-level on-chip cache, as happens in the new Intel Nehalem [11] and AMD Barcelona CMP architectures. In this way, the number of sets of the last-level private cache can increase as the integration scale becomes higher. As we discussed in Section 3, under this scenario the directory structure scales up to a number of tiles less or equal than the number of sets of the last-level of private caches and, therefore, it will be able to scale for a larger number of tiles.

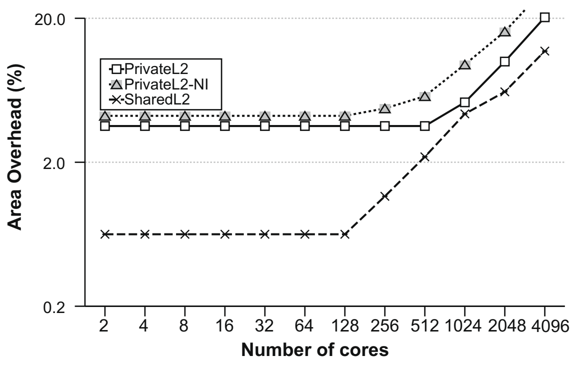
### 7.2. Locality of the accesses to the shared cache

Regarding the locality problem, there are two main ways of reducing the latency of the accesses to a shared L2 cache. One of them is to map memory regions to the tile whose processor is more frequently requesting them (e.g., a first-touch policy [8]). Another approach is to use the local L2 cache bank as a victim cache, to avoid accessing the home tile for some L1 misses (i.e., victim replication [34]).

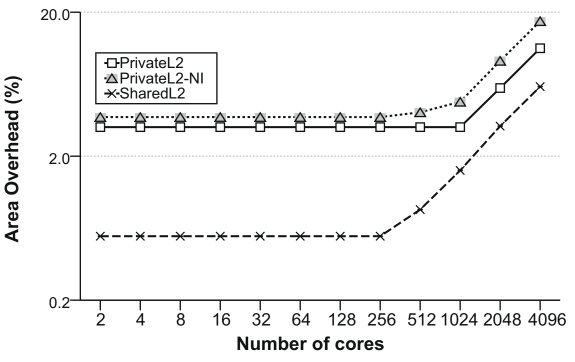
A first-touch policy maps a memory page to a tile according to the first reference to that page. On a page fault, the OS looks for a free physical page address that maps to the tile whose processor is requesting the block. Since address translation is performed at page size granularity, the granularity of the interleaving must be at least the size of a memory page. Under these assumptions, the bits that identify the home tile cannot be the less significant ones, i.e., they cannot be chosen from the page offset. As discussed in Section 3.1 a coarser granularity for the interleaving restricts even more the scalability of the directory.

A solution to this scalability problem is to change the private cache indexing, i.e., the address bits used to define the cache set. If these bits are chosen from the bits that identify the home tile, the scalability will be the same as if block-grained interleaving were used. Remember that the rule to achieve a scalable directory is $bits\_home \subseteq bits\_private\_cache\_set$. Unfortunately, this private cache indexing can increase the cache miss rate. This happens because the same bits used for identifying the home tile are used for indexing the block in the private cache, and we are trying to assign the same local home to the blocks requested by the local cache (first-touch policy). Therefore, there may be some sets that are almost unused in the private cache, thus impacting in cache hit rate.

On the other hand, victim replication is an approach that improves locality without changing the home directory, but instead, replicating blocks in the local shared slice when they are evicted from a private cache. This approach allows to use the described scalable directory organization. However, since a block can be either in any private cache or in any slice of the shared cache, the number of entries of each directory slice must be the same as the number of entries of the private and shared caches, as previously described for a non-inclusive private cache hierarchy (*PrivateL2-NI* label in Fig. 8).



(a) Tilera Tile64: 8KB direct mapped L1 cache and 64KB 2-way associative L2 cache.



(b) Simulated tiled CMP: 64KB 4-way associative L1 cache and 512KB 8-way associative L2 cache.

**Fig. 8.** Directory memory overhead for two different systems and several configurations. (a) Tilera Tile64: 8KB direct mapped L1 cache and 64KB 2-way associative L2 cache. (b) Simulated tiled CMP: 64KB 4-way associative L1 cache and 512KB 8-way associative L2 cache.

## 8. Conclusions and future work

In CMP architectures, the cache-coherence protocol is a key component since it can add requirements of area or power consumption to the final design and, therefore, could restrict severely its scalability. Although directory-based cache-coherence protocols are the best choice when designing many-core CMPs, the memory overhead introduced by the directory structure may not scale gracefully with the number of cores, specially when the coherence information is kept by using a full-map sharing code.

In this work, we demonstrate that a directory organization based on duplicate tags, which are distributed among the tiles of a tiled CMP by following a particular granularity for the directory interleaving, can scale up to a certain number of cores. The rule to achieve this scalability is $bits\_home \subseteq bits\_private\_cache\_set$. Therefore, a directory can scale meanwhile the number of cores of the CMP is less than the number of sets of the private L1 cache.

We show that, under these conditions, the size of each directory bank does not depend on the number of tiles in the system. The total size of each directory bank in the studied organization is $c \times (l_t + 2)$, where $c$ is the number of entries of the private L1 cache and $l_t$ is the size of the tag field. We also have measured the area overhead of the proposed directory organization obtaining an overhead of 0.56% with respect to the area taken by the on-chip data caches. Moreover, since the structure of each directory bank does not change with the number of tiles, the same building block could be used for systems with different number of tiles, thus making easier the design of tiled CMPs with varying number of tiles.

We have also redesigned the cache-coherence protocol to take full advantage of this directory organization. In particular, since each directory entry is mapped to only one cache entry, we can perform the replacements in an implicit way along with the requests which cause them, thus saving the network traffic introduced by replacements. This technique called *implicit replacements* leads to average reductions of 15% and 13% for 16 and 32-core configurations compared to a traditional full-map directory with unlimited caches. Moreover, compared to a directory organization based on duplicate tags that needs to inform the directory about evictions of shared blocks, the implicit replacements mechanism saves 35% of coherence messages on average for a 16-core configuration and 32% when 32 cores are considered. These reductions in network traffic finally results in significant savings in power consumption.

Finally, we also study the constrains of the proposed scalable directory organization and discuss how future chip multiprocessors can deal with them.

As part of our future work, we plan to design an approach with the best characteristics of fine-grained interleaving, i.e., the scalability of the directory, and coarse-grained interleaving, i.e., the reductions in the latency of the accesses to the shared cache. Although we have pointed out that the use of techniques like victim replication could achieve a good trade-off, it would be very interesting to perform a deeper study on this area. On the other hand, a detailed analysis of the particular amount of power that is saved with the implicit replacements mechanism is also left as future work.

## Acknowledgements

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] Manuel E. Acacio, José González, José M. Garc´ıa, and José Duato, A new scalable directory architecture for large-scale multiprocessors, in: 7th Int'l Symp. on High-Performance Computer Architecture (HPCA), January 2001, pp. 97–106.
[2] Manuel E. Acacio, José González, José M. Garc´ıa, José Duato, A two-level directory architecture for highly scalable cc-NUMA multiprocessors, IEEE Transactions on Parallel and Distributed Systems (TPDS) 16 (1) (2005) 67–79.
[3] Anant Agarwal, Richard Simoni, John L. Hennessy, Mark A. Horowitz, An evaluation of directory schemes for cache coherence, in: 15th Int'l Symp. on Computer Architecture (ISCA), May 1988, pp. 280–289.
[4] Mani Azimi, Naveen Cherukuri, Doddaballapur N. Jayasimha, et al., Integration challenges and tradeoffs for tera-scale architectures, Intel Technology Journal 11 (3) (2007) 173–184.
[5] Luiz A. Barroso, Kourosh Gharachorloo, Robert McNamara, et al., Piranha: a scalable architecture based on single-chip multiprocessing, in: 27th Int'l Symp. on Computer Architecture (ISCA), June 2000, pp. 12–14.
[6] Shane Bell, Bruce Edwards, John Amann, et al., TILE64™ processor: a 64-core SoC with mesh interconnect, in: IEEE Int'l Solid-State Circuits Conference (ISSCC), January 2008, pp. 88–598.
[7] David Chaiken, John Kubiatowicz, Anant Agarwal, LimitLESS directories: a scalable cache coherence scheme, in: 4th Int'l Conference on Architectural Support for Programming Language and Operating Systems (ASPLOS), April 1991, pp. 224–234.
[8] Sangyeun Cho, Lei Jin, Managing distributed, shared L2 caches through OS-level page allocation, in: 39th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), December 2006, pp. 455–465.
[9] Anoop Gupta, Wolf-Dietrich Weber, Todd C. Mowry, Reducing memory traffic requirements for scalable directory-based cache coherence schemes, in: Int'l Conference on Parallel Processing (ICPP), August 1990, pp. 312–321.
[10] Jaehyuk Huh, Changkyu Kim, Hazim Shafi, Lixin Zhang, Doug Burger, Stephen W. Keckler, A NUCA substrate for flexible CMP cache sharing, in: 19th Int'l Conference on Supercomputing (ICS), June 2005, pp. 31–40.

[11] Intel Corporation, White paper, First the Tick, Now the Tock: Next Generation Intel Microarchitecture, Nehalem, April 2009.
[12] Natalie D. Enright Jerger, Li-Shiuan Peh, Mikko H. Lipasti, Virtual tree coherence: leveraging regions and in-network multicast tree for scalable cache coherence, in: 41th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), November 2008, pp. 35–46.
[13] Jinseok Kong, Pen-Chung Yew, Gyungho Lee, Minimizing the directory size for large-scale shared-memory multiprocessors, IEICE Transactions on Information and Systems E88-D (11) (2005) 2533–2543.
[14] Hung Q. Le, William J. Starke, J. Stephen Fields, Francis P. O'Connell, Dung Q. Nguyen, Bruce J. Ronchetti, Wolfram M. Sauer, Eric M. Schwarz, Michael T. Vaden, IBM POWER6 microarchitecture, IBM Journal of Research and Development 51 (6) (2007) 639–662.
[15] Man-Lap Li, Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen, Eric Debes, The ALPBench benchmark suite for complex multimedia applications, in: Int'l Symp. on Workload Characterization, October 2005, pp. 34–45.
[16] Nir Magen, Avinoam Kolodny, Uri Weiser, Nachum Shamir, Interconnect-power dissipation in a microprocessor, in: Int'l Workshop on System Level Interconnect Prediction (SLIP'04), February 2004, pp. 7–13.
[17] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, et al., Simics: a full system simulation platform, IEEE Computer 35 (2) (2002) 50–58.
[18] Milo M.K. Martin, Daniel J. Sorin, Bradford M. Beckmann, et al., Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, Computer Architecture News 33 (4) (2005) 92–99.
[19] Michael R. Marty, Mark D. Hill, Virtual hierarchies to support server consolidation, in: 34th Int'l Symp. on Computer Architecture (ISCA), June 2007, pp. 46–56.
[20] Shubhendu S. Mukherjee, Mark D. Hill, An evaluation of directory protocols for medium-scale shared-memory multiprocessors, in: 8th Int'l Conference on Supercomputing (ICS), July 1994, pp. 64–74.
[21] Ashwini K. Nanda, Anthony-Trung Nguyen, Maged M. Michael, Douglas J. Joseph, High-throughput coherence control and hardware messaging in Everest, IBM Journal of Research and Development 45 (2) (2001) 229–244.
[22] Brian W. O'Krafka, A. Richard Newton, An empirical evaluation of two memory-efficient directory methods, in: 17th Int'l Symp. on Computer Architecture (ISCA) IEEE/ACM, June 1990, pp. 138–147.
[23] Valentín Puente, José A. Gregorio, Ramón Beivide, SICOSYS: an integrated framework for studying interconnection network in multiprocessor systems, in: 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, January 2002, pp. 15–22.
[24] Alberto Ros, Manuel E. Acacio, José M. García, A novel lightweight directory architecture for scalable shared-memory multiprocessors, in: 11th Int'l Euro-Par Conference, August 2005, pp. 582–591.
[25] Alberto Ros, Manuel E. Acacio, José M. García, Scalable directory organization for tiled cmp architectures, in: Int'l Conference on Computer Design (CDES), July 2008, pp. 112–118.
[26] Manish Shah, Jama Barreh, Jeff Brooks, et al., UltraSPARC T2: a highly-threaded, power-efficient, SPARC SoC, in: IEEE Asian Solid-State Circuits Conference, November 2007, pp. 22–25.
[27] Richard Simoni, Cache Coherence Directories for Scalable Multiprocessors, Ph.D. Thesis, Stanford University, 1992.
[28] Richard Simoni, Mark A. Horowitz, Dynamic pointer allocation for scalable cache coherence directories, in: Int'l Symp. on Shared Memory Multiprocessing, April 2001, pp. 72–81.
[29] Sun Microsystems, Inc., Santa Clara, CA 95054, OpenSPARC™ T2 System-on-Chip (SoC) Microarchitecture Specification, December 2007.
[30] Michael B. Taylor, Jason Kim, Jason Miller, et al., The raw microprocessor: a computational fabric for software circuits and general purpose programs, IEEE Micro 22 (2) (2002) 25–35.
[31] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, Norman P. Jouppi, Cacti 5.1. Technical Report HPL-2008-20, HP Labs, April 2008.
[32] Hangsheng Wang, Li-Shiuan Peh, Sharad Malik, Power-driven design of router microarchitectures in on-chip networks, in: 36th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), December 2003, pp. 105–111.
[33] Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder P. Singh, Anoop Gupta, The SPLASH-2 programs: Characterization and methodological considerations, in: 22nd Int'l Symp. on Computer Architecture (ISCA), June 1995, pp. 24–36.
[34] Michael Zhang, Krste Asanovic, Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors, in: 32nd Int'l Symp. on Computer Architecture (ISCA), June 2005, pp. 336–345.

**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 at the same university as a Ph.D. student with a fellowship from the Spanish government, receiving the Ph.D. 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 Ph.D. 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 Ph.D. degree in Computer Engineering in 1987 and 1991, respectively, both from the Technical University of Valencia (Spain). He 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 applications for GPUs. He has published more than 110 refereed papers in different journals and conferences in these fields.

He 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).