Dealing with Traffic-Area Trade-Off in Direct Coherence Protocols for Many-Core CMPs

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

Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30100 Murcia, Spain
{a.ros,meacacio,jmgarcia}@ditec.um.es

Abstract. In many-core CMP architectures, the cache coherence protocol is a key component since it can add requirements of area and power consumption to the final design and, therefore, it could restrict severely its scalability. Area constraints limit the use of precise sharing codes to small- or medium-scale CMPs. Power constraints make impractical to use broadcast-based protocols for large-scale CMPs.

Token-CMP and DiCo-CMP are cache coherence protocols that have been recently proposed to avoid the indirection problem of traditional directory-based protocols. However, Token-CMP is based on broadcasting requests to all tiles, while DiCo-CMP adds a precise sharing code to each cache entry. In this work, we address the traffic-area trade-off for these indirection-aware protocols. In particular, we propose and evaluate several implementations of DiCo-CMP which differ in the amount of coherence information that they must store. Our evaluation results show that our proposals entail a good traffic-area trade-off by halving the traffic requirements compared to Token-CMP and considerably reducing the area storage required by DiCo-CMP.

1 Introduction

Current chip multiprocessors (CMPs) have a relatively small number of cores, which are typically connected through a shared medium, i.e., a bus or a crossbar (e.g., the dual-core IBM Power6 [1] and the eight-core Sun T2 [2]). However, CMP architectures that integrate tens of processor cores (usually known as many-core CMPs) are expected for the near future [3], making undesirable elements that could compromise the scalability of these designs. For example, the area required by a shared network becomes impractical as the number of cores grows [4]. Therefore, tiled CMPs designed as arrays of replicated tiles connected over a point-to-point network are a scalable alternative to current small-scale CMP designs and they will help in keeping complexity manageable.

In these architectures, each tile contains at least one level of private caches which are kept coherent by using a cache coherence protocol. The cache coherence protocol is a key component since it adds requirements of area and power consumption, which can condition systems scalability. Although a great deal of attention was devoted to scalable cache coherence protocols in the last decades

Y. Dou, R. Gruber, and J. Joller (Eds.): APPT 2009, LNCS 5737, pp. 11–27, 2009.

in the context of shared-memory multiprocessors, the technological parameters and constrains entailed by many-core CMPs demand new solutions to the cache coherency problem [5]. One of these constrains is the use of unordered networks, that prevent from using the popular snooping-based cache coherence protocol.

Two traditional cache coherence protocols aimed to be used with unordered networks are *Hammer* [6], implemented in the AMD Opteron™, and *Directory* [7]. *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. Unfortunately, this storage could become prohibitive for many-core CMPs [3]. 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 [8,9,10].

On the other hand, these traditional cache coherence protocols introduce the well-known indirection problem. In both protocols, the ordering point for the requests to the same memory block is the *home* tile. Therefore, all cache misses must reach this ordering point before performing coherence actions, a fact that introduces extra latency in the critical path of cache misses. Recently, Token-CMP [11] and DiCo-CMP [12] protocols have been proposed to deal with the indirection problem. These indirection-aware protocols avoid the access to the home tile through alternative serialization mechanisms. 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. 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. These indirection-aware protocols 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 tiles, which is clearly non-scalable. Otherwise, DiCo-CMP sends requests to just one tile, but it adds a bit-vector field that keeps track of sharers to each cache entry, which does not scale with the number of cores.

The aim of this work is to address the traffic-area trade-off of indirection-aware protocols for many-core tiled CMPs. Although this trade-off has been widely studied for traditional protocols, in this work we consider protocols that try to avoid indirection. Particularly, we perform this study by relaxing the accuracy of the sharing codes used in DiCo-CMP. The other important contribution of this work is the evaluation of the state of the art in cache coherence protocols for future many-core CMPs in a common framework.

We have implemented and evaluated several cache coherence protocols based on the direct coherence concept which differ in the amount of coherence information that they store. Particularly, *DiCo-LP-1*, which only stores the identity of one sharer along with the data block, and *DiCo-NoSC*, which does not store

any coherence information along with the data caches, are the best alternatives. *DiCo-LP-1* presents a good traffic-area trade-off by requiring slightly more area than *Token-CMP* (1% for 32 cores, and same complexity order –$O(log_2 n)$–) and slightly increasing network traffic compared to *DiCo-CMP* (11% on average for 32 cores). *DiCo-NoSC* does not need to modify the structure of caches to add any extra field and, therefore, introduces less area requirements than *Token-CMP* (4% for 32 cores). However, it increases network traffic by 35% compared to *DiCo-CMP*, but still halving the traffic when compared to *Token-CMP*.

The rest of the paper is organized as follows. Section 2 discusses the cache coherence protocols that could be be used in many-core CMPs. DiCo-CMP and the implementations evaluated in this work are described in Section 3. Section 4 focus on the evaluation methodology. Section 5 shows performance results. In Section 6 we present the related work. Finally, Section 7 concludes the paper.

2 Background on Cache Coherence Protocols

This section describes the cache coherence protocols proposed in the literature aimed to be used in systems with unordered networks. We describe their implementation for a tiled CMP, in which each tile includes private L1 caches (both instruction and data caches) and a slice of the L2 cache. The L2 cache is physically distributed and logically shared among the different processing cores (L2 NUCA architecture [13]). Each memory block is assigned to a particular cache bank (or tile) which is called its *home* bank (or *home* tile). We focus on the cache coherence protocol employed for avoiding inconsistencies between data stored in the L1 caches. We also assume that caches use MOESI states, and that L1 and L2 caches are non-inclusive. 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.

2.1 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 coherence actions can be performed. Then, the request is forwarded to the corresponding tiles according to the coherence information (or it is broadcast if the protocol does not maintain any coherence information). All processors that receive the forwarded request answer to the requesting core by sending either an acknowledgment (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 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.

Hammer-CMP. *Hammer* is the cache coherence protocol used by AMD in their Opteron systems. Like snooping-based protocols, *Hammer* does not store any coherence information about the blocks held in private caches and it relies on broadcasting requests to solve cache misses. The advantage with respect to snooping-based protocols is that *Hammer* targets systems that use a point-to-point interconnection. However, the ordering point in this protocol is the home tile, a fact that introduces indirection for every cache miss. In this work we evaluate an implementation of the AMD's Hammer protocol for tiled CMPs, that we call *Hammer-CMP*. As an optimization for tiled CMPs, our implementation adds a small structure to each home tile which stores the tag of the blocks that are held in the private L1 caches. This optimization avoids off-chip accesses when the block can be obtained on-chip, and uses a small structure whose size does not increase with the number of cores.

Directory-CMP. The *directory*-based protocol that we have implemented is similar to the intra-chip coherence protocol used in Piranha [14]. This protocol avoids broadcasting requests by storing in the home tile precise information about the state of each block in the private caches. This information consists in a full-map (or bit-vector) sharing code employed for keeping track of the sharers, and a pointer identifying the owner tile, i.e., the tile that provides the data block. The bit-vector field allows the protocol to send invalidation messages just to the caches currently sharing the block. The owner field is used in a MOESI protocol to avoid forwarding requests to all sharers on read misses. In this way, requests are only forwarded to the tile that provides the block. This precise directory information allows the protocol to reduce considerably network traffic compared to *Hammer-CMP*.

2.2 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. 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, *DiCo-CMP* keeps traffic low by sending requests to only one tile. However, coherence information used by its previous implementations [12] include bit-vector sharing codes, which are not scalable in terms of area requirements.

Token-CMP. Token coherence is a framework for designing coherence protocols whose main asset is that it decouples the correctness substrate from the performance policies. Token coherence protocols avoid both the need of a totally ordered network and the introduction of indirection. They keep cache coherence by assigning T tokens to every memory block, where one of the T is the owner token. Then, a processor can read a block only if it holds at least one token for that block. On the other hand, a processor can write a block only if it holds all tokens for that block. Token coherence avoids starvation by issuing a

Table 1. Summary of cache coherence protocols

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

persistent request when a processor detects potential starvation. In this paper, we evaluate Token-CMP [11], which is a performance policy aimed to achieve low-latency cache-to-cache transfer misses. Token-CMP uses a distributed arbitration scheme for persistent requests, which are issued after a single retry to optimize the access to contended blocks.

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 to the home one. In this paper we evaluate DiCo-CMP [12], an implementation of direct coherence for CMPs. Particularly, we implement the *base* policy presented that work because it is the policy that incurs in less area and traffic requirements and it obtains similar execution times than Token-CMP.

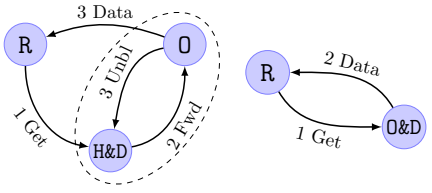
2.3 Summary

Table 1 summarizes the described protocols. 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.

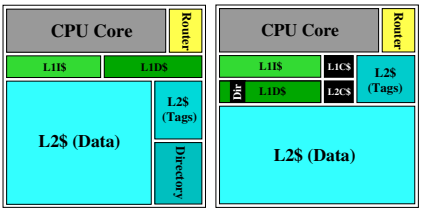
3 Traffic-Area Trade-Off in Direct Coherence Protocols

3.1 DiCo-CMP Basis and Storage Requirements

As previously discussed, traditional protocols introduce indirection in the critical path of cache misses. Figure 1(a) (left) gives an example of a cache miss suffering from indirection in Directory-CMP. When a cache miss takes place it is necessary to access the home tile to obtain the directory information and order the requests before performing coherence actions (*1 Get*). In case of a cache-to-cache transfer, the request is subsequently sent 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, other requests for the same block cannot be processed



(a) Cache-to-cache transfer in Directory-CMP (left) and DiCo-CMP (right). (R=Requester; H=Home; D=Directory; O=Owner).



(b) Organization of a tile in Directory-CMP (left) and DiCo-CMP (right). Black boxes are the elements added by DiCo-CMP.

Fig. 1. Behavior and tile design of Directory-CMP and DiCo-CMP

by the directory until it receives the unblock message (*3 Unbl*). As shown in Figure 1(a) (right), DiCo-CMP sends directly the request to the owner tile (*1 Get*). In this way, data is provided by it (*2 Data*), thus requiring only two hops to solve the miss. This is achieved by assigning the task of keeping cache coherence and ensuring ordered accesses to the owner tile. Therefore, DiCo-CMP extends the tags' part of the L1 data caches with a bit-vector field (L2 caches already include this field in Directory-CMP) to allow the protocol to keep track of sharers of a block along with its owner copy. In contrast, DiCo-CMP does not need the directory structure in the home tile that traditional directory protocols require. Additionally, by keeping together the owner block and the directory information, control messages between them are not necessary, thus saving some network traffic.

On the other hand, the drawback of DiCo-CMP is that the owner tile can change on write misses and, therefore, finding it could be difficult in some cases. Hence, DiCo-CMP needs two extra hardware structures that are used to record the identity of the owner cache of every memory block: the *L1 coherence cache* and the *L2 coherence cache*, as shown in Figure 1(b).

- *L1 coherence cache* (L1C$): The information stored in this structure is used by the requesting core to directly send local requests to the owner tile. Therefore, this structure is located close to each processor's core. Although DiCo-CMP can update this information in several ways, we consider in this work the *base* policy presented in [12], in which this information is updated by using the coherence messages sent by the protocol, i.e., invalidation and data messages.
- *L2 coherence cache* (L2C$): Since the owner tile can change on write misses, this structure is responsible for tracking the owner cache for each block allocated in any L1 cache. The L2C$ replaces the directory structure required by Directory-CMP and it is accessed each time a request fails to locate the owner tile. Therefore, this information is updated through control messages whenever the owner tile changes.

3.2 DiCo-CMP Cache Coherence Protocol

When a processor issues a request that misses in its private L1 cache, the request is directly sent 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 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.

If the request is received by a tile that is not the current owner of the block, it is simply re-sent to the home tile, where the L2C$ is accessed. Then, in case of a hit in the L2C$, the request is sent to the current owner tile. In absence of race conditions the request will reach the owner tile. If there is a miss in the L2C$ the request is solved by providing the block from main memory, where, in this case, a valid copy of the block resides. In this case, a new entry pointing to the current L1 owner tile has to be allocated in the L2C$.

If the request reaches the owner tile, the miss can be immediately solved. If the owner is the home tile all requests (reads and writes) are solved by deallocating the block from the home tile and allocating it in the L1 cache of the requester. Again, the identity of the new owner tile is stored in the L2C$.

When the owner is the L1 cache, read misses are completed by sending a copy of the block to the requester and adding it to the sharing code field. Write misses are solved by sending invalidation messages to all the tiles sharing the block and by sending the data block to the requester. Acknowledgement messages are collected at the requesting cache as in all protocols evaluated in this work.

Finally, since the L2C$ must store up-to-date information regarding the owner tile, every time that the owner tile changes, a control message is sent to the L2C$ indicating the identity of the new owner. These messages must be processed by the L2C$ in the very same order in which they were generated. Otherwise, the L2C$ could fail to store the identity of the current owner. The order is guaranteed by sending an acknowledgement from the L2C$ to the new owner. Until this message is not received by the new owner, it cannot give the ownership to another tile. Note that these two control messages are not in the critical path of the current miss.

3.3 Reducing Storage Requirements for DiCo-CMP

DiCo-CMP needs two structures that keep the identity of the tile where the owner copy of the block resides. These two structures does 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 necessary to solve cache misses in DiCo-CMP, since ensures that the tile that keeps coherence for each block can always be found. On the other hand, the L1C$ is necessary to avoid indirection in cache misses and, therefore, it is essential to obtain good performance.

Apart from these structures, DiCo-CMP also adds a full-map sharing code to each cache entry. Since the memory overhead of this field can become prohibitive

for many-core CMPs, we study some alternatives that differ in the amount of coherence information stored. These alternatives have at least area requirements of order $O(log_2 n)$, due to the L1C$ and the L2C$. The particular compressed sharing code employed only impacts on the number of invalidations sent for write misses, because in DiCo-CMP cache misses are solved from the owner tile and, therefore, read misses are never broadcast. Next, we comment on the alternatives evaluated in this work.

DiCo-CV-K is a DiCo-CMP protocol that reduces the size of the sharing code field by using a *coarse vector* [10]. 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, if one of the tiles in the group holds the block, all tiles belonging to that group will receive an invalidation message. Particularly, we study two configurations using a coarse vector sharing code with values for K of 2 and 4. 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 [9]. In this scheme, each entry has a limited number of P 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 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. However, this kind of misses is not very frequent since the sharing degree of applications is usually low [7]. The overhead of this sharing code is $O(P \times log_2 n)$. In particular, evaluate this protocol with a value for P of 1.

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 *DiCo-CV-K* or *DiCo-LP-P*. However, it incurs in less traffic than *Hammer-CMP* and *Token-CMP*. *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 independent control message. On the other hand, although *Token-CMP* avoids unnecessary acknowledgements, it also relies on broadcasting requests for all cache misses.

4 Simulation Environment

We perform the evaluation using the full-system simulator Virtutech Simics [15] extended with Multifacet GEMS 1.3 [16], 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 [17], a detailed interconnection network simulator. We simulate CMP systems with 16 and 32 tiles to show that our proposals

Table 2. System parameters

GEMS Parameters		SICOSYS Parameters	
Processor frequency	4 GHz	Network frequency	2 GHz
Cache hierarchy	Non-inclusive	Topology	4x4 & 8x4 Mesh
Cache block size	64 bytes	Switching technique	Wormhole, Multicast
Split L1 I & D caches	128KB, 4 ways, 4 hit cycles	Routing technique	Deterministic X-Y
Shared unified L2 cache	1MB/tile, 4 ways, 7 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	200 cycles	Link bandwidth	1 flit/cycle

scale with the number of cores. Table 2 shows the values of the main parameters used for the evaluation, where cache latencies have been calculated using the CACTI 5.3 tool [18] 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 44 bits, like in the SUN UltraSPARC-III architecture [19].

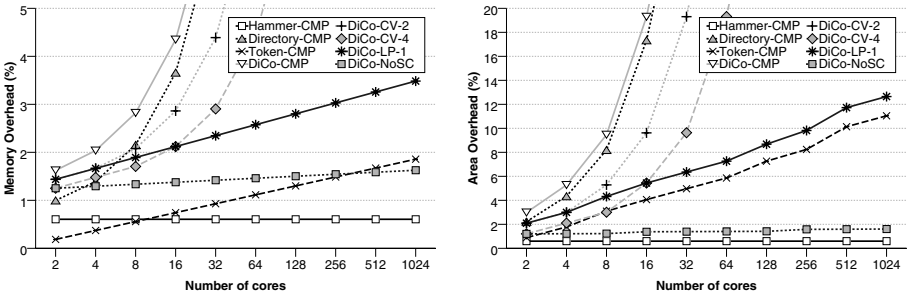
The ten applications used in our simulations cover a variety of computation and communication patterns. *Barnes* (8192 bodies, 4 time steps), *FFT* (256K points), *Ocean* (258x258 ocean), *Radix* (1M keys, 1024 radix), *Raytrace* (teapot), *Volrend* (head) and *Water-Nsq* (512 molecules, 4 time steps) are scientific applications from the SPLASH-2 benchmark suite [20]. *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 [21]. 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.

5 Evaluation Results

5.1 Impact on Area Overhead

First, we compare the memory overhead introduced by coherence information for all the protocols considered in this work. Although some protocols can entail extra overhead as a consequence of the additional mechanisms that they demand (e.g., timeouts for reissuing requests in *Token-CMP*), we only consider the amount of memory required to keep coherence information. Figure 2 shows the storage overhead introduced by these protocols in terms of both number of bits and estimated area, varying the number of cores from 2 to 1024.

Although the original *Hammer* protocol does not require 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 private L1 caches but not in the shared L2 cache. However, this structure introduces a slight overhead which however keeps constant when the number of cores increases.



(a) Overhead in terms of bits. (b) Overhead in terms of area.

Fig. 2. Overhead introduced by the coherence protocols evaluated in this work

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 by using a bit-vector, the number of required bits is n and, consequently, the width of each 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, which requires $log_2(n + 1)$ bits (the owner-token bit and non-owner token count). These additional bits are stored in the tags' part of both cache levels. Therefore, *Token-CMP* has an acceptable scalability in terms of area.

DiCo-CMP stores directory information for owner blocks stored in any L1 or L2 cache. Therefore, a full-map sharing code is added to each cache line. 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. Hence, this is the protocol with more area requirements.

In this work, we propose to reduce this overhead by introducing compressed sharing codes in DiCo-CMP. *DiCo-CV-2* and *DiCo-CV-4* save storage compared to *DiCo-CMP* but they are still non-scalable. In contrast, *DiCo-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)$–. 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 presence of the L1 and L2 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 major overhead when the area of the structures is considered.

5.2 Impact on Network Traffic

Figure 3 compares the network traffic generated by the protocols discussed previously for the 16-core and the 32-core configurations. Each bar plots the number of bytes transmitted through the interconnection network normalized with respect to *Hammer-CMP*.

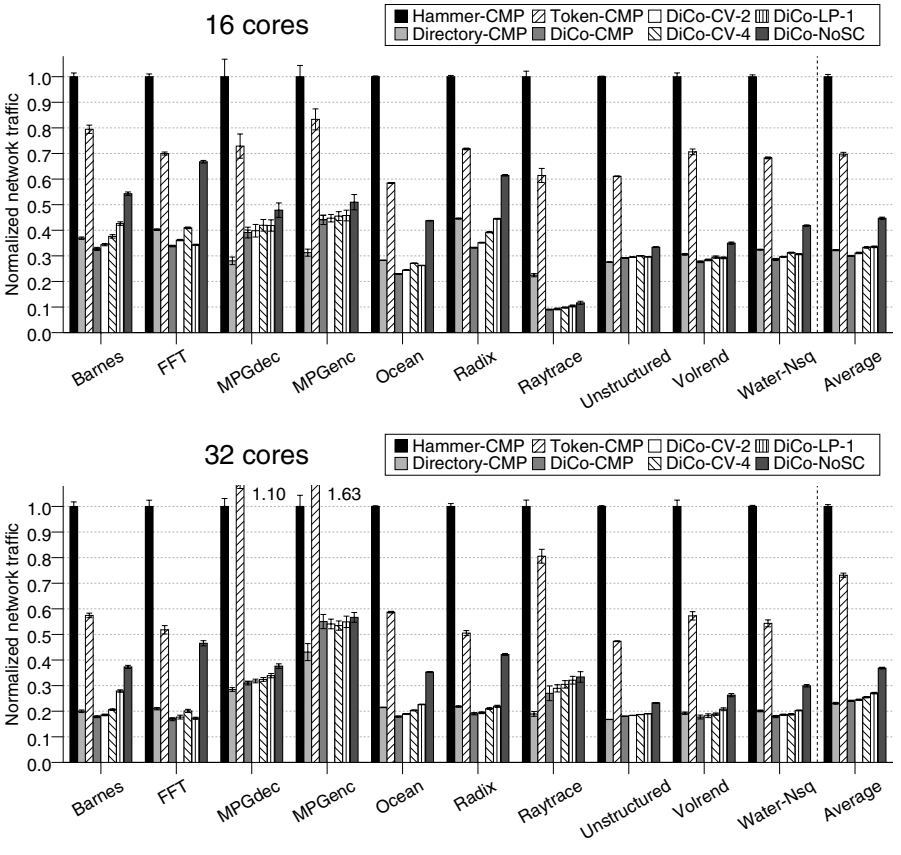


Fig. 3. Normalized network traffic

As expected, *Hammer-CMP* introduces 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 bit-vector that filters unnecessary invalidations. *Token-CMP* generates more network traffic than *Directory-CMP*, because it relies on broadcast, 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). However, for some applications, like *MPGdec* and *MPGenc*, *Token-CMP* generate more traffic than *Hammer-CMP* for the 32-core configuration. This increase is due to two main factors. First, in Hammer-CMP, read misses that found the data block in the L2 cache do not broadcast requests whereas Token-CMP always needs to broadcast read requests. Second, the high contention found in these applications increases the amount of reissued persistent requests in *Token-CMP*. Finally, we can also observe that *DiCo-CMP* has similar traffic requirements than *Directory-CMP*.

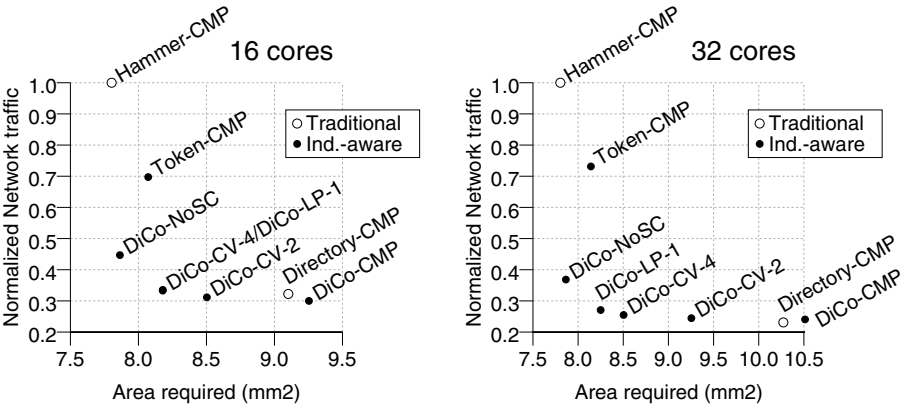


Fig. 4. Traffic-area trade-off

In general, we can see that compressed sharing codes increase network traffic. However, the increase in traffic is admissible. Even *DiCo-NoSC*, which does not keep track of sharers, generates an acceptable amount of network traffic (36% less traffic than *Token-CMP* for 16 cores and 50% for 32 cores). As previously commented, *DiCo-NoSC* stores in the L2C$ a pointer to the owner block which prevent read misses of broadcasting requests, as happens in Hammer-CMP and Token-CMP.

5.3 Traffic-Area Trade-Off

Figure 4 shows the traffic-area trade-off for all the protocols evaluated in this work. The figure also differentiates between traditional and indirection-aware protocols. We can see that, in general, the base protocols aimed to be used with tiled CMPs do not have a good traffic-area trade-off: both *Hammer-CMP* and *Token-CMP* are constrained by traffic while both *Directory-CMP* and *DiCo-CMP* are constrained by area.

However, the use of different compressed sharing codes for DiCo-CMP can lead to a good compromise between network traffic and area requirements. The *DiCo-CV* approaches have low traffic overhead but the area requirements considerably increase with the number of cores. Both *DiCo-LP-1* and *DiCo-NoSC* are very close to an ideal protocol with the best of the base protocols. The difference is that *DiCo-LP-1* is more efficient in terms of generated traffic while *DiCo-NoSC* is more efficient in terms of area requirements. Particularly, *DiCo-LP-1* requires slightly more area than *Token-CMP* (1% for 32 cores, and same complexity order –$O(log_2 n)$–) and slightly increases network traffic compared to *DiCo-CMP* (11% on average for 32 cores). On the other hand, *DiCo-NoSC* does not need to modify the structure of caches to add any extra field and, therefore, introduces less area requirements than *Token-CMP* (4% for 32 cores), but with the same complexity

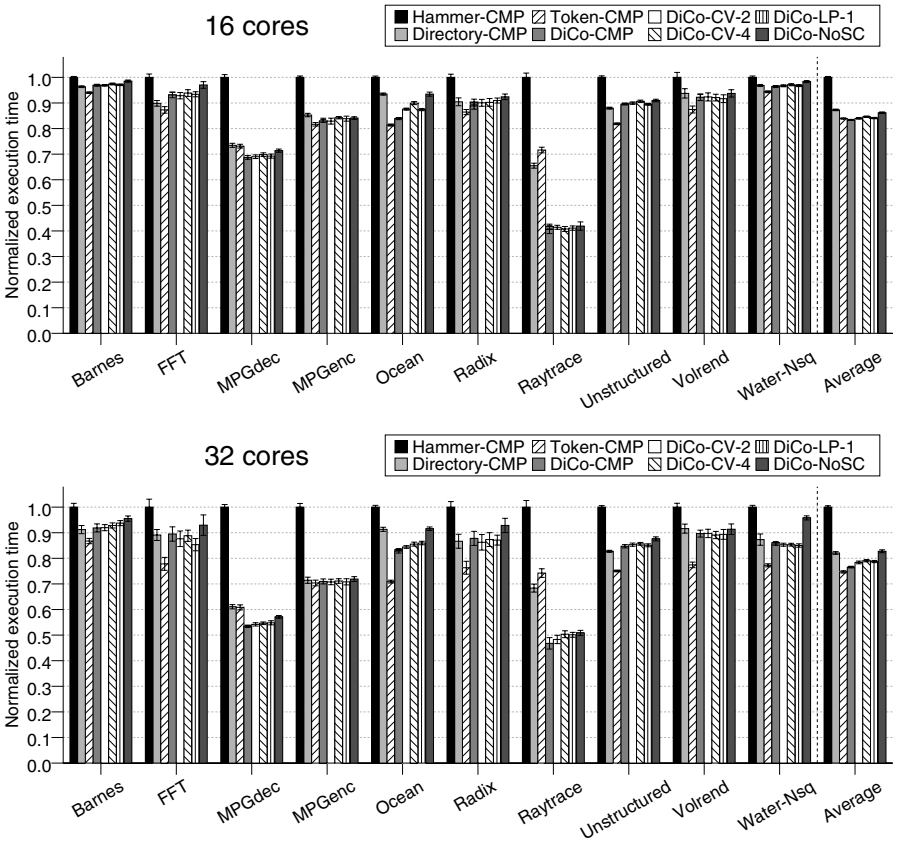


Fig. 5. Normalized execution times

order –$O(log_2 n)$– . However, it increases network traffic by 35% compared to *DiCo-CMP*, but still halving the traffic when compared to *Token-CMP*.

5.4 Impact on Execution Time

Figure 5 plots the average execution times for the applications evaluated in this work 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 network traffic. Moreover, on each miss *Hammer-CMP* must wait for all the acknowledgement messages before the requested block can be accessed. On the contrary, in *Directory-CMP* only write misses must wait for acknowledgements.

On the other hand, indirection-aware protocols reduce average execution time when compared to traditional protocols. Particularly, *Token-CMP* obtains average improvements of 16% compared to *Hammer-CMP* and 4% compared to

Directory-CMP for 16 cores. Similar improvements are obtained with *DiCo-CMP*. For 32 cores, the average improvements of indirection-aware protocols becomes more significant. On the other hand, when *DiCo-CMP* employs compressed sharing codes, the execution time increases. However, it remains close to *DiCo-CMP*, except for *DiCo-NoSC* mainly when a 32-core CMP is considered.

6 Related Work

DiCo-CMP was recently proposed by Ros *et al.* [12] to avoid the indirection of traditional coherence protocols in tiled CMPs. This protocol adds a bit-vector sharing code to each cache entry (particularly, in the tags part), thus compromising scalability. In this work, we propose and evaluate several implementations of DiCo-CMP that use compressed sharing codes to scale gracefully with the number of cores and do not require a prohibitive amount of network traffic.

Snoopy protocols do not introduce indirection because they are based on a totally-ordered interconnection network. Unfortunately, these interconnection networks are not scalable. Some proposals have focused on using snoopy protocols with arbitrary network topologies. Martin. *et al.* [22] present a technique that allows SMPs to utilize unordered networks (with some modifications to support snooping). Bandwidth Adaptive Snooping Hybrid (BASH) [23] is an hybrid coherence protocol that dynamically decides whether to act like snoopy protocols (broadcast) or directory protocols (unicast) depending on the available bandwidth. In contrast, the protocol presented in this work does not changes dynamically, but only broadcast requests for a small number of cache misses, thus obtaining network traffic reductions.

Cheng *et al.* [24] adapt already existing coherence protocols for reducing energy consumption and execution time in CMPs with heterogeneous networks. In particular, they assume a heterogeneous network comprised of several sets of wires, each one with different latency, bandwidth, and energy characteristics, and propose to send each coherence message through a particular set of wires depending on its latency and bandwidth requirements. Our proposals are orthogonal to this work and the ideas presented in [24] could also be applied to direct coherence protocols.

Martin *et al.* propose to use destination-set prediction to reduce the bandwidth required by a snoopy protocol [25]. Differently from our proposals, this approach is based on a totally-ordered interconnect, which is not suitable for large-scale tiled CMPs. Regarding indirection avoidance, Cheng *et al.* propose to convert 3-hop read misses into 2-hop read misses for memory blocks that exhibit the producer-consumer sharing pattern [26] by using extra hardware to detect when a block is being accessed according to this pattern. In contrast, direct coherence obtains 2-hops misses for read, write and upgrade misses without taking into account sharing patterns.

7 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 power constrains 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.

This work addresses the traffic-area trade-off of indirection-aware cache coherence protocols through several implementations of direct coherence for CMPs. We evaluate several cache coherence protocols that differ in the amount of coherence information that they store. Particularly, *DiCo-LP-1*, which only stores the identity of one sharer along with the data block, and *DiCo-NoSC*, which does not store any coherence information in the data caches, are the alternatives that achieve a best compromise between traffic and area. Note that both approaches include the coherence caches required by direct coherence protocols. *DiCo-LP-1* presents a good trade-off by requiring slightly more area than *Token-CMP* (1% for 32 cores and same order –$O(log_2 n)$–) and slightly increasing network traffic compared to *DiCo-CMP* (11% for 32 cores). *DiCo-NoSC* does not need to modify the structure of caches and, therefore, has less area requirements than *Token-CMP* (4% for 32 cores), but with the same complexity order –$O(log_2 n)$–. However, it increases network traffic by 35% compared to *DiCo-CMP*, but still halving the traffic when compared to *Token-CMP*. Finally, *DiCo-LP-1* improves execution time compared to *DiCo-NoSC* due to reductions in network traffic. Finally, we believe that both alternatives can be considered for many-core tiled CMPs depending on the particular system constraints.

References

1. Le, H.Q., et al.: IBM POWER6 microarchitecture. IBM Journal of Research and Development 51(6), 639–662 (2007)
2. Shah, M., et al.: UltraSPARC T2: A highly-threaded, power-efficient, SPARC SOC. In: IEEE Asian Solid-State Circuits Conference, November 2007, pp. 22–25 (2007)
3. Azimi, M., et al.: Integration challenges and tradeoffs for tera-scale architectures. Intel. Technology Journal 11(3), 173–184 (2007)
4. Kumar, R., Zyuban, V., Tullsen, D.M.: Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In: 32nd Int'l. Symp. on Computer Architecture (ISCA), June 2005, pp. 408–419 (2005)
5. Bosschere, K.D., et al.: High-performance embedded architecture and compilation roadmap. Transactions on HiPEAC I, 5–29 (January 2007)
6. Owner, J.M., Hummel, M.D., Meyer, D.R., Keller, J.B.: System and method of maintaining coherency in a distributed communication system. U.S. Patent 7069361 (June 2006)

7. Culler, D.E., Singh, J.P., Gupta, A.: Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, Inc., San Francisco (1999)
8. Agarwal, A., Simoni, R., Hennessy, J.L., Horowitz, M.: An evaluation of directory schemes for cache coherence. In: 15th Int'l. Symp. on Computer Architecture (ISCA), May 1988, pp. 280–289 (1988)
9. Chaiken, D., Kubiatowicz, J., Agarwal, A.: LimitLESS directories: A scalable cache coherence scheme. In: 4th Int. Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS), April 1991, pp. 224–234 (1991)
10. Gupta, A., Weber, W.D., Mowry, T.C.: Reducing memory traffic requirements for scalable directory-based cache coherence schemes. In: Int'l. Conference on Parallel Processing (ICPP), August 1990, pp. 312–321 (1990)
11. Marty, M.R., Bingham, J., Hill, M.D., Hu, A., Martin, M.M., Wood, D.A.: Improving multiple-cmp systems using token coherence. In: 11th Int'l. Symp. on High-Performance Computer Architecture (HPCA), February 2005, pp. 328–339 (2005)
12. Ros, A., Acacio, M.E., García, J.M.: DiCo-CMP: Efficient cache coherency in tiled cmp architectures. In: 22nd Int'l. Parallel and Distributed Processing Symp. (IPDPS) (April 2008)
13. Kim, C., Burger, D., Keckler, S.W.: An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In: 10th Int. Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS), October 2002, pp. 211–222 (2002)
14. Barroso, L.A., et al.: Piranha: A scalable architecture based on single-chip multiprocessing. In: 27th Int'l. Symp. on Computer Architecture (ISCA), June 2000, pp. 12–14 (2000)
15. Magnusson, P.S., et al.: Simics: A full system simulation platform. IEEE Computer 35(2), 50–58 (2002)
16. Martin, M.M., et al.: Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. Computer Architecture News 33(4), 92–99 (2005)
17. Puente, V., Gregorio, J.A., Beivide, R.: SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems. In: 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, January 2002, pp. 15–22 (2002)
18. Thoziyoor, S., Muralimanohar, N., Ahn, J.H., Jouppi, N.P.: Cacti 5.1. Technical Report HPL-2008-20, HP Labs (April 2008)
19. Horel, T., Lauterbach, G.: UltraSPARC-III: Designing third-generation 64-bit performance. IEEE Micro. 19(3), 73–85 (1999)
20. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: Characterization and methodological considerations. In: 22nd Int'l. Symp. on Computer Architecture (ISCA), June 1995, pp. 24–36 (1995)
21. Li, M.L., Sasanka, R., Adve, S.V., Chen, Y.K., Debes, E.: The ALPBench benchmark suite for complex multimedia applications. In: Int'l. Symp. on Workload Characterization, October 2005, pp. 34–45 (2005)
22. Martin, M.M., et al.: Timestamp snooping: An approach for extending SMPs. In: 9th Int. Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS), November 2000, pp. 25–36 (2000)
23. Martin, M.M., Sorin, D.J., Hill, M.D., Wood, D.A.: Bandwidth adaptive snooping. In: 8th Int'l. Symp. on High-Performance Computer Architecture (HPCA), January 2002, pp. 251–262 (2002)

24. Cheng, L., Muralimanohar, N., Ramani, K., Balasubramonian, R., Carter, J.B.: Interconnect-aware coherence protocols for chip multiprocessors. In: 33rd Int'l. Symp. on Computer Architecture (ISCA), June 2006, pp. 339–351 (2006)
25. Martin, M.M., Harper, P.J., Sorin, D.J., Hill, M.D., Wood, D.A.: Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors. In: 30th Int'l. Symp. on Computer Architecture (ISCA), June 2003, pp. 206–217 (2003)
26. Cheng, L., Carter, J.B., Dai, D.: An adaptive cache coherence protocol optimized for producer-consumer sharing. In: 13th Int'l. Symp. on High-Performance Computer Architecture (HPCA), February 2007, pp. 328–339 (2007)