We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



6

Cache Coherence Protocols for Many-Core CMPs

Alberto Ros, Manuel E. Acacio and José M. García
Universidad de Murcia
Spain

IntechOpen

1. Introduction

Multi-core architectures have emerged as the best alternative to take advantage of the increasing number of transistors currently offered in a single die. For example, the dual-core IBM Power6 (Le et al., 2007) and the eight-core Sun UltraSPARC T2 (Shah et al., 2007) have a relatively small number of cores, which are typically connected through a shared medium, i.e., a bus or a crossbar. However, CMP architectures that integrate tens of processor cores (usually known as many-core CMPs) are expected for the near future, after Intel recently unveiled the 80-core Polaris prototype (Azimi et al., 2007). Since the area required by a shared interconnect becomes impractical as the number of cores grows (Kumar et al., 2005), it seems that the processing cores of future CMPs will be connected by means of unordered point-to-point networks. Hence, tiled CMP architectures (Taylor et al., 2002; Zhang & Asanovic, 2005), which are designed as arrays of replicated tiles connected over a point-to-point network, have arisen as a scalable alternative to current small-scale CMP designs and they will help in keeping complexity manageable.

On the other hand, most CMP systems provide programmers with the intuitive shared-memory model, which requires efficient support for cache coherence. Although a great deal of attention was devoted to scalable cache coherence protocols in the last decades in the context of shared-memory multiprocessors, the technological parameters and constraints entailed by many-core CMPs demand new solutions to the cache coherence problem (Bosschere et al., 2007; Azimi et al., 2007).

In this chapter, we focus on three main design goals for cache coherence protocols aimed at being employed in many-core CMPs: performance, on-chip network traffic, and area requirements. For example, area constraints prevent from using an ordered interconnection network and, consequently, the popular snooping-based cache coherence protocol. Additionally, on-chip network traffic has been previously reported to constitute a significant fraction (approaching 50% in some cases) of the overall chip power (Wang et al., 2003; Magen et al., 2004).

We will firstly describe two cache coherence protocols which are used in current commodity chip multiprocessors, discussing their scalability constraints and bottlenecks: *Hammer*, implemented in the AMD Opteron^{TM}(Ahmed et al., 2002), and *Directory* used in Piranha (Barroso et al., 2000). *Hammer* avoids keeping coherence information at the cost of broadcasting requests to all cores. Although it is very efficient in terms of area requirements, it generates a prohibitive amount of network traffic, which translates into excessive power consumption. On the other hand, *Directory* reduces network traffic compared to *Hammer* by storing in a directory structure precise information about the private caches holding memory blocks. Unfor-

tunately, the storage overhead that directories entail could become prohibitive for many-core CMPs (Azimi et al., 2007). Since neither the network traffic generated by *Hammer* nor the extra area required by *Directory* scale with the number of cores, a great deal of attention was paid in the past to address this traffic-area trade-off (Agarwal et al., 1988; Gupta et al., 1990; Chaiken et al., 1991; Mukherjee & Hill, 1994; Acacio et al., 2001).

On the other hand, these traditional cache coherence protocols introduce indirection in the critical path of cache misses. In both protocols, the ordering point for the requests to the same memory block is the *home* node or tile. Therefore, all cache misses must reach this ordering point before any coherence actions can be performed, a fact that adds extra latency to cache misses. Recently, Token-CMP (Martin et al., 2003) and DiCo-CMP (Ros et al., 2008a) protocols have been proposed to deal with the indirection problem. These indirection-aware protocols avoid the access to the home node through alternative serialization mechanisms. In this way, they reduce the latency of cache misses compared to *Hammer* and *Directory*, which translates into performance improvements. Although Token-CMP entails low memory overhead, it is based on broadcasting requests to all nodes, which is clearly non-scalable. Otherwise, DiCo-CMP sends requests to just one node, but it adds a full-map sharing code that keeps track of sharers to each cache entry, which does not scale with the number of cores.

In this chapter, we discuss both protocols that are used nowadays, such as *Hammer* and *Directory*, and these two novel indirection-aware protocols (Token-CMP and DiCo-CMP). We also study how they can scale up to a greater number of cores. In particular, we perform this study by considering direct coherence (DiCo) protocols and, therefore we first describe this kind of protocols in detail. Finally, we compare all the described protocols in terms of performance, network traffic and area requirements, thus performing a detailed evaluation of a wide range of cache coherence protocols for many-core CMPs in a common framework.

The rest of the chapter is organized as follows. Section 2 introduces tiled CMP architectures. Section 3 discusses and presents a classification of some cache coherence protocols that could be used in tiled CMPs. Section 4 offers a detailed description of direct coherence protocols, and Section 5 discusses several implementations that differ in the amount of coherence information that they keep. Section 6 focuses on the evaluation methodology. Section 7 shows and analyses performance results. In Section 8, we present a review of the related work and, finally, Section 9 concludes the chapter.

2. Tiled CMPs

Tiled CMP architectures are designed as arrays of identical or close-to-identical building blocks known as *tiles*. In these architectures, each tile is comprised of a processing core, one or several levels of caches, and a network interface or router that connects all tiles through a tightly integrated and lightweight point-to-point interconnection network (e.g., a two-dimensional mesh). Differently from shared networks, point-to-point interconnects are suitable for many-core CMPs because their peak bandwidth and area overhead scale with the number of cores. Tiled CMPs can easily support families of products with varying number of tiles, including the option of connecting multiple separately tested and speed-binned dies within a single package. Therefore, it seems that they will be the choice for future many-core CMPs.

In this chapter, we assume a tiled CMP with two levels of on-chip caches, as shown in Figure 1. The first one (L1 cache) is private to its local processing core. In contrast, the second one (L2 cache) is logically shared (but physically distributed) among the processing cores. Therefore, each cache block maps to a particular L2 cache bank, which is called the *home* tile for that block.

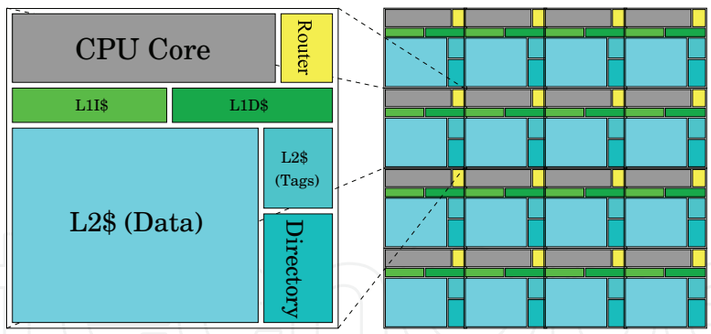


Fig. 1. Organization of a tile (left) and a 4×4 tiled CMP (right).

The home bank of each block is commonly obtained from its address bits. Particularly, the bits usually chosen for the mapping to a particular bank are the less significant ones without considering the block offset (Huh et al., 2005; Zhang & Asanovic, 2005; Shah et al., 2007).
Since, wire delay of future CMPs will cause cross-chip communications to reach tens of cycles (Agarwal et al., 2000; Ho et al., 2001), the access latency to a multibanked shared cache will be dominated by the delay to reach each particular cache bank rather than the time spent accessing the bank itself. In this way, the access latency to the shared cache can be drastically different depending on the cache bank where the requested block maps. The resulting cache design is what is known as non-uniform cache architecture (NUCA) (Kim et al., 2002).
The main downside of a NUCA organization is the long cache access latency (on average), since it depends on the bank wherein the block is allocated, especially when home banks are assigned by taking some fixed bits from the block address. Since, in this case, the distribution of the blocks is performed in a round-robin fashion without considering the distance from the requesting cores to the home banks, it is more important to avoid the indirection to the home tile, because for most misses the requested block could map to a remote cache bank.

3. Cache coherence protocols for tiled CMPs

As introduced at the beginning of this chapter, traditional snooping-based protocols require an ordered interconnect to keep cache coherence, but such interconnects do not scale in terms of area requirements. This section describes and classifies the four cache coherence protocols considered in this chapter as potential candidates to be employed in tiled CMPs (i.e., with unordered networks): *Hammer*, *Directory*, *Token*, and *DiCo*. In particular, we classify these cache coherence protocols into *traditional* protocols, in which cache misses suffer from indirection, and *indirection-aware* protocols, which try to avoid the indirection problem. For each type, we also differentiate between area-demanding and traffic-intensive protocols.
We discuss the implementation of these cache coherence protocols for a tiled CMP in which each tile includes a private L1 cache and a slice of the shared L2 cache, as described in the previous section. In this way, cache coherence is maintained among data stored in the L1 caches. We also assume that private caches use MOESI states, and that L1 and L2 caches are non-inclusive.

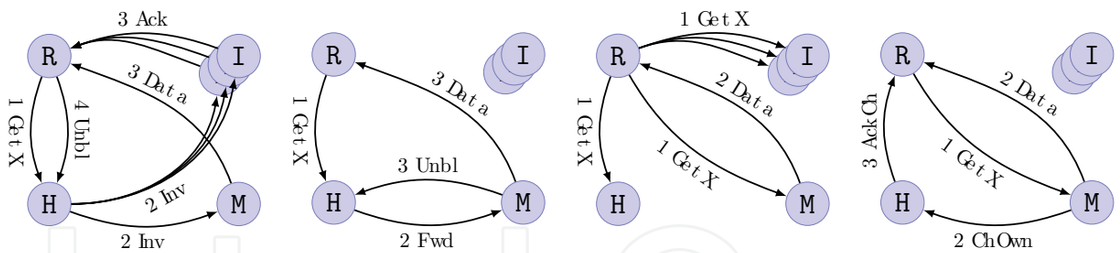
3.1 Traditional protocols

In traditional protocols, the requests issued by several cores to the same block are serialized through the home tile, which enforces cache coherence. Therefore, all requests must be sent

to the home tile before any coherence action can be performed. Then, requests are forwarded to the corresponding tiles according to the coherence information (if needed). All processors that receive a forwarded request answer to the requesting core by sending either an acknowledgment (and invalidating the block in case of write misses) or the requested data block. The requesting core can access the block when it receives all the acknowledgment and data messages. The access to the home tile introduces indirection, which causes that most cache misses take three hops in the critical path.
Examples of these traditional protocols are *Hammer* and *Directory*. As commented in the introduction, *Hammer* has the drawback of generating a considerable amount of network traffic. On the other hand, directory protocols that use a precise sharing code to keep track of cached blocks introduce an area overhead that does not scale with the number of cores.

3.1.1 Hammer-CMP

Hammer (Owner et al., 2006) is the cache coherence protocol used by AMD in their Opteron systems (Ahmed et al., 2002). Like snooping-based protocols, *Hammer* does not store any coherence information about the blocks held in the private caches and, therefore, it relies on broadcasting requests to all tiles to solve cache misses. Its key advantage with respect to snooping-based protocols is that it targets systems that use unordered point-to-point interconnection networks. In contrast, the ordering point in this protocol is the home tile, a fact that introduces indirection on every cache miss.
We have implemented a version of the AMD's Hammer protocol for tiled CMPs that we call *Hammer-CMP*. As an optimization, our implementation adds a small structure to each home tile. This structure stores a copy of the tags for the blocks that are held in the private L1 caches. In this way, cache miss latencies are reduced by avoiding off-chip accesses when the block can be obtained on-chip. Moreover, the additional structure has small size and it does not increase with the number of cores.
On every cache miss, *Hammer-CMP* sends a request to the home tile. If the memory block is present on chip (this information is given by the structure that we add to each home tile), the request is forwarded to the rest of tiles to obtain the requested block, and to eliminate potential copies of the block in case of a write miss. Otherwise, the block is requested to the memory controller.
All tiles answer to the forwarded request by sending either an acknowledgment or the data message to the requesting core. The requesting core needs to wait until it receives the response from each other tile. When the requester receives all the responses, it sends an unblock message to the home tile. This message notifies the home tile about the fact that the miss has been satisfied. In this way, if there is another request for the same block waiting at the home tile, it can be processed. This unblock message prevents the occurrence of race conditions.
Figure 2(a)shows an example of how *Hammer-CMP* solves a cache-to-cache transfer miss. The requesting core (*R*) sends a write request (*1 GetX*) to the home tile (*H*). Then, invalidation messages (*2 Inv*) are sent to all other tiles. The tile with the ownership of the block (*M*) responds with the data block (*3 Data*). The other tiles that do not hold a copy of the block (*I*) respond with acknowledgment messages (*3 Ack*). When the requester receives all the responses, it sends the unblock message (*4 Unbl*) to the home tile. First, we can see that this protocol requires three hops in the critical path before the requested data block is obtained. Second, broadcasting invalidation messages increases considerably the traffic injected into the interconnection network and, therefore, its power consumption.



(a) Hammer-CMP. (b) Directory-CMP. (c) Token-CMP. (d) DiCo-CMP.

Fig. 2. A cache-to-cache transfer miss in each one of the described protocols.

3.1.2 Directory-CMP

The directory-based protocol that we have implemented for CMPs (*Directory-CMP*) is similar to the intra-chip coherence protocol used in Piranha (Barroso et al., 2000). In particular, the directory information consists in a full-map (or bit-vector) sharing code, that is employed for keeping track of the sharers. This sharing code allows the protocol to send invalidation messages just to the caches currently sharing the block, thus removing unnecessary coherence messages. In addition, directory-based protocols that implement MOESI states add an *owner* field that identifies the owner tile to the directory information of each block. The owner field allows the protocols to detect the tile that must provide the block in case of several sharers. In this way, requests are only forwarded to that tile. The use of directory information allows the protocol to reduce considerably network traffic when compared to *Hammer-CMP*.

In the implemented directory protocol, on every cache miss, the core that causes the miss sends the request only to the home tile, which is the serialization point for all requests issued for the same block. Each home tile includes an on-chip directory cache that stores the sharing and owner information for the blocks that it manages. This cache is used for the blocks that do not hold a copy in the shared cache. In addition, the tags' part of the shared cache also include a field for storing the sharing information for those blocks that have a valid entry in that cache. Once the home tile decides to process the request, it accesses the directory and it performs the appropriate coherence actions. These coherence actions include forwarding the request to the owner tile, and invalidating all copies of the block in case of write misses.

When a tile receives a forwarding request it provides the data to the requester if it is already available or, in other case, the request must wait until the data is available. Like in *Hammer-CMP*, all tiles must respond to the invalidation messages with an acknowledgment message to the requester. Since acknowledgment messages are collected by the requester, it is necessary to inform the requester about the number of acknowledgments that it has to receive before accessing the requested data block. In our particular implementation, this information is sent from the home tile, which knows the number of invalidation messages issued, to the requester along with the forwarding and data messages. When the requester receives all acknowledgments and the data block, data can be accessed.

Figure 2(b)shows an example of how *Directory-CMP* solves a cache-to-cache transfer miss. The request is sent to the home tile, where the directory information is stored (*1 GetX*). Then, the home tile forwards the request to the provider of the block, which is obtained from the directory information (*2 Fwd*). The provider sends the unblock message to the home tile to allow subsequent requests to be processed (*3 Unbl*) and it also sends the data to the requester (*3 Data*). When the data block arrives to the requester, the miss is considered solved. As we can see, although this protocol introduces indirection to solve cache misses (three hops in the

critical path of the miss), few coherence messages are required to solve them, which finally translates into savings in network traffic and less power consumption. This characteristic allows the directory protocol to scale up to a greater number of cores than *Hammer-CMP*.

3.2 Indirection-aware protocols

Recently, new cache coherence protocols have been proposed to avoid the indirection problem of traditional protocols. *Token-CMP* avoids indirection by broadcasting requests to all tiles and maintains coherence through a token counting mechanism. Token-CMP only cares about requests ordering in case of race conditions. In those cases, a persistent requests mechanism is responsible for ordering the different requests. Although the area required to store the tokens of each block is reasonable, network requirements are prohibitive for may-core CMPs.
On the other hand, in *DiCo-CMP* the ordering point is the tile that provides the block in a cache miss and indirection is avoided by directly sending the requests to that tile. *DiCo-CMP* keeps traffic low by sending requests to only one tile. However, coherence information used in its original implementation (Ros et al., 2008a) include bit-vector sharing codes, which are not scalable in terms of area requirements.

3.2.1 Token-CMP

Token coherence (Martin et al., 2003) is a framework for designing coherence protocols whose main asset is that it decouples the correctness substrate from several different performance policies. Token coherence protocols can avoid both the need of a totally ordered network and the introduction of additional indirection caused by the access to the home tile in the common case of cache-to-cache transfers. Token coherence protocols keep cache coherence by assigning T tokens to every memory block, where one of them is the owner token. Then, a processing core can read a block only if it holds at least one token for that block and has valid data. On the other hand, a processing core can write a block only if it holds all T tokens for that block and has valid data. Token coherence avoids starvation by issuing a persistent request when a core detects potential starvation.
In this chapter, we use *Token-CMP* (Marty et al., 2005) in our simulations. *Token-CMP* is a performance policy aimed at achieving low-latency cache-to-cache transfer misses. It targets CMP systems, and uses a distributed arbitration scheme for persistent requests, which are issued after a single retry to optimize the access to contended blocks.
Particularly, on every cache miss, the requesting core broadcasts requests to all other tiles. In case of a write miss, they have to answer with all tokens that they have. The data block is sent along with the owner token. When the requester receives all tokens the block can be accessed. On the other hand, just one token is required upon a read miss. The request is broadcast to all other tiles, and only those that have more than one token (commonly the one that has the owner token) answer with a token and a copy of the requested block.
Figure 2(c)shows an example of how *Token-CMP* solves a cache-to-cache transfer miss. Requests are broadcast to all tiles (*1 GetX*). The only tile with tokens for that block is *M*, which responds by sending the data and all the tokens (*2 Data*). We can see that this protocol avoids indirection since only two hops are introduced in the critical path of cache misses. However, as happens in *Hammer-CMP*, this protocol also has the drawback of broadcasting requests to all tiles on every cache miss, which results in high network traffic and, consequently, power consumption in the interconnect.

	Traditional	Indirection-aware
Traffic-intensive	Hammer-CMP	Token-CMP
Area-demanding	Directory-CMP	DiCo-CMP

Table 1. Summary of cache coherence protocols.

3.2.2 DiCo-CMP

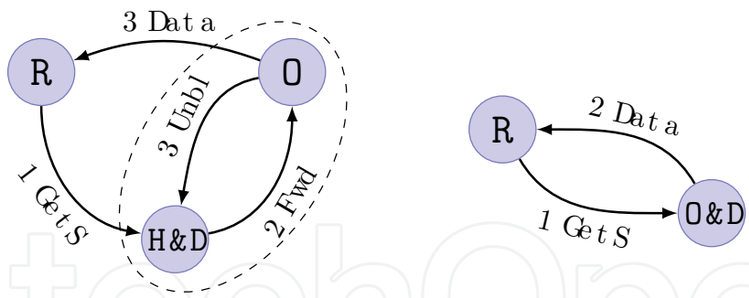
Direct coherence protocols where proposed both to avoid the indirection problem of traditional directory-based protocols and to reduce the traffic requirements of token coherence protocols. In direct coherence, the ordering point for the requests to a particular memory block is the current owner tile of the requested block. In this way, the tile that must provide the block in case of a cache miss is the one that keeps coherence for that block. Indirection is avoided by directly sending requests to the corresponding owner tile instead of to the home tile. In this work we evaluate *DiCo-CMP* (Ros et al., 2008a), an implementation of direct coherence for CMPs. Particularly, we implement the *Base* policy presented in that paper because it is the policy that incurs in less area and traffic requirements.
Figure 2(d)shows an example of how *DiCo-CMP* solves a cache-to-cache transfer miss. The request is directly sent to the tile that has the ownership of the requested block (*1 GetX*). This tile responds by sending the data to the requesting core (*2 Data*), thus requiring just two hops in the critical path of cache misses. Out of the critical path of the miss, the owner tile informs the home tile about the change of ownership (*2 ChOwn*). Then, the home tile acknowledges the change of ownership (*3 AckCh*) allowing to move again the ownership of the block (if requested). Direct coherence protocols are explained in detail in next section. The main drawback of this protocol is that it adds a sharing code to every cache entry, which could result in high area requirements.

3.3 Summary

Table 1 summarizes the protocols described before. This table focuses on the three main metrics evaluated throughout this chapter. The first one is the applications' execution time, which can be affected by the indirection to the home tile. The second one is the network traffic, which impacts power consumption. The third one is the area requirements, which can severely limit the scalability of the CMP. *Hammer-CMP* and *Token-CMP* are based on broadcasting requests on every cache miss. Although the storage required to keep coherence in these protocols is small, they generate a prohibitive amount of network traffic. On the other hand, *Directory-CMP* and *DiCo-CMP* achieve more efficient utilization of the interconnection network at the cost of increasing storage requirements compared to *Hammer-CMP* and *Token-CMP*. Finally, the key advantage of *Token-CMP* and *DiCo-CMP* is that they avoid the indirection problem for most cache misses, thus reducing the execution time compared to traditional protocols.

4. Direct coherence protocols

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



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

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

4.1 Direct coherence basis

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

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

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

- *Order requests*: Cache coherence maintenance requires to serialize the requests issued by different cores to the same block. In snooping-based cache coherence protocols, the requests are ordered by the shared interconnection network. However, since tiled CMP architectures implement an unordered network, this serialization of the requests must be carried out by another component. Directory protocols assign this task to the home tile of each memory block. On the other hand, this task is performed by the owner tile in direct coherence protocols.
- *Keep coherence information*: Coherence information is used to track blocks stored in private caches. In protocols that include the *O* state, like MOESI protocols, coherence information also identifies the owner tile. In particular, *sharing information* is used to invalidate all cached blocks on write misses, while *owner information* is used to know

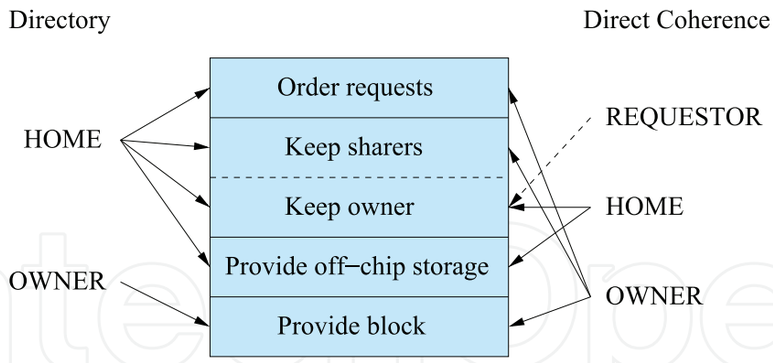
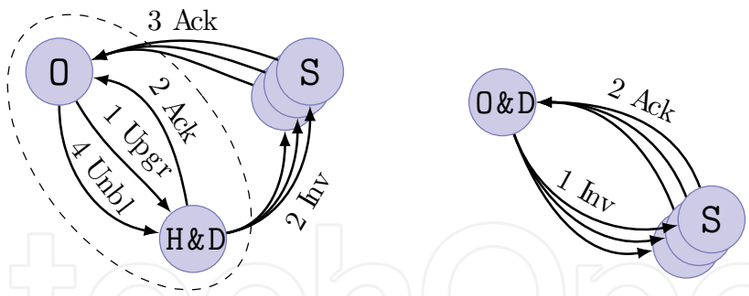


Fig. 4. Tasks performed in cache coherence protocols.

the identity of the provider of the block on every miss. Directory protocols store coherence information at the home tile, where cache coherence is maintained. Instead, direct coherence requires that sharing information be stored in the owner tile for keeping coherence there, while owner information is stored in two different components. First, the requesting cores need to know the owner tile to send the requests to it. Processors can easily keep the identity of the owner tile, e.g., by recording the last core that invalidated their copy. However, this information can become stale and, therefore, it is only used for avoiding indirection (dashed arrow in Figure 4). Then, the responsible for tracking the up-to-date identity of the owner tile is the home tile which must be notified on every ownership change.

- *Provide the data block*: If the valid copy of the block resides on chip, data is always provided by the owner tile, since it always holds a valid copy. The owner of a block is either a tile holding the block in the exclusive or the modified state, the last core that wrote the block when there are multiple sharers, or the the L2 cache bank within the home tile in case of an eviction of the owner block from some L1 cache.
- *Provide off-chip storage*: When the valid copy of a requested block is not stored on chip, an off-chip access is required to obtain the block. Both in directory and direct coherence protocols the home tile is responsible for detecting that the owner copy of the block is not stored on chip. It is also responsible for sending the off-chip request and receiving the data block.

Another example of the advantages of direct coherence is shown in Figure 5. This diagram represents an upgrade that takes place in a tile whose L1 cache holds the block in the owned state, which happens frequently in common applications (e.g., for the producer-consumer pattern). In a directory protocol, upgrades are solved by sending the request to the home tile (*1 Upgr*), which replies with the number of acknowledgements that must be received before the block can be modified (*2 Ack*), and sends invalidation messages to all sharers (*2 Inv*). Sharers confirm their invalidation to the requester (*3 Ack*). Once all the acknowledgements have been received by the requester, the block can be modified and the directory is unblocked (*4 Unbl*). In contrast, in *DiCo-CMP* only invalidation messages (*1 Inv*) and acknowledgements (*2 Ack*) are required because the directory information is stored along with the data block, thereby solving the miss with just two hops in the critical path.
Additionally, by keeping together the owner block and the directory information, control messages between them are not necessary, thus saving some network traffic (two messages in Fig-



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

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

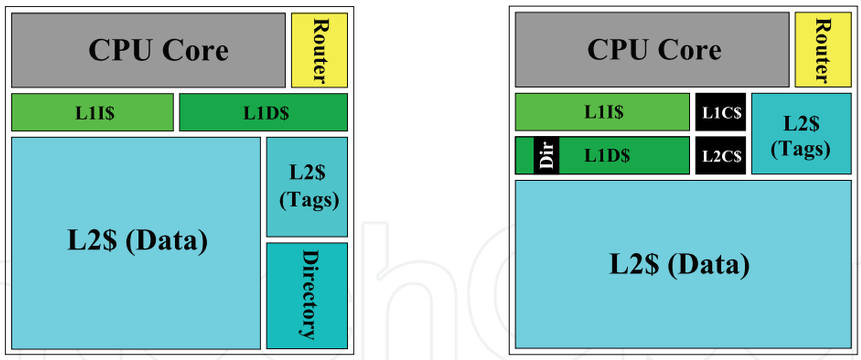
ure 3 and three in Figure 5). Moreover, this allows the O&D node to solve cache misses without using transient states, thus reducing the number of states and making the implementation simpler than a directory protocol. Finally, the elimination of transient states at the directory reduces waiting time for the subsequent requests and, therefore, average miss latency.

4.2 Changes to the structure of the tiles of a CMP

The new distribution of roles that characterizes direct coherence protocols requires some modifications in the structure of the tiles that build the CMP. Firstly, the identity of the sharers for every block is stored in the corresponding owner tile instead of the home one to allow caches to keep coherence for the memory blocks that they hold in the owned state. Therefore, *DiCo-CMP* extends the tags' part of the L1 caches with a sharing code field, e.g., a full-map (L2 caches already include this field in directory protocols). In contrast, *DiCo-CMP* does not need to store a directory structure at the home tile, as happens in directory protocols.
Additionally, *DiCo-CMP* adds two extra hardware structures that are used to record the identity of the owner tile of the memory blocks stored on chip:

- *L1 coherence cache* (L1C$): The pointers stored in this structure are used by the requesting core to avoid indirection by directly sending local requests to the corresponding owner tile. Therefore, this structure is located close to each processor's core. Although *DiCo-CMP* can update this information in several ways, we consider in this chapter the *Base* policy presented in Ros et al. (2008a), in which this information is updated by using the coherence messages sent by the protocol, i.e., invalidation and data messages.
- *L2 coherence cache* (L2C$): Since the owner tile can change on write misses, this structure must track the owner tile for each block allocated in any L1 cache. This structure replaces the directory structure required by directory protocols and it is accessed each time a request fails to locate the owner tile. This information must be updated whenever the owner tile changes through control messages. These messages must be processed by the L2C$ in the very same order in which they were generated in order to avoid any incoherence when storing the identity of the owner tile, as described later in Section 4.3.3.

Figure 6 shows a tile design for directory protocols and for direct coherence protocols. A comparison among the extra storage and structures required by all the protocols evaluated in this chapter can be found in Section 7.4.



(a) Organization of a tile for directory protocols.

(b) Organization of a tile for direct coherence protocols.

Fig. 6. Modifications to the structure of a tile required by direct coherence protocols.

4.3 Description of the cache coherence protocol

4.3.1 Requesting processor

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

4.3.2 Request received by a tile that is not the owner

When a request is received by a tile that is not the current owner of the block, it simply re-sends the request. If the tile is not the home one, the request is re-sent to it. Otherwise, if the request is received by the home tile and there is a hit in the L2C$, the request is sent to the current owner tile. In absence of race conditions the request will reach the owner tile. Finally, if there is a miss in the L2C$ and the home tile is not the owner of the block, the request is solved by providing the block from main memory, where, in this case, a fresh copy of the block resides. This is because the L2C$ always keeps an entry for the blocks stored in the private L1 caches. If the owner copy of the block is not present in either any L1 cache or in the L2 cache, it resides off-chip. After the off-chip access, the block is allocated in the requesting L1 cache, which gets the ownership of the block, but not in the L2 cache (as occurs in the other protocols evaluated), since we assume that the L1 and the L2 cache are non-inclusive. In addition, it is necessary to allocate a new entry in the L2C$ pointing to the current L1 owner tile.

4.3.3 Request received by the owner tile

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

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

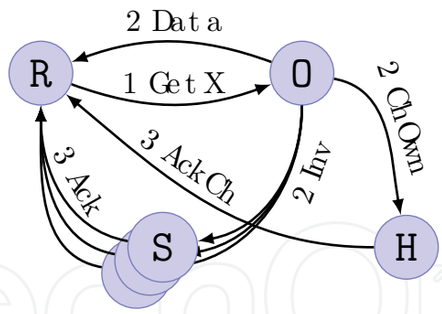


Fig. 7. Example of ownership change upon write misses. R=Requester; O=Owner; S=Sharers; H=Home.

When the owner is an L1 cache, read misses are completed by sending a copy of the block to the requester and adding it to the sharing code field kept along with the block. For write misses, the owner tile sends invalidation messages to all the tiles that hold a copy of the block in their L1 caches and, then, it sends the data block to the requester. Acknowledgement messages are collected at the requesting core. As previously shown in Figure 5, write misses (upgrade) that take place in the owner tile just need to send invalidations and receive acknowledgements (two hops in the critical path).

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

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

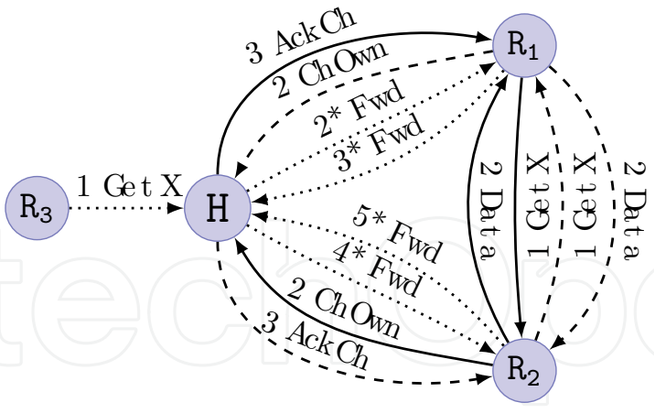


Fig. 8. Example of a starvation scenario in direct coherence protocols. R_x=Requester; H=Home. Continuous arrows represent cache misses that take place in R_1, dashed arrows represent misses in R_2 and dotted arrows represent misses in R_3.

4.3.4 Replacements

In our particular implementation, when a block with the ownership property is evicted from an L1 cache, it must be allocated at the L2 cache along with the up-to-date directory information. Differently from *Directory-CMP* and *Hammer-CMP* protocols and similarly to *Token-CMP*, replacements are performed by sending the writeback message directly to the home tile (instead of requiring three-phase replacements). This operation can be easily performed in direct coherence protocols because the tile where these blocks are stored is the responsible for keeping cache coherence and, as consequence, no complex race conditions can appear. When the writeback message reaches the home tile, the L2C$ deallocates its entry for this block because the owner tile is now the home one. On the other hand, replacements for blocks in shared state are performed transparently, i.e., no coherence actions are needed.

Finally, no coherence actions must be performed in case of an L1C$ replacement. However, when an L2C$ entry is evicted, the protocol should ask the owner tile to invalidate all the copies from the private L1 caches. Luckily, as happens to the directory cache in directory protocols, an L2C$ with the same number of entries and associativity than the L1 cache is enough to completely remove this kind of replacements (Ros et al., 2008b).

4.4 Preventing starvation

Directory protocols avoid starvation by enqueuing requests in FIFO order at the directory buffers. Differently in *DiCo-CMP*, write misses can change the tile that keeps coherence for a particular block and, therefore, some requests can take some extra time until this tile is finally found. If a memory block is repeatedly written by several processors, a request could take some time to find the owner tile ready to process it, even when it is sent by the home tile. Hence, some processors could be solving their requests while other requests are starved. Figure 8 shows an example of a scenario in which starvation appears. R_1 and R_2 tiles are issuing write requests repeatedly and, therefore, the owner tile is continuously moving from R_1 to R_2 and vice versa. On every change of owner the home tile is notified, and the requesting core is acknowledged. However, at the same time, the home tile is trying to re-send the request issued by R_3 tile to the owner one, but the request is always returned to the home tile because the write request issued by R_1 or R_2 arrives before to the owner tile.

DiCo-CMP detects and avoids starvation by using a simple mechanism. In particular, each time that a request must be re-sent to the L2C$ in the home tile, a counter into the request message is increased. The request is considered starved when this counter reaches a certain value (e.g, three accesses to the L2C$ for the evaluation carried out in this chapter). When the L2C$ detects a starved request, it re-sends the request to the owner tile, but it records the address of the block. If the starved request reaches the current owner tile, the miss is solved, and the home tile is notified, ending the starvation situation. If the starved request does not reach the owner tile is because the ownership property is moving from a tile to another one. In this case, when the message informing about the change of the ownership arrives to the home tile, it detects that the block is suffering from starvation, and the acknowledgement message required on every ownership change is not sent. This ensures that the owner tile does not change until the starved request can complete.

5. Reducing area requirements in DiCo-CMP

DiCo-CMP needs two structures that keep the identity of the tile where the owner copy of the block resides, the L1C$ and the L2C$. These two structures do not compromise scalability because they have a small number of entries and each one stores a tag and a pointer to the owner tile ($log_2 n$ bits, where n is the number of cores). The L2C$ is needed to solve cache misses in *DiCo-CMP*, since it ensures that the tile that keeps coherence for each block can always be found. On the other hand, the L1C$ is required to avoid indirection in cache misses and, therefore, it is essential to obtain good performance. Moreover, the L2C$ allows read misses to be solved by sending only one forwarding request to the owner tile, since it stores the identity of the owner tile, which significantly reduces network traffic when compared to broadcast-based protocols.

Apart from these structures, *DiCo-CMP* also adds a full-map sharing code to each data cache entry. The memory overhead introduced by this sharing code could become prohibitive in many-core CMPs. In this section, we describe some alternatives that differ in the sharing code scheme added to each entry of the data caches. Since these alternatives always include the L1C$ and the L2C$, they have area requirements of at least $O(log_2 n)$. The particular compressed sharing code employed impacts on the number of invalidations sent in write misses. Next, we comment on the different implementations of direct coherence protocols that we have evaluated.

DiCo-FM is the *DiCo-CMP* protocol described in Ros et al. (2008a) and, therefore, it adds a full-map sharing code to each data cache. Particularly, we evaluate the *Base* policy presented in that work, which obtains good performance with low traffic overhead.

DiCo-CV-K reduces the size of the sharing code field by using a *coarse vector* (Gupta et al., 1990) instead of a full-map sharing code. In a coarse vector, each bit represents a group of K tiles, instead of just one. A bit is set when at least one of the tiles in the group holds the block in its private cache. Therefore, even when just one of the tiles in the group requested a particular block, all tiles belonging to that group will receive an invalidation message before the block can be written. Particularly, we study a configuration that uses a coarse vector sharing code with $K = 2$. In this case, 8 bits are needed for a 16-core configuration. Although this sharing code reduces the memory required by the protocol, its size still increases linearly with the number of cores.

DiCo-LP-P employs a *limited pointers* sharing code (Chaiken et al., 1991). In this scheme, each entry has a limited number of pointers for the first P sharers of the block. Actually, since *DiCo-CMP* always stores the information about the owner tile in the L2C$, the first pointer

Protocol	Sharing Code	Bits L1 and L2	Bits L1C$ and L2C$	Order
DiCo-FM	Full-map	n	$log_2 n$	$O(n)$
DiCo-CV-K	Coarse vector	$\frac{n}{K}$	$log_2 n$	$O(n)$
DiCo-LP-P	Limited pointers	$1 + P \times (1 + log_2 n)$	$log_2 n$	$O(log_2 n)$
DiCo-BT	Binary Tree	$\lceil log_2(1 + log_2 n) \rceil$	$log_2 n$	$O(log_2 n)$
DiCo-NoSC	None	0	$log_2 n$	$O(log_2 n)$

Table 2. Bits required for storing coherence information.

is employed to store the identity of the second sharer of the block. When the sharing degree of a block is greater than $P + 1$, write misses are solved by broadcasting invalidations to all tiles. Therefore, apart from the pointers, it is necessary an extra bit indicating the overflow situation. However, this situation is not very frequent since the sharing degree of the applications is usually low (Culler et al., 1999). In particular, we evaluate this protocol with a P value of 1. Under this assumption, the number of bits needed to store the sharing information considering 16 cores is 5.

DiCo-BT uses a sharing code based on a *binary tree* (Acacio et al., 2001). In this approach, tiles are recursively grouped into clusters of two elements, thus leading to a binary tree with the tiles located at the leaves. The information stored in the sharing code is the smallest cluster that covers all the sharers. Since this scheme assumes that for each block the binary tree is computed from a particular leave (the one representing the home tile), it is only necessary to store the number of the level in the tree, i.e., 3 bits for a 16-core configuration.

Finally, *DiCo-NoSC* (no sharing code) does not maintain any coherence information along with the owner block. In this way, this protocol does not need to modify the structure of data caches to add any field. This lack of information implies broadcasting invalidation messages to all tiles upon write misses, although this is only necessary for blocks in shared state because the owner tile is always known in *DiCo-CMP*. This scheme incurs in more network traffic compared to the previous ones. However, it falls into less traffic than *Hammer-CMP* and *Token-CMP*. This is because *Hammer-CMP* requires broadcasting requests on every cache miss, and what is more expensive in a network with multicast support, every tile that receives the request answers with a control message. On the other hand, although *Token-CMP* avoids these response messages, it also relies on broadcasting requests for all cache misses.

Table 2 shows the number of bits required for storing coherence information in each implementation, both for the coherence caches (L1C$ and L2C$) and for the data caches (L1 and L2). Other compressed sharing codes, like *tristate* (Agarwal et al., 1988), *gray-tristate* (Mukherjee & Hill, 1994) or *binary tree with subtrees* (Acacio et al., 2001) could also be implemented instead of those shown in this table. However, for a 16-core tiled CMP, they incur in similar overhead than *DiCo-CV-2* (8, 8 and 7 bits respectively), which does not significantly increases network traffic, as we will see in Section 7.3. For a greater number of cores, these compressed sharing codes could be more appropriate.

6. Simulation environment

We perform the evaluation using the full-system simulator Virtutech Simics (Magnusson et al., 2002) extended with Multifacet GEMS 1.3 (Martin et al., 2005), that provides a detailed memory system timing model. Since the network modeled by GEMS 1.3 is not very precise, we have extended it with SICOSYS (Puente et al., 2002), a detailed interconnection network sim-

GEMS Parameters		SICOSYS Parameters	
Processor frequency	3 GHz	Network frequency	1.5 GHz
Cache hierarchy	Non-inclusive	Topology	4x4 Mesh
Cache block size	64 bytes	Switching technique	Wormhole, Multicast
Split L1 I & D caches	128KB, 4 ways, 3 hit cycles	Routing technique	Deterministic X-Y
Shared unified L2 cache	1MB/tile, 8 ways, 6 hit cycles	Data message size	4 flits
		Control message size	1 flit
L1C$ & L2C$	512 sets, 4 ways, 2 hit cycles	Routing time	2 cycles
Directory cache	512 sets, 4 ways, 2 hit cycles	Link latency (one hop)	2 cycles
Memory access time	300 cycles	Link bandwidth	1 flit/cycle

Table 3. System parameters.

ulator. We simulate CMP systems with 16 tiles. Table 3 shows the values of the main parameters used for the evaluation, where cache latencies have been calculated using the CACTI 5.3 tool (Thoziyoor et al., 2008) for 45nm technology. We also have used CACTI to measure the area of the different structures needed in each one of the evaluated protocols. In this study, we assume that the length of the physical address is 40 bits, like in the SUN UltraSPARC-III architecture (Horel & Lauterbach, 1999).

The ten applications used in our simulations cover a variety of computation and communication patterns. *Barnes* (8192 bodies, 4 time steps), *FFT* (64K points), *Ocean* (130x130 ocean), *Radix* (512K keys, 1024 radix), *Raytrace* (teapot), *Volrend* (head) and *Water-Nsq* (512 molecules, 4 time steps) are scientific applications from the SPLASH-2 benchmark suite (Woo et al., 1995). *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 (Li et al., 2005). 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.

7. Evaluation results

In this section, we compare the different alternatives described in Section 5 with all the base protocols described in this chapter. First, we show to what extent direct coherence protocols avoid indirection, and its impact on execution time. Then, we analyze the network traffic generated by each protocol, and the area required by them to store the coherence information. Finally, we summarize these results by showing the trade-off in terms of execution time, network traffic and area requirements of the protocols evaluated.

7.1 Impact on indirection

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

- *2-hop misses*: Misses belonging to this category does not suffer from indirection since the number of hops in the critical path of the miss is two. In *Hammer-CMP*, misses fall into this category when the home tile of the requested block can provide the copy of

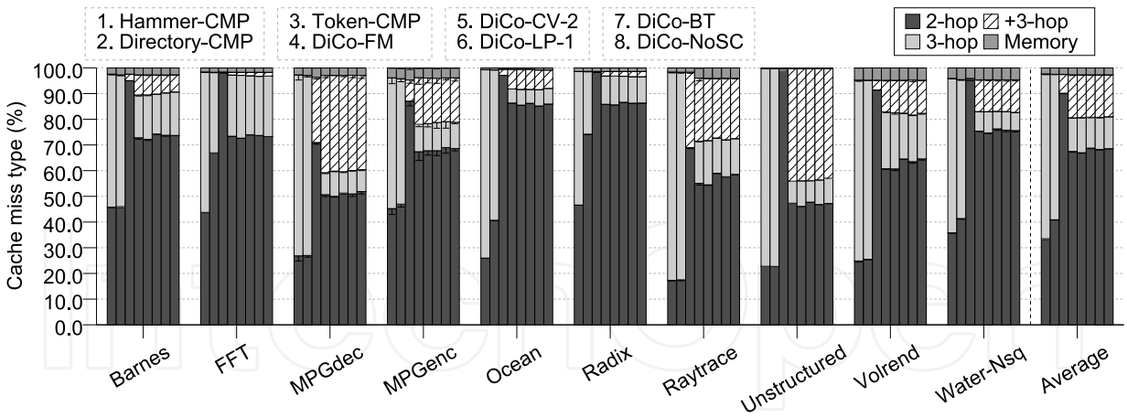


Fig. 9. How each miss type is solved for the applications evaluated in this chapter.

the block and it is not necessary to invalidate blocks from other tiles. In directory protocols, misses fall into this category in the same cases as *Hammer-CMP*, but also when the miss takes place in the home tile. *Token-CMP* solves all misses that do not require persistent requests in two hops. Finally, *DiCo-CMP* solves cache misses using two hops either when the request is directly sent to the current owner tile and invalidations are not required or when the miss takes place either in the home tile or in the owner tile (upgrades).

In all protocols, when the miss takes place in the home tile and this tile holds the owner block in the L2 cache, the miss is solved without generating network traffic (0-hop miss). These misses are also included in this category because they do not introduce indirection.

- *3-hop misses*: A miss belongs to this category when three hops in the critical path are necessary to solve it. This category represents the misses suffering from indirection in traditional protocols. In contrast, 3-hop misses never take place in *Token-CMP*.
- *+3-hop misses*: We include in this category misses that need more than three hops in the critical path to be solved. This type of misses only happens in *DiCo-CMP*, when the identity of the owner tile is mis-predicted, or in *Token-CMP*, when persistent requests are required to solve the miss. The traditional protocols evaluated in this chapter never require more than three hops to solve cache misses since the acknowledgements to invalidation messages are collected by the requesting core.
- *Memory misses*: Misses that require off-chip accesses since the owner block is not stored on chip fall into this category.

Figure 9 shows the percentage of cache misses that fall into each category. As commented in Section 2, in tiled CMP architectures it is not very frequent that the requester tile be the home one for the requested block because the distribution of blocks among tiles is performed in a round-robin fashion. Therefore, traditional protocols have a lot of cache misses with indirection. However, the fact that sometimes a coherent copy of the block is found in the L2 cache bank of the home tile, decreases the number of misses with indirection. In this way, the first and second bars in Figure 9 shows that most applications have an important fraction of misses suffering from indirection when traditional protocols are considered, like *Barnes*, *MPGdec*,

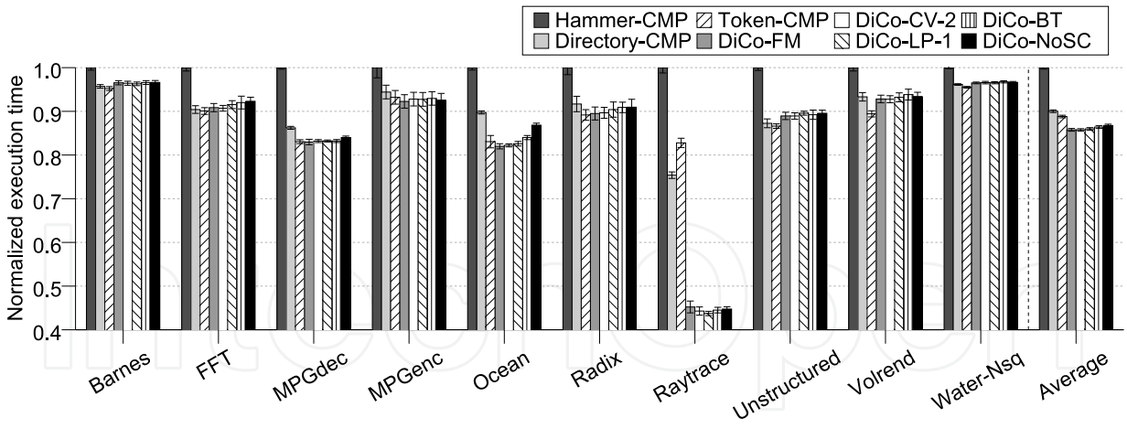


Fig. 10. Normalized execution times.

MPGenc, *Ocean*, *Raytrace*, *Unstructured*, *Volrend* and *Water-Nsq*, while other applications, like *FFT* and *Radix*, have most of the misses solved in two hops when a directory protocol is considered. *Hammer-CMP* has more cache misses suffering from indirection because sometimes it has to broadcast forwarding messages due to the lack of information about the identity of the owner tile. Obviously, *DiCo-CMP* will have more impact for the applications that suffer more indirection, although this impact will also depend on the cache miss rate of each application. We also can observe that *Token-CMP* solves most of the misses (90%) needing just two hops (see third bar).

As shown in the fourth bar of Figure 9, *DiCo-FM* increases the percentage of cache misses without indirection compared to both *Hammer-CMP* and *Directory-CMP* (from 34% and 41%, respectively, to 67% on average). On the other hand, *DiCo-FM* solves 17% of cache misses needing more than three hops. This fact is due to owner mis-predictions that can arise for two reasons: (1) staled owner information was found in the L1C$ or (2) the owner tile is changing or busy due to race conditions and the request is sent back to the home tile. Although, the first case can be removed with a precise hints mechanism, as discussed in (Ros et al., 2008a), in this chapter we do not use this mechanism in order to save network traffic.

The remaining bars show the different implementations of direct coherence aimed at reducing the area requirements entailed by this protocol. We can see that, the indirection avoidance is similar. However, the more compressed is the sharing code, the more invalidations are sent, which slightly increases the number of misses without indirection due to a better prediction of owner tiles.

7.2 Impact on execution time

Figure 10 plots the average execution times for the applications evaluated in this chapter normalized with respect to *Hammer-CMP*. Compared to *Hammer-CMP*, *Directory-CMP* improves performance for all applications as a consequence of an important reduction in terms of both misses suffering from indirection and network traffic (as we will see in next section). As discussed in the previous section, the longer latency cache misses are suffered in *Hammer-CMP*. This is because on each cache miss the requesting core must wait for all the acknowledgement messages before the miss can be solved. On the contrary, in *Directory-CMP* only write misses must wait for acknowledgements.

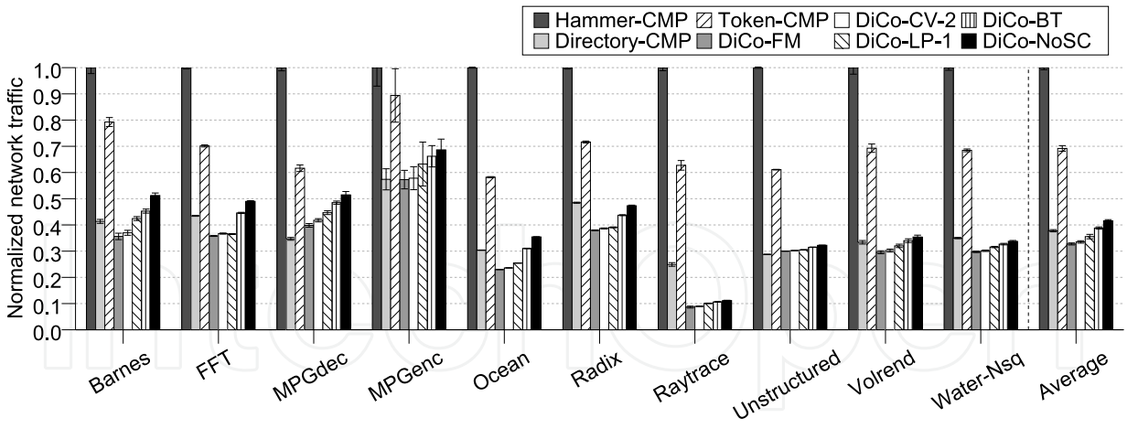


Fig. 11. Normalized network traffic.

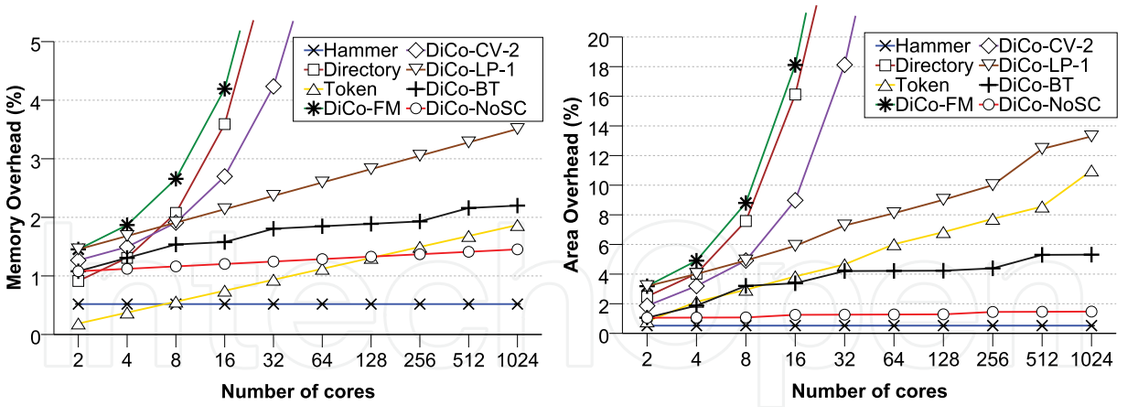
On the other hand, indirection-aware protocols reduce average execution time when compared to traditional protocols. Particularly, *Token-CMP* obtains average improvements of 11% compared to *Hammer-CMP* and 1% compared to *Directory-CMP*. *DiCo-FM* improves the execution time by 14%, 5% and 4% compared to *Hammer-CMP*, *Directory-CMP* and *Token-CMP*, respectively. On the other hand, when *DiCo-CMP* employs compressed sharing codes, the execution time slightly increases. Although the protocol incurs in more network traffic, it also increases the accuracy of owner predictions. Therefore, it remains close to *DiCo-FM*. For *DiCo-CV-2* and *DiCo-LP-1* the increase in execution time is negligible, while *DiCo-BT* and *DiCo-NoSC* increase execution time by 1%.

7.3 Impact on network traffic

Figure 11 compares the network traffic generated by the protocols discussed previously. Each bar plots the number of bytes transmitted through the interconnection network normalized with respect to *Hammer-CMP*.

As expected, *Hammer-CMP* introduces much more network traffic than the other protocols due to the lack of coherence information, which implies broadcasting requests to all cores and receiving the corresponding acknowledgements. *Directory-CMP* reduces considerably traffic by adding a full-map sharing code that filters unnecessary invalidations. *Token-CMP* generates more network traffic than *Directory-CMP*, because it relies on broadcasting requests, and less than *Hammer-CMP*, because it does not need to receive acknowledgements from tiles without tokens (i.e., the tiles that do not share the block). Finally, *DiCo-FM* decreases traffic requirements compared to *Directory-CMP* (by 13%) due to the elimination of control messages between the owner and the home tile, as discussed in Section 4.

In general, we can see that compressed sharing codes increase network traffic compared to a full-map sharing code. However, the increase in traffic is admissible. Particularly, the most scalable alternatives, *DiCo-LP-1*, *DiCo-BT* and *DiCo-NoSC*, increase network traffic by 8%, 16% and 21% compared to *DiCo-FM*, respectively. *DiCo-BT* has similar traffic requirements than *Directory-CMP*, and *DiCo-NoSC*, which does not have any sharing code, generates an acceptable amount of network traffic (40% less traffic than *Token-CMP* and 58% less traffic than *Hammer-CMP*).



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

Fig. 12. Overhead introduced by the cache coherence protocols.

7.4 Impact on area overhead

Finally, we compare the memory overhead introduced by the coherence information for the cache coherence protocols evaluated in this chapter. Although some protocols can entail extra overhead as a consequence of the additional mechanisms that they demand (e.g., timeouts for reissuing requests or large tables for keeping active persistent requests in *Token-CMP*), we only consider the amount of memory needed to keep coherence information. Obviously, the extra tags required to store this information (e.g., for the L1C$ and L2C$) are also considered in this study. Figure 12 shows the storage overhead introduced by these protocols in terms of both number of bits and estimated area (calculated with the CACTI tool). The overhead is plotted for varying number of cores from 2 to 1024.

Although the original *Hammer* protocol does not require any coherence information, our optimized version for CMPs adds a new structure to the home tile. This structure is a 512-set 4-way cache that contains a copy of the tags for blocks stored in the L1 caches but not in the L2 cache. However, this structure introduces a slight overhead which keeps constant with the number of cores.

Directory-CMP stores the directory information either in the L2 tags, when the L2 cache holds a copy of the block, or in a distributed directory cache, when the block is stored in any of the L1 caches but not in the L2 cache. Since the information is stored using a full-map sharing code, the number of required bits is n, and consequently the width of each directory entry grows linearly with the number of cores.

Token-CMP keeps the token count for any block stored both in the L1 and L2 caches. This information only requires $\lceil log_2(n+1) \rceil$ bits for both the owner-token bit and the non-owner token count. These additional bits are stored in the tags' part of both cache levels. In this way, *Token-CMP* has acceptable scalability in terms of area.

DiCo-FM stores directory information along with each owner block held in the L1 and L2 caches. Therefore, a full-map sharing code is added to the tags' part of each cache entry. Moreover, it uses two structures that store the identity of the owner tile, the L1C$ and the L2C$. Each entry in these structures contains a tag and an owner field, which requires $log_2 n$ bits. Therefore, this is the protocol that more area overhead entails.

We propose to reduce this overhead by introducing compressed sharing codes in *DiCo-CMP*. *DiCo-CV-2* saves storage compared to *DiCo-FM* but it is still non-scalable. In contrast, *DiCo-*

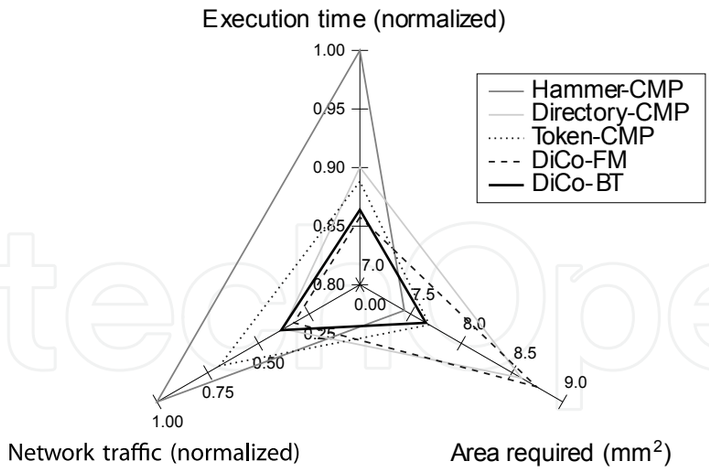


Fig. 13. Trade-off of the three main design goals.

LP-1, which only adds a pointer for the second sharer of the block (the first one is given by the L2C$) has better scalability –$O(log_2 n)$–. *DiCo-BT* reduces even more the area requirements compared to *DiCo-LP-1*, and it scales better than *Token-CMP*. Finally, *DiCo-NoSC*, which does not require to modify data caches to add coherence information, is the implementation of DiCo with less overhead (although it still has order $O(log_2 n)$ due to the need of the coherence caches), at the cost of increasing network traffic. Finally, we can see that a small overhead in the number of required bits results in a significant overhead when the area of the structures is considered.

7.5 Trade-off analysis

Figure 13 shows the trade-off among execution time, network traffic, and area requirements for the base protocols evaluated in this chapter, *DiCo-FM*, and *DiCo-BT*, which constitutes a good alternative when the three metrics evaluated in this chapter are considered. In this way, this graph summarizes the evaluation carried out in this chapter. Results in terms of execution time and network traffic represent the average of all applications, normalized with respect to *Hammer-CMP*. Results in terms of area requirements correspond to the area in mm^2 of each protocol considering both the data caches and the extra structures required to keep the coherence information.

We can see that, in general, the base protocols aimed to be used with tiled CMPs do not have a good trade-off. *Hammer-CMP* has the highest traffic levels and execution times, but also the lowest area requirements ($7.4mm^2$). In contrast, *Directory-CMP*, which reduces both execution time and network traffic compared to *Hammer-CMP* (by 10% and 61%, respectively), at the cost of increasing area requirements ($8.59mm^2$ for a 16-tiled CMP, and $O(n)$). Although *Token-CMP* has acceptable area requirements ($7.68mm^2$ for a 16-tiled CMP) it is limited by traffic, requiring twice the traffic required by *Directory-CMP*. Finally, *DiCo-FM*, that reduces both execution time and traffic requirements when compared to *Token-CMP* (by 4% and 47%, respectively), is the one with the highest area requirements ($8.74mm^2$ for a 16-tiled CMP, and $O(n)$).

However, the use of different compressed sharing codes for *DiCo-CMP* can lead to a good compromise between network traffic and area requirements, and still guaranteeing low average execution time. In general, *DiCo-LP-1*, *DiCo-BT* and *DiCo-NoSC* are very close to an

ideal protocol with the best characteristics of the base protocols, for the sake of clarity, we only show the trade-off for *DiCo-BT*. *DiCo-BT* requires less area ($7.65mm^2$ for a 16-tiled CMP) than all evaluated protocols except *Hammer-CMP*, it also generates similar network traffic than *Directory-CMP* and, finally, it has a low average execution time (increasing just by 1% the best approach, *DiCo-FM*).

8. Related work

In the shared-memory multiprocessors domain, Acacio et al. propose to avoid the indirection for cache-to-cache transfer misses (Acacio et al., 2002a) and upgrade misses (Acacio et al., 2002b) separately by predicting the current holders of every cache block. Predictions must be verified by the corresponding directory controller, thus increasing the complexity of the protocol on mis-predictions. Hossain et al. (2008) propose different optimizations for each sharing pattern considering a chip multiprocessor architecture. Particularly, they accelerate the producer-consumer pattern by converting 3-hop read misses into 2-hop read misses. Again, communication between the cache providing the data block and the directory is necessary, thus introducing more complexity in the protocol. In contrast, direct coherence is applicable to all types of misses (reads, writes and upgrades) and just the identity of the owner tile is predicted. Moreover, the fact that the directory information is stored along with the owner of the block simplifies the protocol. Finally, differently from the techniques proposed by Acacio et al., direct coherence avoids predicting the current holders of a block by storing the up-to-date directory information in the owner tile.

Also in the context of shared-memory multiprocessors, Cheng et al. (2007) have proposed converting 3-hop read misses into 2-hop read misses for memory blocks that exhibit the producer-consumer sharing pattern by using extra hardware to detect when a block is being accessed according to this pattern. In contrast, direct coherence obtains 2-hop misses for read, write and upgrade misses without taking into account sharing patterns.

Jerger et al. (2008) propose Virtual Tree Coherence (VTC). This mechanism uses coarse-grain coherence tracking (Cantin et al., 2006) and the sharers of a memory region are connected by means of a virtual tree. Since the root of the virtual tree serves as the ordering point in place of the home tile, and the root tile is one of the sharers of the region, the indirection can be avoided for some misses. Contrarily, direct coherence protocols keep the coherence information at block granularity and the ordering point always has the valid copy of the block, which leads to less network traffic and lower levels of indirection.

Huh et al. (2005) propose to allow replication in a NUCA cache to reduce the access time to a shared multibanked cache. More recently, Beckmann et al. (2006) present ASR that replicates cache blocks only when it is estimated that the benefits of replication (lower L2 hit latency) exceeds its costs (more L2 misses). In contrast, direct coherence reduces miss latencies by avoiding the access to the L2 cache when it is not necessary, and no replication is performed. It could be also used in conjunction with techniques that try to make the best use of the limited on-chip cache storage.

Martin et al. (2000) present a technique that allows snooping-based protocols to utilize unordered networks by adding logical timing to coherence requests and reordering them on destiny to establish a total order. Likewise, Agarwal et al. (2009) propose In-Network Snoop Ordering (INSO) to allow snooping over unordered networks. Since direct coherence protocols do not rely on broadcasting requests, they generate less traffic and, therefore, less power consumption when compared to snooping-based protocols.

Martin et al. (2003) propose to use destination-set prediction to reduce the bandwidth required by a snoopy protocol. Differently from DiCo-CMP, this proposal is based on a totally-ordered interconnect (a crossbar switch), which does not scale with the number of nodes. Destination-set prediction is also used by Token-M in shared-memory multiprocessors with unordered networks (Martin, 2003). However, on mis-predictions, requests are solved by resorting on broadcasting after a time-out period. Differently, in direct coherence protocols mis-predictions are re-sent immediately to the owner cache, thus reducing both latency and network traffic.

9. Conclusions

Tiled CMP architectures have recently emerged as a scalable alternative to current small-scale CMP designs, and will be probably the architecture of choice for future many-core CMPs. On the other hand, although a great deal of attention was devoted to scalable cache coherence protocols in the last decades in the context of shared-memory multiprocessors, the technological parameters and constraints entailed by CMPs demand new solutions to the cache coherence problem. New cache coherence protocols, like *Token-CMP* and *DiCo-CMP*, have been recently proposed to cope with the indirection problem of traditional protocols. However, neither *Token-CMP* nor *DiCo-CMP* scale efficiently with the number of cores, and future cache coherence protocols need to be efficient in terms of execution time, network traffic generated and area requirements.

In this chapter, we take into consideration these three constraints, and we discuss and evaluate both protocols that are used nowadays, such as *Hammer* and *Directory*, and novel indirection-aware protocols, such as *Token-CMP* and *DiCo-CMP*. In this way, we perform a detailed evaluation of a wide range of cache coherence protocols for many-core CMPs in a common framework. We also study several implementations of *DiCo-CMP* that differ in the amount of coherence information that they store in order to achieve the best trade-off among the three constraints considered.

Particularly, we show that *DiCo-LP-1*, which only stores the identity of one sharer along with the data block, *DiCo-BT*, which codifies the directory information just using three bits, and *DiCo-NoSC*, which does not store any coherence information in the data caches (and it does not need to modify the structure of the caches), are the alternatives that achieve a better trade-off. For example, *DiCo-BT* requires less area than all evaluated protocols, except *Hammer-CMP*, it also generates similar network traffic than *Directory-CMP* and, finally, it has a low average execution time (increasing just by 1% the best approach, *DiCo-FM*).

10. Acknowledgements

This work has been jointly supported by Spanish MEC 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).

11. References

Acacio, M. E., González, J., García, J. M. & Duato, J. (2001). A new scalable directory architecture for large-scale multiprocessors, *7th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 97–106.

Acacio, M. E., González, J., García, J. M. & Duato, J. (2002a). Owner prediction for accelerating cache-to-cache transfer misses in cc-NUMA multiprocessors, *SC2002 High Performance Networking and Computing*.

Acacio, M. E., González, J., García, J. M. & Duato, J. (2002b). The use of prediction for accelerating upgrade misses in cc-NUMA multiprocessors, *11th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 155–164.

Agarwal, A., Simoni, R., Hennessy, J. L. & Horowitz, M. A. (1988). An evaluation of directory schemes for cache coherence, *15th Int'l Symp. on Computer Architecture (ISCA)*, pp. 280–289.

Agarwal, N., Peh, L.-S. & Jha, N. K. (2009). In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects, *15th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 67–78.

Agarwal, V., Hrishikesh, M. S., Keckler, S. W. & Burger, D. (2000). Clock rate versus IPC: the end of the road for conventional microarchitectures, *27th Int'l Symp. on Computer Architecture (ISCA)*, pp. 248–259.

Ahmed, A., Conway, P., Hughes, B. & Weber, F. (2002). AMD Opteron^{TM} shared-memory MP systems, *14th HotChips Symp.*

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

Barroso, L. A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R. & Verghese, B. (2000). Piranha: A scalable architecture based on single-chip multiprocessing, *27th Int'l Symp. on Computer Architecture (ISCA)*, pp. 12–14.

Beckmann, B. M., Marty, M. R. & Wood, D. A. (2006). ASR: Adaptive selective replication for CMP caches, *39th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pp. 443–454.

Bosschere, K. D., Luk, W., Martorell, X., Navarro, N., O'Boyle, M., Pnevmatikatos, D., Ramirez, A., Sainrat, P., Seznec, A., Stenstrom, P. & Temam, O. (2007). High-performance embedded architecture and compilation roadmap, *Transactions on HiPEAC I* pp. 5–29.

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

Chaiken, D., Kubiatowicz, J. & Agarwal, A. (1991). LimitLESS directories: A scalable cache coherence scheme, *4th Int. Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pp. 224–234.

Cheng, L., Carter, J. B. & Dai, D. (2007). An adaptive cache coherence protocol optimized for producer-consumer sharing, *13th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 328–339.

Culler, D. E., Singh, J. P. & Gupta, A. (1999). *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann Publishers, Inc.

Gupta, A., Weber, W.-D. & Mowry, T. C. (1990). Reducing memory traffic requirements for scalable directory-based cache coherence schemes, *Int'l Conference on Parallel Processing (ICPP)*, pp. 312–321.

Ho, R., Mai, K. W. & Horowitz, M. A. (2001). The future of wires, *Proceedings of the IEEE* **89**(4): 490–504.

Horel, T. & Lauterbach, G. (1999). UltraSPARC-III: Designing third-generation 64-bit performance, *IEEE Micro* **19**(3): 73–85.

Hossain, H., Dwarkadas, S. & Huang, M. C. (2008). Improving support for locality and fine-grain sharing in chip multiprocessors, *17th Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 155–165.

Huh, J., Kim, C., Shafi, H., Zhang, L., Burger, D. & Keckler, S. W. (2005). A NUCA substrate for flexible CMP cache sharing, *19th Int'l Conference on Supercomputing (ICS)*, pp. 31–40.

Jerger, N. D. E., Peh, L.-S. & Lipasti, M. H. (2008). Virtual tree coherence: Leveraging regions and in-network multicast tree for scalable cache coherence, *41th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pp. 35–46.

Kim, C., Burger, D. & Keckler, S. W. (2002). An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches, *10th Int. Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pp. 211–222.

Kumar, R., Zyuban, V. & Tullsen, D. M. (2005). Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling, *32nd Int'l Symp. on Computer Architecture (ISCA)*, pp. 408–419.

Le, H. Q., Starke, W. J., Fields, J. S., O'Connell, F. P., Nguyen, D. Q., Ronchetti, B. J., Sauer, W. M., Schwarz, E. M. & Vaden, M. T. (2007). IBM POWER6 microarchitecture, *IBM Journal of Research and Development* **51**(6): 639–662.

Li, M.-L., Sasanka, R., Adve, S. V., Chen, Y.-K. & Debes, E. (2005). The ALPBench benchmark suite for complex multimedia applications, *Int'l Symp. on Workload Characterization*, pp. 34–45.

Magen, N., Kolodny, A., Weiser, U. & Shamir, N. (2004). Interconnect-power dissipation in a microprocessor, *Int'l workshop on System Level Interconnect Prediction (SLIP'04)*, pp. 7–13.

Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A. & Werner, B. (2002). Simics: A full system simulation platform, *IEEE Computer* **35**(2): 50–58.

Martin, M. M. (2003). *Token Coherence*, PhD thesis, University of Wisconsin-Madison.

Martin, M. M., Harper, P. J., Sorin, D. J., Hill, M. D. & Wood, D. A. (2003). Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors, *30th Int'l Symp. on Computer Architecture (ISCA)*, pp. 206–217.

Martin, M. M., Hill, M. D. & Wood, D. A. (2003). Token coherence: Decoupling performance and correctness, *30th Int'l Symp. on Computer Architecture (ISCA)*, pp. 182–193.

Martin, M. M., Sorin, D. J., Ailamaki, A., Alameldeen, A. R., Dickson, R. M., Mauer, C. J., Moore, K. E., Plakal, M., Hill, M. D. & Wood, D. A. (2000). Timestamp snooping: An approach for extending SMPs, *9th Int. Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pp. 25–36.

Martin, M. M., Sorin, D. J., Beckmann, B. M., Marty, M. R., Xu, M., Alameldeen, A. R., Moore, K. E., Hill, M. D. & Wood, D. A. (2005). Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, *Computer Architecture News* **33**(4): 92–99.

Marty, M. R., Bingham, J., Hill, M. D., Hu, A., Martin, M. M. & Wood, D. A. (2005). Improving multiple-cmp systems using token coherence, *11th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pp. 328–339.

Mukherjee, S. S. & Hill, M. D. (1994). An evaluation of directory protocols for medium-scale shared-memory multiprocessors, *8th Int'l Conference on Supercomputing (ICS)*, pp. 64–74.

Owner, J. M., Hummel, M. D., Meyer, D. R. & Keller, J. B. (2006). *System and method of maintaining coherency in a distributed communication system*, U.S. Patent 7069361.

Puente, V., Gregorio, J. A. & Beivide, R. (2002). SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems, *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pp. 15–22.

Ros, A., Acacio, M. E. & García, J. M. (2008a). DiCo-CMP: Efficient cache coherency in tiled cmp architectures, *22nd Int'l Parallel and Distributed Processing Symp. (IPDPS)*.

Ros, A., Acacio, M. E. & García, J. M. (2008b). Scalable directory organization for tiled cmp architectures, *Int'l Conference on Computer Design (CDES)*, pp. 112–118.

Shah, M., Barreh, J., Brooks, J., Golla, R., Grohoski, G., Gura, N., Hetherington, R., Jordan, P., Luttrell, M., Olson, C., Saha, B., Sheahan, D., Spracklen, L. & Wynn, A. (2007). UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC, *IEEE Asian Solid-State Circuits Conference*, pp. 22–25.

Taylor, M. B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffman, H., Lee, J.-W., Johnson, P., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S. & Agarwal, A. (2002). The raw microprocessor: A computational fabric for software circuits and general purpose programs, *IEEE Micro* **22**(2): 25–35.

Thoziyoor, S., Muralimanohar, N., Ahn, J. H. & Jouppi, N. P. (2008). Cacti 5.1, *Technical Report HPL-2008-20*, HP Labs.

Wang, H., Peh, L.-S. & Malik, S. (2003). Power-driven design of router microarchitectures in on-chip networks, *36th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pp. 105–111.

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

Zhang, M. & Asanovic, K. (2005). Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors, *32nd Int'l Symp. on Computer Architecture (ISCA)*, pp. 336–345.

IntechOpen



Parallel and Distributed Computing

Edited by Alberto Ros

ISBN 978-953-307-057-5

Hard cover, 290 pages

Publisher InTech

Published online 01, January, 2010

Published in print edition January, 2010

The 14 chapters presented in this book cover a wide variety of representative works ranging from hardware design to application development. Particularly, the topics that are addressed are programmable and reconfigurable devices and systems, dependability of GPUs (General Purpose Units), network topologies, cache coherence protocols, resource allocation, scheduling algorithms, peertopeer networks, largescale network simulation, and parallel routines and algorithms. In this way, the articles included in this book constitute an excellent reference for engineers and researchers who have particular interests in each of these topics in parallel and distributed computing.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Alberto Ros, Manuel E. Acacio and Jose M. Garcia (2010). Cache Coherence Protocols for Many-Core CMPs, Parallel and Distributed Computing, Alberto Ros (Ed.), ISBN: 978-953-307-057-5, InTech, Available from: http://www.intechopen.com/books/parallel-and-distributed-computing/cache-coherence-protocols-for-many-core-cmps

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License, which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen