Scalable Directory Organization for Tiled CMP Architectures

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

Abstract

Although directory-based cache coherence protocols are the best choice when designing chip multiprocessor architectures (CMPs) with tens of processor cores on chip, the memory overhead introduced by the directory structure may not scale gracefully with the number of cores. In this work, we show that a directory organization based on duplicating tags, which are distributed among the tiles of a tiled CMP with a fine-grained interleaving, is scalable. That is to say, the size of each directory bank is independent on the number of tiles of the system. Moreover, based on this directory organization we propose and evaluate the implicit replacements mechanism which leads to savings of up to 32% in terms of number of messages in the interconnection network.

Keywords: *Tiled chip multiprocessors, cache coherence, directory organization, scalability, implicit replacements.*

1 Introduction

Tiled CMP architectures [21, 24] have recently emerged as a scalable alternative to current small-scale CMP designs (e.g. the dual-core IBM Power 6 [10]), and future CMPs that will integrate tens of cores on chip will be probably designed as arrays of identical or close-to-identical building blocks (tiles) connected over a switched direct network (see for example the Intel Tera-Scale Computing Project [3, 22]). 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. 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 adds requirements of area and power consumption to the final design, and therefore, could restrict severely its scalability. When the number of cores is large, 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 protocols reduce power consumption compared to broadcast-based protocols by keeping track of the sharers of each block in a directory structure. In a traditional directory organization, each directory entry keeps track of the sharers of the corresponding memory block through a simple bit-vector (one bit per private cache). Since this structure does not scales with the number of cores, many approaches aimed at improving its scalability 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. Another alternative to the bit-vector scheme which keeps the same sharing information is to duplicate the tags of all private caches. This scheme has been recently used in CMPs as Piranha [4] or Niagara 2 [20].

In tiled CMPs, the directory structure is distributed between the L2 cache banks, usually included in the L2 tags' portion. In this way, each tile keeps the sharing information of the blocks mapped to the L2 cache bank that it contains. 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 area. Moreover, a hard to scale directory organization could limit the use of the same tile to CMPs with a small range of tiles.

In this work, we show that a directory organization based on duplicating tags, which are distributed among the tiles of a tiled CMP with a fine-grained interleaving, is scalable. In particular, we show that the size of each directory bank does not depend on the number of tiles. In this organization, each directory entry stores the tag of the block allocated in the corresponding entry of the private cache, a valid bit and an ownership bit. If the ownership bit is set the cache is known to be the owner of the block. In this way, the size of each directory bank is $c * (l_t + 2)$, 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 one entry of some private cache, and vice versa, the directory interleaving must be defined taking the less significant bits of the memory address [9], as we discuss in Section 3.

Secondly, this directory organization allows us to modify the coherence protocol in order to remove extra the messages caused by replacements. We have named this technique as *implicit replacements*. Since each cache entry is associated to a directory entry (the same way too) is not necessary that the requesting tile sends a message to the directory informing about the replacement. The directory knows which block is being replaced when the request for a new block arrives to it. We have found that the implicit replacements technique leads to savings of up to 32% in the number of messages in the interconnection network (12% on average).

The rest of the paper is organized as follows. In Section 2 we present a review of the related work. The directory organization is discussed in Section 3. The proposed cache coherence protocol is described in Section 4. Section 5 introduces the methodology employed in the evaluation. In Section 6 we study the area requirements of our proposal. Section 7 shows the performance results obtained by our proposal. And finally, Section 8 concludes the paper.

2 Related Work

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 ($O(nm)$ when bit-vectors keep track of the sharers of every memory block). 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, and then, we comment the proposals on which our work is based.

Some proposals reduce the width of directory entries by using compressed sharing codes instead of a bit-vector. 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 caches the memory block). Another compressed sharing codes are *tristate* [2] (also called superset scheme), *Gray-tristate* [14] or *binary tree with subtrees* [1]. Other authors propose to have a limited number of pointers per entry, which are chosen for covering the common case [5, 19], and overflow situations are handled by broadcasting invalidation messages.

Other proposals try to reduce the directory height by combining several entries into a single one (*directory entry combining*) [18]. An alternative way is to organize the directory structure as a cache (*sparse directory*) [16, 7], or include this information in the tags of private caches [17], 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 two set of techniques result in extra coherence messages being sent or increased cache miss rates, providing scalability in terms of memory at the expense of performance or power (as a consequence of an increase in network traffic), thus making them not very suitable for tiled CMP architectures.

The idea of having duplicate tags has also been used in distributed shared-memory multiprocessors, for example, by Nanda *et al.* in Everest [15]. 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 bit-vector directory. However, the number of entries in the CCR directory grows linearly with the number of nodes in the system.

Subsequently, other proposals have used a similar scheme for implementing the directory structure in CMPs [4, 6, 20]. In Piranha [4] a directory structure that keeps a duplicate copy of the L1 tags is used to avoid the use of snooping at L1 caches. However, the directory is centralized which is a bottleneck for large-scale systems.

On the other hand, some proposals for tiled CMPs adds bit-vectors to the L2 tags to indicate which L1 caches have copies of the block [8]. This technique saves an extra directory structure when the L1 and L2 caches are inclusive. Again, the bit-vector used to keep track of the sharers does not scale with the number of tiles.

In [9], the directory interleaving is studied to reduce the size of a distributed directory that stores a linked list of pointers to the sharers of each cache block. An interleaving 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 list of pointers has long latency accesses.

Recently, in [13] 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 Directory Organization

In this section we study how to adapt a directory organization based on duplicate tags to scalable tiled CMPs. In the first subsection, we discuss how the directory interleaving affects the scalability of the directory structure. In the second subsection, we describe the structure of the directory and how the sharing information can be obtained from it.

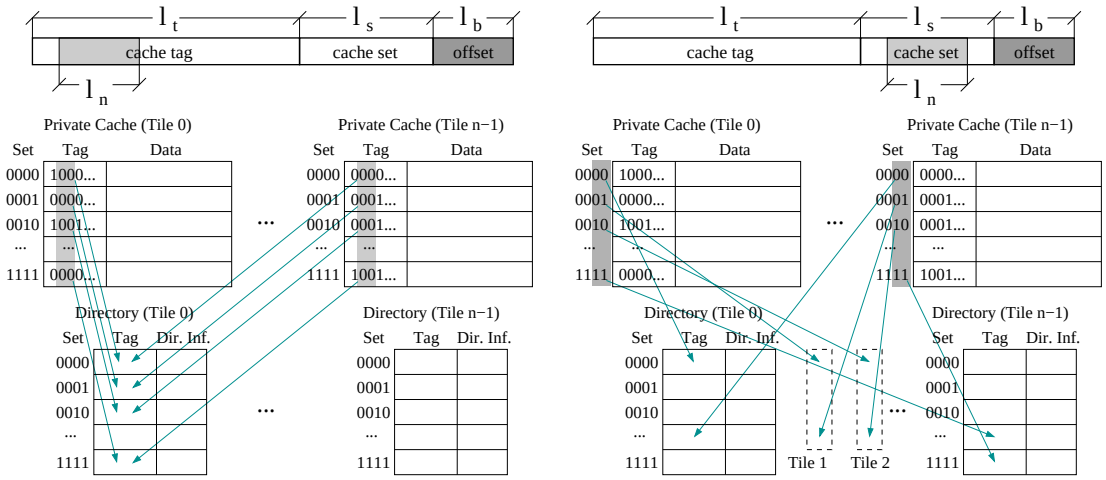


Figure 1. Granularity of directory interleaving and its effect on directory size.

3.1 Directory Interleaving

In tiled CMPs, the shared L2 cache is distributed among all the tiles in the system. In this way, each tile is in charge of handling the requests for a particular region of the total shared cache. This tile is called the *home* tile for the memory blocks that it handles. An important decision when designing the memory hierarchy of tiled CMPs is the granularity of both the shared L2 cache and the directory interleaving. Cache and directory interleaving may be different. However, this policy incurs in extra coherence messages among the different banks of the distributed shared cache thus making the coherence protocol less efficient and more complex. Therefore, it is desirable that both shared L2 cache and directory have the same interleaving.

The shared L2 cache 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). The position of these bits defines the granularity of the interleaving, and as shown in Figure 1, the number of entries that each directory bank must have to be able to keep the sharing information of those cached blocks belonging to it.

In Figure 1 we can observe two alternative ways of distributing the shared L2 cache and their consequences. Looking at the address of a memory block we can distinguish three main fields: the block *offset* (l_b) which represent 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 belong to the cache tag field, the shared L2 cache is split into huge continuous regions (coarse-grained interleaving). Under this configuration, all the blocks stored in the private caches could map to the same directory bank. This situation is shown in Figure 1 (left). 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 $n * s * a$. In particular, each directory must have s sets of $n * a$ ways each one. Therefore, the order required by this structure is $O(n * s * a)$, or $O(n * c)$, where c is the number of entries of each last level private cache.

Otherwise, if the l_n bits belong to the cache set field, the shared L2 cache is split in a great amount of small regions (fine-grained interleaving), as shown in Figure 1 (right). Under this configuration, each entry of each L1 cache maps to only one entry of the directory. Therefore, the number of entries required by each directory will be $s * a$. In particular, each directory bank must have s/n sets of $n * a$ ways each one. The order required by this structure is $O(c)$, which scales with the number of nodes of the system.

Therefore, our organization uses an interleaving where the l_n bits belong to the cache set fields. 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 $n * a$, but this is not the common case. In any case, the order of the entries needed by the directory is $O(max\{s, n\} * a)$, that is to say, the number of entries completely scales for values of s larger than values of n.

3.2 Duplicate Tags

In the previous section we have described how the number of entries of the directory can scale with the number of tiles. However, the size of the entries used to keep the directory information does not scale with the number of tiles (i.e. $O(n)$ for the bit-vector, or $O(p * log_2 n)$ when p pointers are used to locate the cached copies). In this section we propose to store the tag of the block and two bits in each directory entry. The first bit is the valid 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

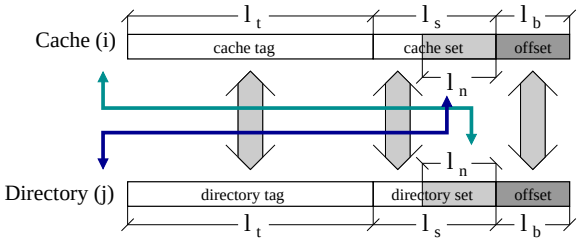


Figure 2. Mapping between cache entries and directory entries.

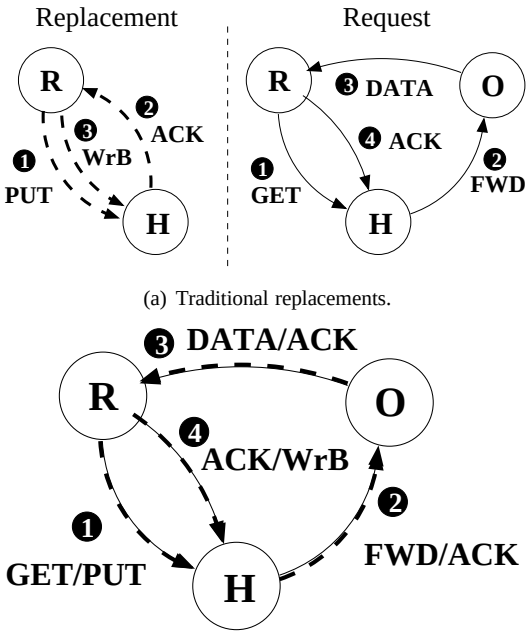
only one cache entry. This bit is used to locate all the copies of the block on a write miss. The second bit is the ownership bit and when it is set the cache entry is known to have the ownership of the block. This bit is used to implement a MOESI protocol.

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 maintains constant as the number of cores of the CMP grows. The total size of each directory bank is $c * (l_t + 2)$.

The associativity of the directory can be reduced from $n * a$ 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. Figure 2 shows how cache entries are mapped to directory entries, to achieve this small associativity. We can see that l_n bits of the directory set are used to identify the tile that holds the copy in its private L1 cache, and l_n bits of the cache set are used to identify the home directory. In this way, each directory bank has the same number of sets and ways than the L1 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.

Considering this mapping, all the coherence information for a particular block can be found in the following way. To know whether a block is stored in a particular cache it is necessary to find the tag in the directory set whose identifier is obtained by changing the l_n bits that identify the home directory with the l_n bits that identify the tile which contains that cache. If the tag is found and the valid bit is set, the block is held in that cache. If the ownership bit is set, that cache is the owner of the block. By searching this information in the corresponding n directory entries the complete directory information is obtained. If the l_n bits are the less significant, the search can be performed by reading n consecutive sets of the directory bank.

Updating the directory information only requires modifying few bits. On a write miss, the n corresponding valid bits are unset and the directory entry for the new owner is set with the tag of the block and both the valid and the ownership bits are set. Adding a new sharer only requires writing the tag of the block in the corresponding directory entry and setting the valid bit.



(a) Traditional replacements.

(b) Implicit replacements.

Figure 3. Differences between the proposed coherence protocol and a traditional coherence protocol.

4 Implicit replacements

The proposed directory organization allows us to modify the coherence protocol 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, as shown in Figure 3.

There are two main factors that allows the presented directory organization to support implicit replacements. Firstly, due to the fine-grained directory interleaving we ensure that the evicted block and the requested block map to the same home, and therefore, the same directory bank. If a coarse-grained interleaving was chosen these blocks could map to different directory banks. 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 both the requested and the evicted block and it is not necessary to attach the address of the evicted block to the request messages. Note that the size of coherence messages does not change. It is only necessary to add a field indicating the way within the set of the requested block (2 bits in our case).

In Figure 3(a), we can see how a replacement is usually performed. When a block must be stored in cache and the corresponding set is full, an old block must be evicted. In current directory protocols evictions of shared blocks are usually performed transparently without informing the directory. Later, we will explain why when the directory is or-

ganized with duplicate tags, this kind of replacements must inform the directory. On the other hand, evictions of private or owned blocks must store the dirty data in the next cache level. For simplicity these writebacks could be performed in a tree-hop transaction as illustrated in Figure 3(a) (left). First, the cache asks the home tile permission to *writeback* the block (PUT). Then the home tile confirms the transaction (ACK), and finally the block is sent to the next cache level (WrB). Figure 3(a) (right) shows a cache-to-cache transfer miss. Requests (GET) are sent to the home tile to get the directory information, and then are forwarded (FWD) to the owner cache where the data is provided (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 (ACK).

Figure 3(b) shows how the implicit replacements are performed along with the requests that cause them. On each cache miss a MSHR entry (Miss Status Hold Register) is allocated with the information about the request. Moreover, another MSHR entry must be allocated for the evicted block (if any). The former entry needs to keep a pointer to the later one. Moreover, the particular way within the set where the new block will be stored is specified in the each coherence message. When the GET/PUT message reaches the home tile, another two MSHR entries must be allocated (as usually), one of them pointing to the other. Finally, when the data arrives to the requesting cache (DATA/ACK) both MSHRs are deallocated and the writeback is performed (ACK/WrB), thus allowing the directory processing the subsequent requests for both blocks.

Finally, it is necessary to inform the directory about evictions of shared blocks in some protocols like Piranha [4] which uses duplicate tags. This is because before adding a new entry to the directory an old entry must be deallocated if the number of entries of the directory is the same than the number of entries of the caches. This can be avoided by taking into account the way within the set that the block is going to use.

5 Simulation Environment

We evaluate our proposal with full-system simulation using Virtutech Simics [11] extended with Multifacet GEMS [12]. GEMS provides a detailed memory system timing model which accounts for all protocol messages and state transitions. We simulate a 16-tiled CMP system with one level of private cache and a logically shared and physically distributed L2 cache. Table 1 shows the values of the main parameters of this system.

We have implemented all the coherence protocols evaluated in Section 7. These implementations have been exhaustively checked using a tester program provided by GEMS that checks all race conditions to raise any incoherence. The *Bit-Vector* protocol organizes the directory as a tagged cache with a bit-vector for each entry. This protocols has silent evictions of shared blocks. The rest of protocols organize the directory as duplicate tags. In *DupTag-Sh*, evictions of shared blocks incurs in tree-hop transactions. In *DupTag*, these evictions are silent as in the base directory protocol. Finally, *DupTag-Implicit* is the protocol that implements the implicit replacements mechanism.

Table 1. System parameters.

16-tiled CMP	
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I & D caches	32KB, 4-way
L1 cache hit time	4 cycles
Shared unified L2 cache	8MB (512KB/tile), 4-way
L2 cache hit time	6 + 9 cycles (tag + data)
Memory access time	160 cycles
Network topology	4x4 Mesh
On-chip link latency (one hop)	2 cycles
Off-chip link latency (one hop)	20 cycles

The eight scientific applications used in our simulations cover a variety of computation and communication patterns. Barnes (8192 bodies, 4 time steps), FFT (256K complex doubles), Ocean (258x258 ocean), Radix (1M keys, 1024 radix), Raytrace (teapot), Volrend (head-scaleddown4) and Water-NSQ (512 molecules, 4 time steps) are from the SPLASH-2 benchmark suite [23]. Unstructured (Mesh.2K, 5 time steps) is a computational fluid dynamics application. 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 Directory Memory Overhead

In this section we study the directory memory overhead of our proposed organization compared to some of the schemes described in the related work. Figure 4 shows this overhead as a function of the number of tiles in the system. The directory organizations shown in the graph are *Bit-Vector L2 Inclusive*, *Duplicate Tags*, *Bit-Vector + FGI*, *Coarse Vector (K=4) + FGI*, *Limited pointers (3) + FGI*, and finally our proposal (*Duplicate Tags + FGI*). The characteristics of all these schemes are described below. The overhead of the directory structure has been calculated for the values shown in Table 1.

In the graph, we plot results for several directory organizations. *Bit-Vector L2 Inclusive* is currently used in the proposed tiled CMPs in which the L1 and the L2 are inclusive (the L2 contains all blocks held in the L1s). The directory is stored in the tags' part of the L2 cache, thus removing the need of tags for the directory structure. Apart from forcing inclusion and using a bit-vector sharing code (it could use another compressed sharing code), the main drawback of this scheme is that it needs the same number of entries than the L2 cache.

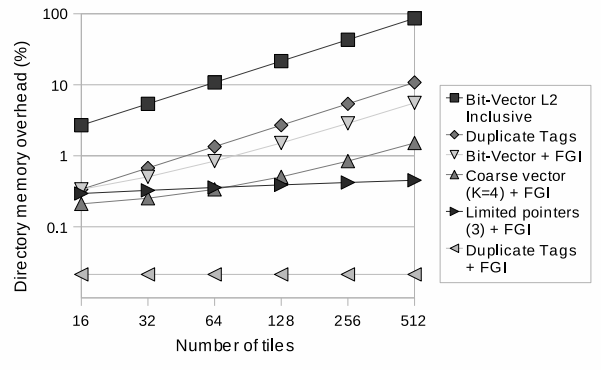


Figure 4. Directory memory overhead as a function of the number of tiles.

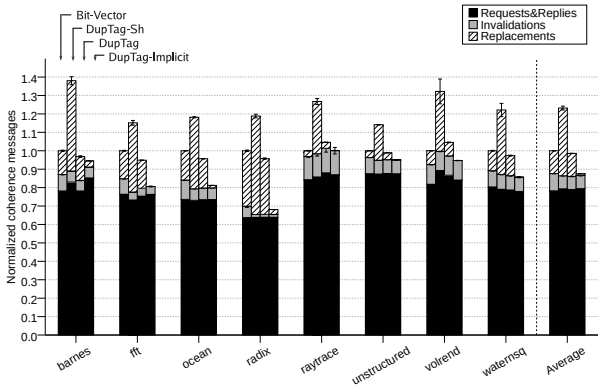


Figure 5. Reductions in the number of coherence messages.

The problem of using *Duplicate Tags* with coarse-grained interleaving (or a dynamic mapping) is that the number of entries needed for each directory bank to allocate the information of all the cached blocks does not scale, as explained in Section 3.1.

When fine-grained interleaving (FGI) is considered, the number of entries of the directory bank is reduced to the number of L1 cache entries. As a consequence, the *Bit-Vector + FGI* organization requires less storage than the previous schemes, but still remains non-scalable. In *Coarse Vector (K=4) + FGI* 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. Again the area of the directory structure is reduced, but it does not scale. In *Limited pointers (3) + FGI* only three pointers are used to identify the caches that share each memory block. When the number of sharers is higher than three, writes are performed by broadcasting invalidation messages (a broadcast bit is also required per entry). This organization scales as $O(3 * log_2 n)$. However, differently from the proposed organization, compressed sharing codes incur in extra coherence messages since they do not store precise information about all the caches that hold blocks.

Finally, we can see that by combining fine-grained interleaving with duplicate tags (*Duplicate Tags + FGI*) we can achieve a completely scalable directory organization which keeps the same information than a bit-vector directory.

7 Evaluation Results

In this section, we evaluate the results in terms of number of coherence messages removed. Regarding performance (execution time), all the protocols achieve similar results. In particular, our protocol is able to remove all coherence messages caused by L1 replacements, without loosing any performance. In Figure 5 we can observe the percentage of messages saved by our proposal.

The number of coherence messages of each one of the evaluated protocols has been normalized with respect to a traditional directory-based protocol that uses a bit-vector sharing code (*Bit-Vector* protocol). Compared to this protocol, the implicit replacements mechanism reduces the number of coherence messages up to 32% for Radix (12% on average). Moreover, if we consider the *DupTag-Sh* protocol our proposal can save 29% of coherence messages on average. Finally, the *DupTag* protocol slightly reduces the number of coherence messages compared to *Bit-Vector* mainly since the number of invalidations is reduced. This is because when a new tag is allocated in the duplicate tag directory, the old tag is removed thus implicitly informing to the directory of the evictions of shared blocks, and therefore most of the invalidations that find the block evicted from cache in *Bit-Vector* are removed in *DupTag*.

8 Conclusions

In this work, we show that a directory organization based on duplicating tags, which are distributed among the tiles of a tiled CMP following a fine-grained interleaving is scalable. We show that 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 $c * (l_t + 2)$, where c is the number of entries of the private L1 cache if the cache hierarchy is inclusive or the aggregate number of entries of private L1 caches if the cache hierarchy is non-inclusive, and l_t is the size of the tag field.

Since the structure of each directory bank does not change with the number of tiles, the same building block can be used for systems with different number of tiles, thus making the design of large-scale CMPs possible.

We have 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 up to 32% in the number of coherence messages (12% on average), and consequently reducing power consumption. To achieve this, the block must be mapped taking into consideration the way within the set in which it is stored (the same way in the private cache as in the directory).

Finally, as our protocol knows the way where the block must be stored, more power consumption could be saved for associative caches if accesses to other ways within the set are removed.

Acknowledgements

This work has been jointly supported by Spanish MEC under grant "TIN2006-15516-C04-03", European Comission FEDER funds under grant "Consolider Ingenio-2010 CSD2006-00046" and European Comission funds under Network of Excellence HiPEAC. A. 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. A Two-Level Directory Architecture for Highly Scalable cc-NUMA Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 16(1):67–79, Jan. 2005.

[2] A. Agarwal, R. Simoni, J. L. Hennessy, and M. Horowitz. An Evaluation of Directory Schemes for Cache Coherence. In *15th Int'l. Symp. on Computer Architecture (ISCA'88)*, pages 280–289, May 1988.

[3] M. Azimi, N. Cherukuri, D. N. Jayasimha, A. Kumar, P. Kundu, S. Park, I. Schoinas, and A. S. Vaidya. Integration challenges and tradeoffs for tera-scale architectures. *Intel Technology Journal*, 11(3):173–184, Aug. 2007.

[4] 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 Int'l Symp. on Computer Architecture (ISCA'00)*, pages 12–14, June 2000.

[5] D. Chaiken, J. Kubiatowicz, and A. Agarwal. LimitLESS Directories: A Scalable Cache Coherence Scheme. In *4th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, pages 224–234, Apr. 1991.

[6] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *33rd Int'l Symp. on Computer Architecture (ISCA'06)*, pages 264–276, June 2006.

[7] A. Gupta, W.-D. Weber, and T. C. Mowry. Reducing memory traffic requirements for scalable directory-based cache coherence schemes. In *Int'l Conference on Parallel Processing (ICPP'90)*, pages 312–321, Aug. 1990.

[8] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA substrate for flexible CMP cache sharing. In *19th Int'l Conference on Supercomputing (ICS'05)*, pages 31–40, June 2005.

[9] J. Kong, P.-C. Yew, and G. Lee. Minimizing the directory size for large-scale shared-memory multiprocessors. *IEICE - Transactions on Information and Systems*, E88-D(11):2533–2543, Nov. 2005.

[10] 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), Nov. 2007.

[11] 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.

[12] 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.

[13] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In *34th Int'l Symp. on Computer Architecture (ISCA'07)*, pages 46–56, June 2007.

[14] S. Mukherjee and M. D. Hill. An Evaluation of Directory Protocols for Medium-Scale Shared-Memory Multiprocessors. In *8th Int'l Conference on Supercomputing (ICS'94)*, pages 64–74, July 1994.

[15] A. K. Nanda, A.-T. Nguyen, M. M. Michael, and D. J. Joseph. High-Throughout Coherence Control and Hardware Messaging in Everest. *IBM Journal of Research and Development*, 45(2):229–244, Mar. 2001.

[16] B. O'Krafka and A. Newton. An Empirical Evaluation of Two Memory-Efficient Directory Methods. In *17th Int. Symp. on Computer Architecture (ISCA'90)*, pages 138–147. IEEE/ACM, June 1990.

[17] A. Ros, M. E. Acacio, and J. M. García. A novel lightweight directory architecture for scalable shared-memory multiprocessors. In *11th Int'l Euro-Par Conference*, volume 3648, pages 582–591, Aug. 2005.

[18] R. Simoni. *Cache Coherence Directories for Scalable Multiprocessors*. PhD thesis, Stanford University, 1992.

[19] R. Simoni and M. Horowitz. Dynamic Pointer Allocation for Scalable Cache Coherence Directories. In *Int'l Symp. on Shared Memory Multiprocessing*, pages 72–81, Apr. 2001.

[20] Sun Microsystems, Inc., Santa Clara, CA 95054. *OpenSPARCTM T2 System-On-Chip (SOC) Microarchitecture Specification*, Dec. 2007.

[21] M. B. Taylor, J. Kim, and J. Miller, et al. The raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, May 2002.

[22] S. Vangal, J. Howard, and G. Ruhl, et al. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *IEEE Int'l Solid-State Circuits Conference (ISSCC)*, Feb. 2007.

[23] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *22nd Int'l Symp. on Computer Architecture (ISCA'95)*, pages 24–36, June 1995.

[24] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *32nd Int'l Symp. on Computer Architecture (ISCA'05)*, pages 336–345, June 2005.