Efficient Cache Coherence Protocol in Tiled Chip Multiprocessors

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

Abstract— **Although directory-based cache coherence protocols are the best choice when designing large-scale chip multiprocessors (CMPs), they introduce indirection to access directory information, which negatively impacts performance. In this work, we present DiCo-CMP, a cache coherence protocol aimed at avoiding indirection to the directory information. In DiCo-CMP, the role of storing up-to-date sharing information and ensuring totally ordered accesses for every memory block is assigned to the cache that must provide the block on a miss. Therefore, DiCo-CMP reduces the miss latency compared to a directory protocol by sending coherence messages directly from the requesting caches to those that must observe them, and reduces the network traffic compared to broadcast-based protocols by sending just one request message for each miss. Using an extended version of GEMS simulator we show that DiCo-CMP achieves improvements in execution time of up to 8% on average over a directory protocol, and reductions in terms of network traffic of up to 42% on average compared to Token-CMP.**

Keywords— **Tiled CMPs, cache coherence protocols, DiCo-CMP, direct coherence, indirection.**

I. Introduction

TILED CMP architectures have recently emerged as a scalable alternative to current CMP designs, and future CMPs will be probably designed as arrays of replicated tiles connected over a switched direct network [1], [2].

Directory-based cache coherence protocols have been typically employed in systems with point-to-point unordered networks (as tiled CMPs are). Unfortunately, these protocols introduce indirection to obtain coherence information from the directory (commonly on chip as a directory cache), thus increasing cache miss latencies. An alternative approach that avoids indirection is Token-CMP [3]. Token-CMP is based on broadcasting requests to all last-level private caches. In this way, caches can directly provide data when they receive a request (no indirection occurs). Unfortunately, the use of broadcasting increases network traffic and, therefore, power consumption in the interconnection network, which has been previously reported to constitute a significant fraction (approaching 50% in some cases) of the overall chip power [4].

In this work, we present DiCo-CMP, a cache coherence protocol for tiled CMP architectures that meets the advantages of directory and Token-CMP protocols and avoids their problems. DiCo-CMP assigns the role of storing up-to-date sharing information and ensuring totally ordered accesses for every memory block to one of the caches that actually shares the block, particularly the one that provides the block on a miss (the owner cache in a MOESI protocol). Indirection is avoided by directly sending the requests to the owner cache instead of to the directory structure kept in the home tile in directory protocols. In our proposal, the identity of the owner caches is recorded in a small structure associated to every core called the L1 coherence cache. Since the owner cache changes on write misses, another structure called the L2 coherence cache keeps up-to-date information about the identity of the owner cache. This L2 coherence cache is accessed each time a request fails to locate the owner cache.

In this way, DiCo-CMP reduces the latency of cache misses compared to a directory protocol by sending coherence messages directly from the requesting caches to those that must observe them, and reduces network traffic compared to Token-CMP by sending just one request message on every cache miss. Detailed simulations using GEMS and several scientific applications show that DiCo-CMP achieves improvements in total execution time of 8% on average over a directory protocol and of 3% on average over Token-CMP. Moreover, our proposal obtains reductions in terms of network traffic (and, consequently, in power consumption) compared to Token-CMP of 28%.

The rest of the paper is organized as follows. In Section II we present a review of the related work. DiCo-CMP is described in Section III. Section IV introduces the methodology employed in the evaluation. Section V shows the performance results obtained by our proposal. And finally, Section VI concludes the paper.

II. Related Work and Background

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

Acacio *et al.* propose to avoid the indirection for cache-to-cache transfer misses [5] and upgrade misses [6] 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 miss-predictions. In contrast, our proposal is applicable to all types of misses and just the identity of the owner cache is predicted. We avoid predicting the current holders by storing the up-to-date directory information in the owner cache.

Departamento de Ingeniería y Tecnología de Computadores. Universidad de Murcia. 30100 Espinardo, Murcia. e-mail: {a.ros,meacacio,jmgarcia}@ditec.um.es

Martin *et al.* propose to use destination-set prediction to reduce the bandwidth required by a snoopy protocol [7]. Differently from DiCo-CMP, this proposal is based on a totally-ordered interconnect, which does not scale with the number of nodes. Destination-set prediction is also used by Token-M with unordered networks [8]. However, on miss-predictions, requests are solved by resorting on broadcasting after a time-out period. Differently, in DiCo-CMP miss-predictions are resent immediately to the owner cache, thus reducing latency and network traffic.

More recently, Cheng *et al.* have proposed converting 3-hop read misses into 2-hop read misses for memory blocks that exhibit the producer-consumer sharing pattern [9] by using extra hardware to detect when a block is being accessed according to this pattern. In contrast, our proposal obtains 2-hops misses for read, write and upgrade misses without taking into account sharing patterns.

A. Directory-CMP

Directory-based coherence protocols have been widely used in shared-memory multiprocessors. Now, several Chip Multiprocessors, as Piranha [10], also use directory protocols to keep cache coherency. In this paper, we compare our proposal against a directory protocol similar to the intra-chip coherence protocol used in Piranha, which is based on MOESI states in the caches. In this implementation, on-chip directory caches are used for accelerating the accesses to directory information for blocks stored in the L1 caches. Moreover, the protocol implements a migratory-sharing optimization [11], in which a cache holding a modified cache block invalidates its copy when responding with the block, thus granting the requesting processor read/write access to the block (even when only read permission was requested).

B. Token-CMP

Token coherence [12] is a framework for designing coherence protocols whose main asset is that it decouples the correctness substrate from several different performance policies. Token coherence protocols can avoid both the need of a totally ordered network and the introduction of additional indirection caused by the directory in the common case of cache-to-cache transfers. Token coherence protocols keep cache coherence by assigning T tokens to every memory block, where one of the T is the owner token. Then, a processor can read a block only if it holds at least one token for that block and has valid data. On the other hand, a processor can write a block only if it holds all T tokens for that block and has valid data. In this paper, we compare our coherence protocol against Token-CMP [3], which targets CMP systems, and uses a distributed arbitration scheme for persistent requests to optimize the access to contended blocks.

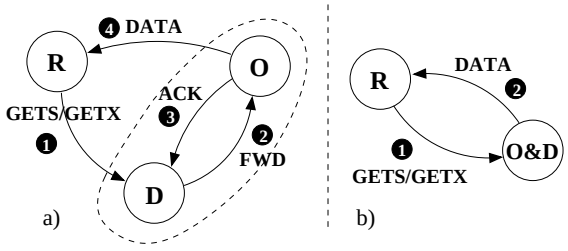

Fig. 1. a) Cache-to-cache transfer in a directory protocol. b) Cache-to-cache transfer in DiCo-CMP. (R=Requester; D=Directory; O=Owner).

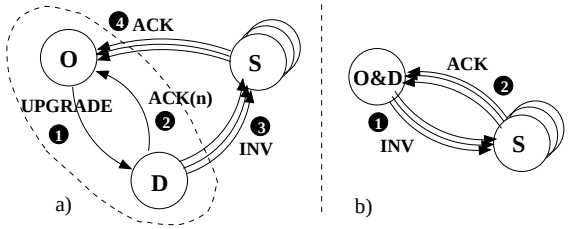

Fig. 2. a) Upgrade in a directory protocol. b) Upgrade in DiCo-CMP. (O=Owner; D=Directory; S=Sharers).

III. The DiCo-CMP Coherence Protocol

As shown in Figure 1.a, in directory-based protocols it is necessary to obtain the directory information before any coherence action can be performed (1). This information is obtained either from the L2 cache or from a directory cache (commonly on chip). Moreover, the access to the directory information serializes the requests to the same block issued by different processors. In case of a cache-to-cache transfer, the request is subsequently sent to the owner cache where the miss is solved (2). It can be observed that first, the miss is solved in three hops, and second, another request for the same block cannot be processed by the directory until it receives the acknowledgement from the owner cache (3). In contrast, DiCo-CMP (Figure 1.b) assigns the role of storing up-to-date sharing information and ensuring totally ordered accesses for every memory block to the owner cache. Indirection is avoided by directly sending the request to the owner cache. Moreover, by keeping together the owner and the directory information the owner cache does not need to receive any acknowledgement to process the next request to the same block, thus saving some control messages and reducing the latency of requests.

Another example of the advantages of DiCo-CMP is shown in Figure 2. This diagram represents an upgrade that takes place in the owner node, which happens frequently in common applications. In a directory protocol, upgrades are solved sending the request to the directory (1), which replies with the number of acknowledgements that must be received before the block can be modified (2), and sends invalidations (3). In DiCo-CMP only invalidations (1) and acknowledgements (2) are required, thus solving the miss with just two hops in the critical path.

DiCo-CMP extends the tags' part of the L1 and L2 caches with a new field used to keep the identity of the sharers for blocks in owned state. Additionally, DiCo-CMP needs two extra hardware structures that keeps a pointer which identifies the owner cache:

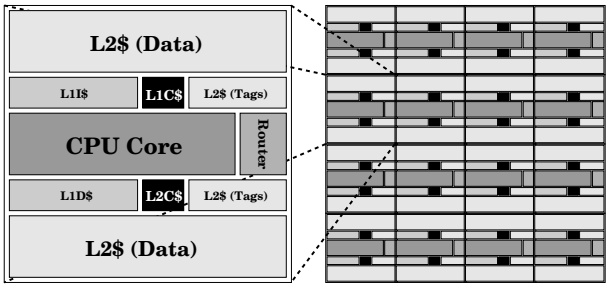

Fig. 3. Organization of a tile (black boxes are the elements added by DiCo-CMP) and a 4×4 tiled CMP.

- *L1 coherence cache* (L1C$): The pointer stored in this structure is used to directly send local requests to the owner cache, thus avoiding indirection. Therefore, this structure is located close to each processor's core. Our cache coherence protocol can update this information in several ways based on network usage (see Section III-C).
- *L2 coherence cache* (L2C$): Since the owner cache can change on write misses, this structure must keep the identity of the current owner cache for each block allocated in any L1 cache. Therefore, this information must be updated whenever the owner cache changes through control messages. This structure is accessed each time a request fails to locate the owner cache.

A. Architecture of Tiled CMPs

The tiled CMP architecture assumed in this work consists of a number of replicated *tiles* connected over a switched direct network. Each tile contains a processing core with primary caches (both instruction and data caches), a slice of the L2 cache, and a connection to the on-chip network. The L2 cache is shared among the different processing cores, but it is physically distributed among them. Therefore, some accesses to the L2 cache will be sent to the local slice while the rest will be serviced by remote slices (L2 NUCA architecture [13]). Moreover, the L1 and L2 caches are non-inclusive to exploit the total available cache capacity on chip. Figure 3 shows the organization of a tile (left) and a 16-tile CMP (right).

The protocols evaluated in this work follow this design. However, each tile in DiCo-CMP adds the two structures introduced in the previous section: the L1 and L2 coherence caches (see black boxes in Figure 3, left). Moreover, to keep the directory information within the owner cache it is necessary to add a new field in the tags' part of the L1 caches. In contrast, DiCo-CMP does not need to keep directory information in on-chip directory caches.

B. Description of the cache coherence protocol

B.1 Requesting processor

When a processor issues a request that misses in its private L1 cache, the request is directly sent to the owner cache in order to avoid indirection. The identity of the potential owner cache is obtained from the L1C$, which is accessed at the time that the cache miss in detected. If there is a hit in the L1C$, the request is sent to the owner cache. Otherwise, the request is sent to the L2 cache where the L2C$ will be accessed to find out the identity of the current owner cache.

B.2 Request received by a non-owner cache

When a request arrives at a cache that is not the owner of the block, the request is simply re-sent to the owner cache. If the cache that receives the request is an L1 cache, it re-sends the request to the L2 cache. On the other hand, if it is the L2 cache and there is a hit in the L2C$, the request is sent to the current owner cache. Finally, if there is a miss in the L2C$ and the L2 cache 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. The requested block is allocated in the requesting L1 cache, which gets the ownership of the block, but not in the L2 cache (as occurs in the directory protocol). Moreover, it is necessary to allocate a new entry in the L2C$ pointing to the current L1 owner cache.

B.3 Request received by the owner cache

If the owner is the L2 cache 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. Moreover, the identity of the new owner cache must be stored in the L2C$.

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

For write misses, the owner cache sends invalidation messages to all the caches that hold a copy of the block, and then, data to the requester. Acknowledgement messages are collected at the requesting cache. Upgrade misses that take place in the owner cache just need to send invalidations and receive acknowledgements (two hops in the critical path).

Finally, since the L2C$ must have up-to-date information regarding the location of the owner cache, every time that the owner cache changes, it is also sent a control message to the L2C$ indicating the identity of the new owner. These messages should be processed by the L2C$ in the very same order in which they were generated. Otherwise, the L2C$ could fail to store the identity of the current owner. To ensure this order, once the L2C$ processes the message reporting an ownership change from the old owner, it sends a confirmation response to the new owner. Until this confirmation message is not received by the new owner, it could use the block (if already received), but cannot give the ownership to another cache. The cache miss is considered finalized once this confirmation response (besides the message with data) has been received.

C. Updating the L1 coherence cache

DiCo-CMP uses the L1C$ to avoid indirection by keeping a pointer that identifies the owner cache. Several policies can be used to update the value of this pointer. A first approach is to store the last processor that invalidated the previous copy of the block from cache (the last processor that wrote the block). When a block is invalidated from an L1 cache, the L1C$ stores the identity of the processor that requests the block. We call this policy as the *base* policy and it does not imply extra messages. Another approach, with higher network usage, sends some hints to update the L1C$s whenever the owner changes. In this approach, each owner cache keeps a set of frequent sharers (i.e. all the cores that have requested the block). When the owner changes all the frequent sharers are informed by means of a control message, and the list of frequent sharers is transferred to the new owner. We name this policy as the *hints* policy. Finally, to find out the potential of our proposal, we have also implemented an *oracle* policy in which the L1C$ always provides the identity of the current owner cache on every cache miss.

D. Preventing starvation

In DiCo-CMP each write miss implies that the cache that keeps cache coherence for a particular block changes, and therefore, some cache misses can take some extra time to find out this cache. If a memory block is repeatedly written by several processors, a request could take some time to find the owner cache ready to process the request. Hence, some processors could be solving their requests while other requests are starved.

DiCo-CMP avoids starvation by using a simple mechanism. In particular, each time that a request must be re-sent to the L2 cache, a counter into the request message is increased. The request is considered starved when this counter reaches a certain value (i.e. two accesses to the L2 cache). When the L2 cache detects a starved request, it re-sends the request to the owner cache, but records the address of the block. If the starved request reaches the current owner cache, the miss is solved, and then the L2 cache is notified, ending the starvation situation if there is not any additional starved request for such address. Otherwise, when the message informing about the change of the ownership arrives at the L2 cache, the block is detected as suffering from starvation and the acknowledgement message required on every ownership change is not sent. This ensures that the identity of the owner does not change until the starved request completes.

IV. Simulation Environment

We evaluate our proposal with full-system simulation using Virtutech Simics extended with Multifacet GEMS. GEMS provides a detailed memory system timing model which accounts for all protocol messages and state transitions. In order to model precisely the interconnection network, and thus, obtain more accurate results, we have replaced the original (not very detailed) network simulator offered by GEMS with the SICOSYS detailed interconnection network simulator. The simulated system is a tiled CMP organized as a 4×4 array of replicated tiles, as described in Section III-A. Table I shows the values of the main parameters of the architectures evaluated in this work.

TABLE I
System parameters.

4x4 tiled CMP	
In-order Processor Parameters	
Processor speed	2 GHz
Max. fetch/retire rate	4
Memory Parameters	
Cache block size	64 bytes
Split L1 I & D caches	128KB, 4-way
L1 cache hit time	4 cycles
Shared unified L2 cache	16MB (1MB/tile), 4-way
L2 cache hit time	6 + 9 cycles (tag + data)
L1 Coherence cache	1 KB, 4-way, 2 hit cycles
L2 Coherence cache	1 KB, 4-way, 2 hit cycles
Memory access time	160 cycles
Multicast Network Parameters	
Topology	4x4 Mesh
Switching technique	Wormhole
Link latency (one hop)	4 cycles
Routing time	2 cycles
Flit size	4 bytes
Link bandwidth	1 flit/cycle

We have implemented the three policies described in Section III-C: *base*, *hints* and *oracle*. We compare our proposal against the Token-CMP protocol described in [3] and a directory protocol similar to the intra-chip coherence protocol used in Piranha [10].

The eight scientific applications used in our simulations cover a variety of computation and communication patterns. Barnes (8192 bodies, 4 time steps), Cholesky (tk16.O), FFT (256K complex doubles), Ocean (258x258 ocean), Radix (1M keys, 1024 radix), Raytrace (teapot) and Water-NSQ (512 molecules, 4 time steps) are from the SPLASH-2 benchmark suite [14]. Unstructured (Mesh.2K, 5 time steps) is a computational fluid dynamics application. We account for the variability in multithreaded workloads by doing multiple simulation runs for each benchmark and injecting random perturbations in memory systems timing. Results correspond to the parallel phase of each program.

V. Evaluation Results

A. Impact on the number of misses with indirection

DiCo-CMP improves the performance of parallel applications by avoiding indirection. Figure 4 shows the percentage of cache misses that suffer indirection. We consider that a read miss is free from indirection when it is directly sent to the cache that keeps the directory information for the corresponding block and that can provide a copy of the block (the L2 cache in a directory protocol or the owner cache in DiCo-CMP). Write misses are free from indirection when the condition for read misses is fulfilled and invalidations are not required. Finally, an upgrade miss avoids indirection when it takes place in the owner cache (only for DiCo-CMP). In all the cases, indirection avoidance leads to two-hop misses.

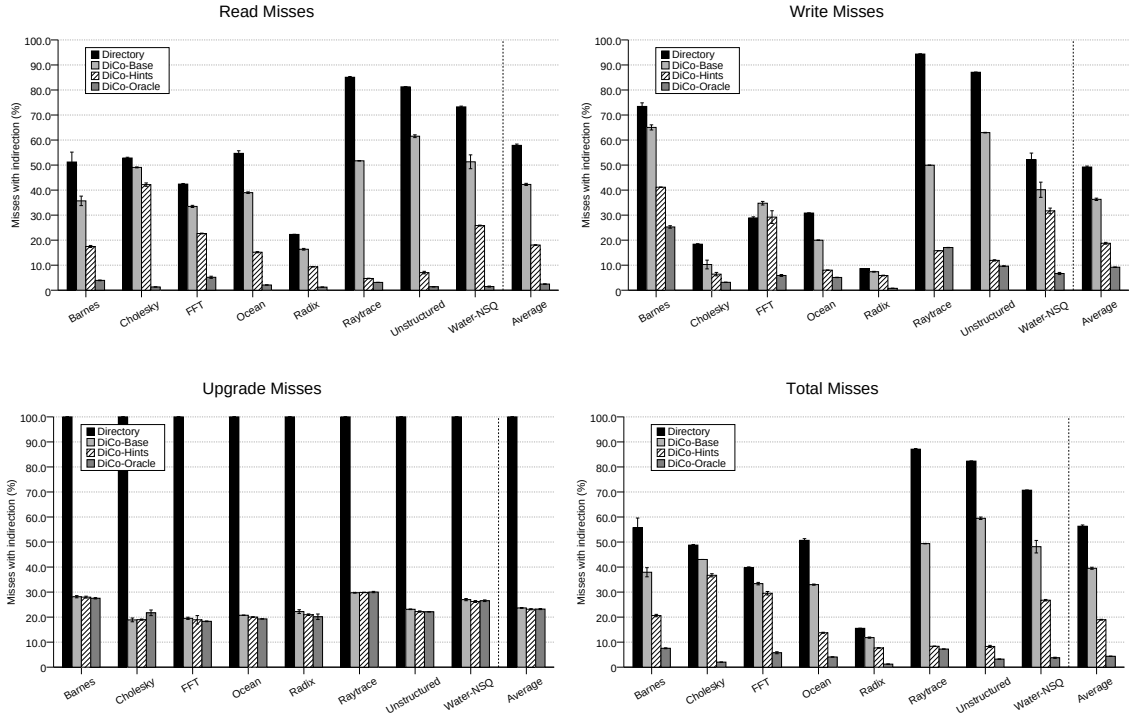

Fig. 4. Percentage of cache misses with indirection.

Considering the type of misses, we can see that for read misses indirection can be more easily avoided than for write misses. This is due to the fact that sometimes write misses require invalidations, thus preventing that the miss can be solved in two hops. On the other hand, the three configurations of DiCo-CMP obtain the same results for upgrade misses, since when a copy of the block resides in the requesting cache the identity of the owner is always known.

Finally, comparing the three configurations of DiCo-CMP, we can see that the *DiCo-base* configuration has some miss-predictions when the miss is sent to the owner cache. In some cases, when a block must be invalidated due to a write miss (or a read miss for a migratory block), the owner cache has the only valid copy of the block. This is the case of the migratory-sharing pattern and it does not require sending invalidations. Thus, the processor that frequently shares the block cannot update the pointer stored in the L1C$, and subsequent misses will fail to find the correct owner cache. As can be observed in Figure 4, the *DiCo-hints* configuration avoids this situation by sending hints to the frequent sharers.

B. Impact on execution time

The ability of avoiding indirection that DiCo-CMP shows, translates into reductions in applications' execution time. Figure 5 plots the average execution times that are obtained by the applications evaluated in this paper. All the results have been normalized with respect to those observed for the Token-CMP protocol.

In general, we can see from Figure 5 that Token-CMP achieves improvements of 5% on average in execution time with respect to a directory protocol. As already discussed, Token-CMP avoids indirection by

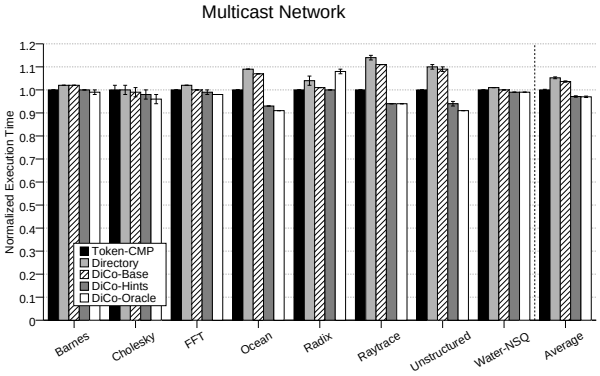

Fig. 5. Normalized execution times.

broadcasting requests to all caches. DiCo-CMP does not rely on broadcasting but requests are just sent to the potential owner cache. It is clear that the performance achieved by DiCo-CMP will depend on its ability to find the actual owner cache. We observe slight improvements in execution time for *DiCo-base* compared to the directory protocol. As commented on in the previous section, just a small fraction of the misses with indirection could be converted into two-hop misses for *DiCo-base*. On the other hand, the significant fraction of two-hop misses that can be achieved when *DiCo-hints* is considered, and the fact that our proposal removes some of the inefficiencies that Token-CMP introduces (broadcasting and persistent requests) translate into improvements of 3% on average over Token-CMP. Finally, the *DiCo-hints* policy can obtain virtually the same results than the unimplementable *DiCo-oracle* policy.

C. Impact on network traffic

Figure 6 compares the network traffic generated by the configurations considered in this paper for

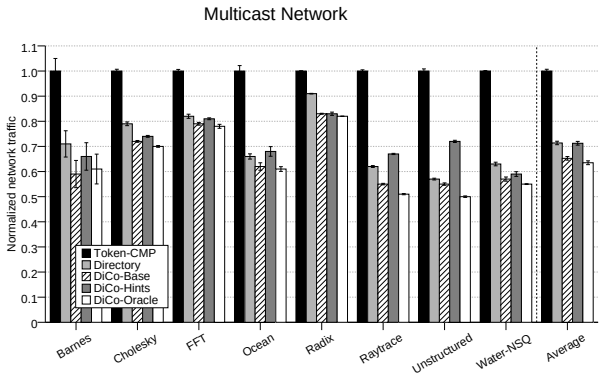

Fig. 6. Normalized network traffic.

the two networks evaluated. In particular, each bar plots the number of bytes transmitted through the interconnection network (the total number of bytes transmitted by all the switches of the interconnect) normalized with respect to the Token-CMP case. As we can see, the fact that Token-CMP needs to broadcast requests makes this protocol obtain the highest traffic levels.

Network traffic can be dramatically reduced when the directory protocol is employed (28% on average). This is due to requests are sent to the directory controller at the L2, which in turn sends coherence messages just to the L1 caches that must observe them. Since DiCo-CMP removes the communication between the directory and the owner cache, less coherence messages are needed to solve cache misses. This reduction in the number of messages translates into lower network traffic compared to a directory protocol (8% for *DiCo-base* and 11% for *DiCo-oracle*). *DiCo-hints*, however, shows higher network traffic due to the use of hints to keep updated the information stored in the L1C$ structures. In some cases, this results in traffic levels higher than the observed in the directory protocol (but always lower than those reached by Token-CMP), as occurs for Raytrace and Unstructured, but on average *DiCo-hints* generates virtually the same network traffic than a directory protocol would do, and 28% less traffic than Token-CMP.

VI. Conclusions

Tiled CMP architectures (i.e. arrays of replicated tiles connected over a switched direct network) have recently emerged as a scalable alternative to current small-scale CMP designs, and will be probably the architecture of choice for future CMPs. The technological parameters and power constrains entailed by CMPs demand new solutions to the cache coherency problem.

In this work, we present DiCo-CMP, a cache coherence protocol for tiled CMP architectures that meets the advantages of directory and Token-CMP protocols and avoids their problems. In DiCo-CMP the role of storing up-to-date sharing information and ensuring totally ordered accesses for every memory block is assigned to the owner cache. Compared to a directory protocol, our proposal avoids the indirection that the access to the directory entails and, therefore, reduces the latency by directly sending the requests to the owner cache (as it would be done in Token-CMP). In this way, DiCo-CMP achieves improvements in total execution time of 8% on average over a directory protocol and of 3% over Token-CMP. DiCo-CMP also reduces network traffic compared to Token-CMP to 28% by sending just one request message per miss, and consequently, the total power consumed in the interconnection network. These results confirm DiCo-CMP as a promising alternative to current cache coherence protocols for tiled CMPs.

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". A. Ros is supported by a research grant from Spanish MEC under the FPU national plan (AP2004-3735).

References

[1] Michael B. Taylor, Jason Kim, and Jason Miller, et al, "The raw microprocessor: A computational fabric for software circuits and general purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, May 2002.

[2] Michael Zhang and Krste Asanovic, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *ISCA*, June 2005, pp. 336–345.

[3] Michael R. Marty, J. Bingham, Mark D. Hill, A. Hu, Milo M.K. Martin, and David A. Wood, "Improving multiple-cmp systems using token coherence," in *HPCA-11*, Feb. 2005, pp. 328–339.

[4] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir, "Interconnect-power dissipation in a microprocessor," in *SLIP*, Feb. 2004, pp. 7–13.

[5] Manuel E. Acacio, José González, José M. García, and José Duato, "Owner prediction for accelerating cache-to-cache transfer misses in cc-NUMA multiprocessors," in *SC*, Nov. 2002.

[6] Manuel E. Acacio, José González, José M. García, and José Duato, "The use of prediction for accelerating upgrade misses in cc-NUMA multiprocessors," in *PACT*, Sept. 2002, pp. 155–164.

[7] Milo M.K. Martin, Pacia J. Harper, Daniel J. Sorin, Mark D. Hill, and David A. Wood, "Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors," in *ISCA*, June 2003, pp. 206–217.

[8] Milo M.K. Martin, *Token Coherence*, Ph.D. thesis, University of Wisconsin-Madison, Dec. 2003.

[9] Liqun Cheng, John B. Carter, and Donglai Dai, "An adaptive cache coherence protocol optimized for producer-consumer sharing," in *HPCA-13*, Feb. 2007, pp. 328–339.

[10] Luiz A. Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese, "Piranha: A scalable architecture based on single-chip multiprocessing," in *ISCA*, June 2000, pp. 12–14.

[11] Per Stenstrom, Mats Brorsson, and Lars Sandberg, "An adaptive cache coherence protocol optimized for migratory sharing," in *20st Int'l Symp. on Computer Architecture (ISCA'93)*, May 1993, pp. 109–118.

[12] Milo M.K. Martin, Mark D. Hill, and David A. Wood, "Token coherence: Decoupling performance and correctness," in *ISCA*, June 2003, pp. 182–193.

[13] Changkyu Kim, Doug Burger, and Stephen W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *ASPLOS X*, Oct. 2002, pp. 211–222.

[14] Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder P. Singh, and Anoop Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ISCA*, June 1995, pp. 24–36.