DiCo-CMP: Efficient Cache Coherency in Tiled CMP Architectures

Alberto Ros, Manuel E. Acacio, José M. García
Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia
Campus de Espinardo S/N, 30100 Murcia, Spain
{a.ros,meacacio,jmgarcia}@ditec.um.es

Abstract

Future CMP designs that will integrate tens of processor cores on-chip will be constrained by area and power. Area constraints make impractical the use of a bus or a crossbar as the on-chip interconnection network, and tiled CMPs organized around a direct interconnection network will probably be the architecture of choice. Power constraints make impractical to rely on broadcasts (as Token-CMP does) or any other brute-force method for keeping cache coherence, and directory-based cache coherence protocols are currently being employed. Unfortunately, directory protocols introduce indirection to access directory information, which negatively impacts performance. In this work, we present DiCo-CMP, a novel cache coherence protocol especially suited to future tiled CMP architectures. 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 (as it would be done in brute-force protocols), and reduces the network traffic compared to Token-CMP (and consequently, power consumption in the interconnection network) 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.

1. Introduction

The huge number of transistors that are currently offered in a single die has made major microprocessor vendors to shift towards multi-core architectures in which several processor cores are integrated on a single chip. Chip-multiprocessors (CMPs) [24] have important advantages over very wide-issue out-of-order superscalar processors. In particular, they provide higher aggregate computational power, multiple clock domains, better power efficiency, and simpler design through replicated building blocks.

Most current CMPs (for example, the IBM Power5 [13]) have a relatively small number of cores (2 to 8), every one with at least one level of private cache. These cores are typically connected through an on-chip shared bus or crossbar. However, the interesting new opportunity is now that Moore's Law will make it possible to double the number of cores every 18 months [7], making undesirable elements that could compromise the scalability of these designs. One of such elements is the interconnection network. As shown in [15], the area required by a shared bus or a crossbar as the number of cores grows has to be increased to the point of becoming impractical. 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 [28, 31].

On the other hand, most CMP systems provide programmers with the intuitive shared-memory model, which requires efficient support for cache coherency. 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 coherency problem [7].

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 [22]. 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 oc-

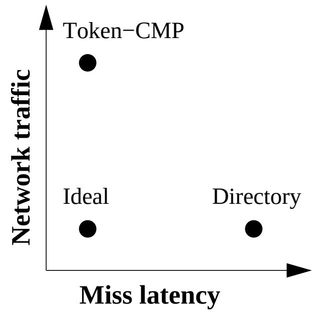


Figure 1. Trade-off between miss latency and network traffic.

curs). 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 [16, 29]. Figure 1 shows the trade-off between Token-CMP and directory-based cache coherence protocols. An ideal protocol for tiled CMPs would avoid the indirection of the directory protocols without relying on broadcasting requests.

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 replaces the directory cache required in directory protocols and 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 (as it would be done in Token-CMP), and reduces network traffic compared to Token-CMP by sending just one request message on every cache miss. Detailed simulations using an extended version of 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 when an interconnection network with multicast support is used. Without multicast support, DiCo-CMP achieves improvements of 6% over a directory protocol and 5% over Token-CMP. Moreover, our proposal reduces network traffic compared to Token-CMP (28% with multicast support and 42% without this kind of support), and consequently, the total power consumed in the interconnection network.

DiCo-CMP is a particular implementation of Direct Coherence for tiled CMP architectures. Direct Coherence represents a family of protocols that addresses the design of a solution to the cache coherence problem that avoids indirection without relying on any brute-force method (as broadcasting requests). In this way, implementations of Direct Coherence in other domains (as for example, distributed shared-memory multiprocessors [26]) are also possible.

The rest of the paper is organized as follows. In Section 2 we present a review of the related work. DiCo-CMP is described in Section 3. In Section 4 we study the area and power requirements of DiCo-CMP. Section 5 introduces the methodology employed in the evaluation. Section 6 shows the performance results obtained by our proposal. And finally, Section 7 concludes the paper.

2. 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 cache coherence protocols. First of all we comment on some of the related works.

Cheng *et al.* [10] 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 proposal is orthogonal to this work and the ideas presented in [10] could also be applied to DiCo-CMP.

Huh *et al.* [12] propose to allow replication in a CMP-NUCA cache to reduce the access time to a shared multi-banked cache. More recently, Beckmann *et al.* [5] 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, our protocol reduces miss latencies by avoiding the access to the L2 cache when it is not necessary, and no replication is performed. Again DiCo-CMP could be also used in conjunction with techniques that try to make the best use of the limited on-chip cache storage.

In the shared-memory multiprocessors domain, Acacio *et al.* propose to avoid the indirection for cache-to-cache transfer misses [1] and upgrade misses [2] 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 (reads, writes and upgrades) 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.

Martin *et al.* propose to use destination-set prediction to reduce the bandwidth required by a snoopy protocol [19]. 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 [18]. 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 and also in the context of shared-memory multiprocessors, 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.

Finally, some authors evaluated the use of hints with different objectives [6, 11]. In these works the authors try to keep updated directory information to find out where a fresh copy of the block can be obtained in case of a read miss. In contrast, we use the hints as a policy to update the location of the owner cache which servers as ordering point and stores up-to-date directory information.

2.1. Directory-CMP

Directory-based coherence protocols [8] have been widely used in shared-memory multiprocessors. Now, several Chip Multiprocessors, as Piranha [4], 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 [27], 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). This optimization has been shown to improve substantially the performance of many applications.

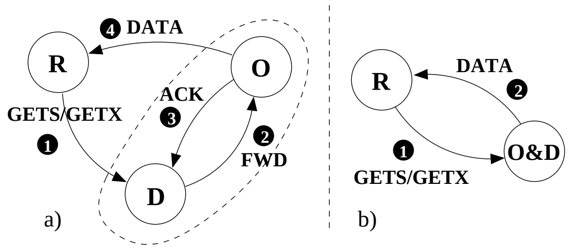


Figure 2. a) Cache-to-cache transfer in a directory protocol. b) Cache-to-cache transfer in DiCo-CMP. (R=Requester; D=Directory; O=Owner).

2.2. Token-CMP

Token coherence [20] 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. Token coherence avoids starvation by issuing a persistent request when a processor detects potential starvation. In this paper, we compare our coherence protocol against Token-CMP [22], which is a performance policy aimed at achieving low-latency cache-to-cache transfer misses. Token-CMP 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.

3. The DiCo-CMP Cache Coherence Protocol

As shown in Figure 2.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

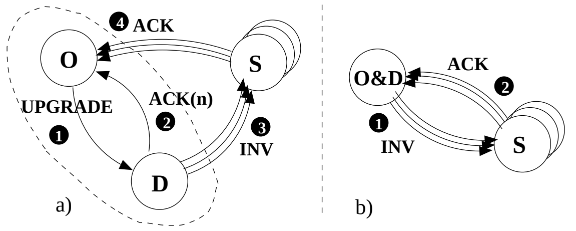


Figure 3. a) Upgrade in a directory protocol. b) Upgrade in DiCo-CMP. (O=Owner; D=Directory; S=Sharers).

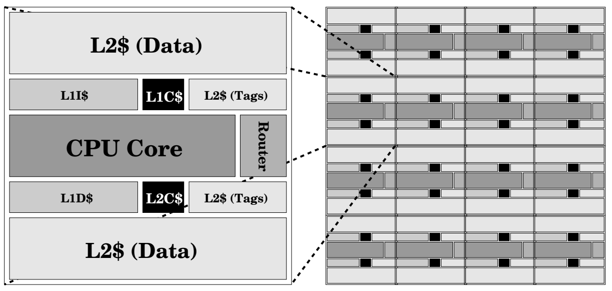


Figure 4. Organization of a tile (black boxes are the elements added by DiCo-CMP) and a 4×4 tiled CMP.

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 2.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. Finally, in DiCo-CMP the O&D cache can solve misses without using transient states (except for those that require invalidations) thus reducing the number of transient states and therefore making the implementation simpler than the directory protocol.

Another example of the advantages of DiCo-CMP is shown in Figure 3. This diagram represents an upgrade that takes place in the owner node, which happens frequently in common applications (i.e. producer-consumer pattern). 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:

- *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 3.3).

- *L2 coherence cache* (L2C$): Since the cache that ensures totally ordered accesses (the owner cache) is no longer fixed and can change on write misses, this structure must keep the identity of the current owner cache for each block allocated in any L1 data cache. This structure replaces the directory structure required by directory protocols and is accessed each time a request fails to locate the owner cache. Therefore, this information must be updated whenever the owner cache changes through control messages. These messages should be processed by the L2C$ in the very same order in which they were generated (see Section 3.2.3).

3.1. 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 between them[1]. 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 [14]). Moreover, the L1 and L2 caches are non-inclusive to exploit the total available cache capacity on chip. Figure 4 shows the organization of a tile (left) and a 16-tile CMP (right).

[1]Alternatively, each L2 slice could have been treated as a *private* L2 cache for the local processor. In this case, cache coherency had to be maintained at the L2 cache level (instead of L1). In any case, our proposal would be equally applicable in this configuration.

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 4, 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. A comparison among the extra storage and structures required by the three alternatives considered in this work can be found in Section 4.

3.2. Description of the cache coherence protocol

3.2.1. 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.

3.2.2. Request received by a cache that is not the owner

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 resends the request to the L2 cache. On the other hand, if it is the L2 cache and there is a hit in the L2C$, the request is sent to the current owner cache. In absence of race conditions the request will finally reach the owner cache. Finally, if there is a miss in the L2C$ 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)[2]. Moreover, it is necessary to allocate a new entry in the L2C$ pointing to the current L1 owner cache.

3.2.3. Request received by the owner cache

Every time a request reaches the owner cache, it is necessary to check whether this cache 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.

On the other hand, if the block is not in a transient state, the miss can be immediately solved. 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 cache of the requesting processor.

For write misses, the owner cache sends in first place 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). If the miss is an upgrade the owner cache checks the sharing code field to know whether the requester still holds a copy of the block (note that a previous write miss from a different processor could have invalidated its copy and in this case the owner cache should also provide a fresh copy of the block).

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. Fortunately, there are several approaches to ensure this order. In our particular implementation, 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 miss status hold register –MSHR– allocated on the cache miss is still held). The cache miss is considered finalized once this confirmation response (besides the message with data) has been received.

3.2.4. Replacements

In our particular implementation, when a block is evicted from the owner L1 cache, the block must be allocated at the L2 cache with the up-to-date directory information. Then, the L2C$ deallocates its entry for this block because the

[2] As already mentioned, we assume for all the configurations that the L1 and the L2 cache are non-inclusive. Our proposal is equally applicable with other configurations for the L1 and L2 caches, obtaining similar results to those presented in this work.

owner cache is now the L2 cache. If the replacement takes place in a cache that is not the owner, the replacement is performed transparently to the rest of the sharers.

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 cache to invalidate all the copies from the L1 caches. Luckily, as happens to the directory cache in directory-based protocols, an L2C$ with the same number of entries than the L1 cache and an associativity equal to the aggregate associativity of all L1 caches is enough to completely remove this kind of replacements.

3.3. 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. Notice that a very large L1C$ can cause an increase in the number of owner miss-predictions due to some entries could store obsolete information.

3.4. 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, even when it is sent by the L2 cache. 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 resent 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 resends 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.

4. Area and Power Considerations

In this section we compare the memory overhead and the extra structures needed by the three alternatives considered in this work: Token-CMP, the directory-CMP protocol and DiCo-CMP. Moreover, we discuss how frequently these structures are accessed to demonstrate that our proposal will not have significant impact on the power consumed by these structures and, therefore, significant reductions on total power consumption can be expected as a result of the important savings in terms of network traffic that DiCo-CMP entails (see Section 6.3).

Token-CMP needs to keep the token count for any block stored both in the L1 and L2 caches. This information only requires $1 + log_2(n)$ bits (the owner-token bit and non-owner token count), where n is the number of processor cores. These additional bits are stored in the tags' part of both cache levels. Moreover, it is necessary to add a counter to the miss status hold registers (MSHRs) that are allocated when misses are issued. This counter is in charge of reissuing transient requests after a timeout period. This timeout implies an increase in the number of accesses to the MSHRs. Finally, Token-CMP needs extra structures to implement the persistent requests mechanism. This structure has only one entry of few bytes per each processor core, and persistent requests are not very frequent.

Directory-based protocols store the on-chip 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. Therefore, a number of entries for the distributed directory cache equal to the number of entries of the L1 caches is enough to always find the directory information on chip for misses that can be solved without leaving the chip (obtaining the block from any cache). The directory must be accessed on each cache miss.

DiCo-CMP stores the directory information for blocks held in any L1 or L2 cache in the owner cache (L1 or L2). Moreover, it uses two structures that store a pointer to the owner cache, the L1 and L2 coherence caches. The L1C$

Table 1. Memory overhead introduced by coherence information (per tile) in a 4x4 tiled CMP

	Structure	Entry size	Entries	Total size	Overhead
Data	L1 cache	64 bytes	2K	128KB	
	L2 cache	64 bytes	16K	1024KB	
Token-CMP	L1$ tags	5 bits	2K	1.25KB	**+0.98%**
	L2$ tags	5 bits	16K	10KB	
Directory	L2$ tags	2 bytes	16K	32KB	**+3.12%**
	Dir cache	2 bytes	2K	4KB	
DiCo-CMP	L1$ tags	2 bytes	2K	4KB	**+3.30%**
	L2$ tags	2 bytes	16K	32KB	
	L1C$	4 bits	2K	1KB	
	L2C$	4 bits	2K	1KB	

is accessed only when it is known that there is a cache miss in order to keep power consumption low. The L2C$ is necessary to locate the owner cache whenever the information in the L1C$ is not correct. This structure is only accessed by misses affected by indirection (about 18% of the cache misses as shown in Section 6.1). As happens with the on-chip directory cache in the directory protocol, the L2C$ does not require more entries than the aggregate number of entries of the L1 caches. Differently from that directory cache, just one pointer is stored in each entry. In this way, the L2C$ required by DiCo-CMP has smaller size than the directory cache employed in the directory-CMP protocol. The *DiCo-hints* policy also requires to store the set of the frequent sharers in the tag's part of the L1 cache, which slightly increases the memory overhead[3].

For the particular configuration of this work (a 4×4 tiled CMP with 128KB L1 private caches), the number of bits required for storing the sharing code is 16 (2 bytes), whereas just $log_2 16 = 4$ bits are needed for storing a single pointer. Table 1 summarizes the structures and their size required by Token-CMP, directory-CMP and DiCo-CMP. Note that the table concentrates on the structures used for keeping coherence information and, therefore, does not account for the extra structures required by Token-CMP and DiCo-CMP to avoid starvation. We can see that DiCo-CMP has an overhead similar to the directory protocol.

5. Simulation Environment

We evaluate our proposal with full-system simulation using Virtutech Simics [17] extended with Multifacet GEMS [21]. 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 [25]. SICOSYS allows to take into account most of the VLSI implementation details with high precision but with much lower computational effort than hardware-level simulators. We have extended SICOSYS to allow us to simulate multicast networks.

[3]The memory overhead for the *DiCo-hints* policy is just 3.64%.

Table 2. 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
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

The simulated system is a tiled CMP organized as a 4×4 array of replicated tiles, as described in Section 3.1. Since we consider tiled CMP designs built from a relatively large number of cores, each tile contains an in-order processor core, thus offering better performance/Watt ratio than a small number of complex cores would obtain. Table 2 shows the values of the main parameters of the architectures evaluated in this work.

We have implemented the three policies described in Section 3.3: *base*, *hints* and *oracle*. These three implementations have been exhaustively checked using a tester program provided by GEMS that checks all race conditions to raise any incoherence. We compare our proposal against the Token-CMP protocol described in [22] and a directory protocol similar to the intra-chip coherence protocol used in Piranha [4].

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 [30]. Unstructured (Mesh.2K, 5 time steps) is a computational fluid dynamics application [23]. We account for the variability in multithreaded workloads [3] by doing multiple simulation runs for each benchmark in each configuration and injecting random perturbations in memory systems timing for each run. The experimental results reported in this paper correspond to the parallel phase of each program.

6. Evaluation Results

We compare the three policies for DiCo-CMP proposed in Section 3.3 with the Token-CMP and the directory protocol described in Section 2. First, we study to what extent DiCo-CMP reduces indirection compared to a directory protocol. Second, we show how the reduction in the number of misses affected by indirection impacts in the applications' execution time. Finally, we analyze the traffic in the on-chip interconnection network. We also show the impact on execution time and network traffic using both an interconnection network supporting multicast and another one without this kind of support.

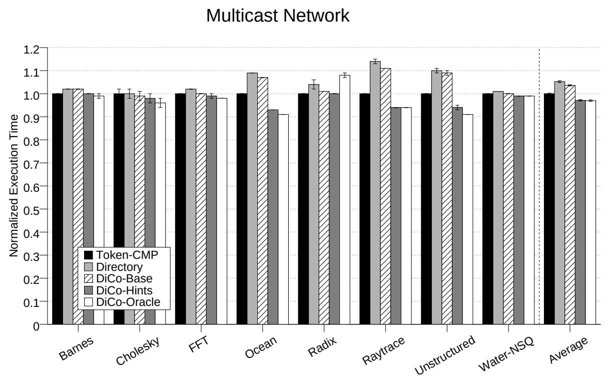
6.1. Impact on the number of misses with indirection

DiCo-CMP improves the performance of parallel applications by avoiding indirection. Figure 5 shows the percentage of cache misses that suffer indirection. Results for Token-CMP are not included because cache misses never suffer from indirection in this protocol.

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.

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. This is because upgrade misses avoid indirection when they take place in the owner cache and, therefore, the policy used to update the value of the L1C$ does not affect them. However, we can see that upgrade misses usually take place in the owner cache (77%).

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



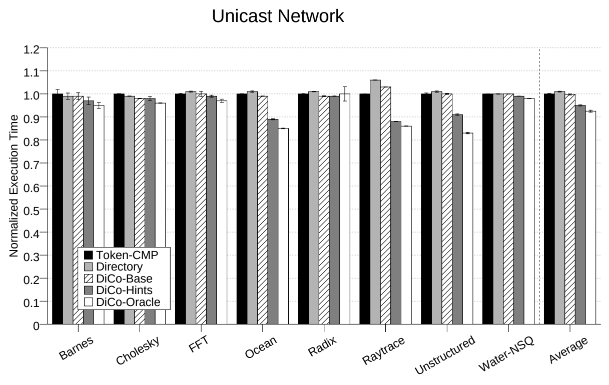


Figure 6. Normalized execution times.

owner cache. As can be observed in Figure 5, the *DiCo-hints* configuration avoids this situation by sending hints to the frequent sharers.

6.2. Impact on execution time

The ability of avoiding indirection that DiCo-CMP shows, translates into reductions in applications' execution time. Figure 6 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 6 that Token-CMP achieves improvements of 5% on average in execution time with respect to a directory protocol using a multicast network. As already discussed, Token-CMP avoids indirection by broadcasting requests to all caches. We can see that without the multicast support the improvement obtained by Token-CMP over a directory protocol is only 1%. 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 im-

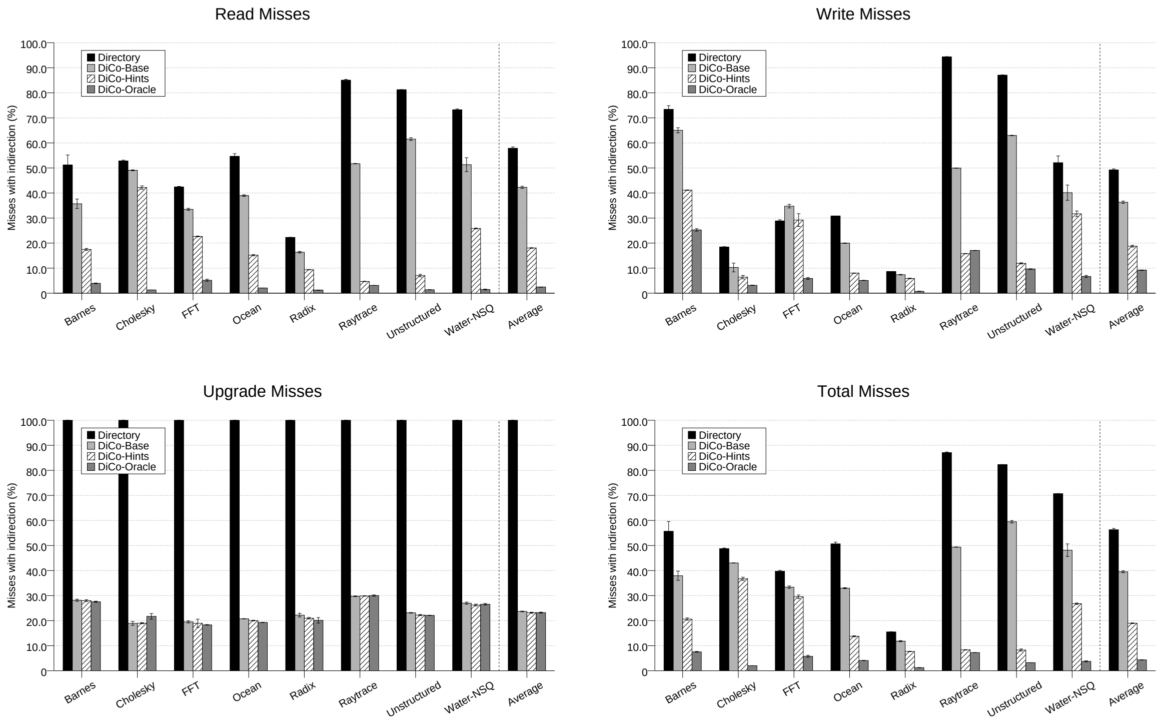


Figure 5. Percentage of cache misses with indirection.

provements 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% or 5% on average over Token-CMP using multicast and unicast networks, respectively. Finally, the *DiCo-hints* policy can obtain virtually the same results than the unimplementable *DiCo-oracle* policy when multicast support is given. In other case, the *DiCo-oracle* policy obtains improvements of 2% with respect to the *DiCo-hints* policy.

6.3. Impact on network traffic

Figure 7 compares the network traffic generated by the configurations considered in this paper for 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. Even when a single multicast message is sent per request for the Token-CMP, it must reach all the L1 caches in the system, thus increasing network traffic.

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. When the interconnection network does not support multicast routing, the network traffic is reduced to 54% on average. 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. Obviously this percentage becomes greater when the interconnect does not provide multicast support. In this case, *DiCo-hints* generates 42% less traffic than Token-CMP.

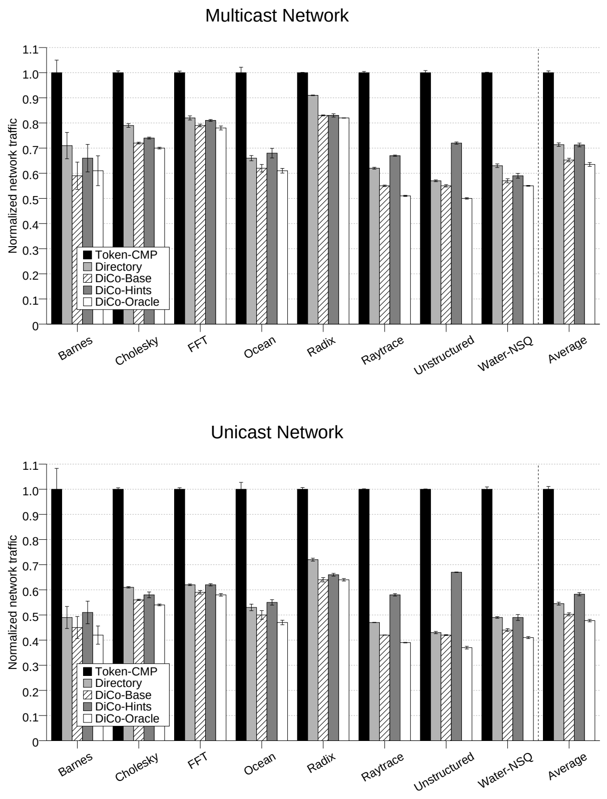


Figure 7. Normalized network traffic.

7. Conclusions and Future Work

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. 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 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. On the other hand, using an interconnection network without multicast support DiCo-CMP achieves improvements in total execution time of 6% on average over a directory protocol and of 5% over Token-CMP. DiCo-CMP also reduces network traffic compared to Token-CMP (28% with multicast support and 42% without this support) 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.

As part of our future work, we plan to design a new policy to update the L1C$ based on prefetching the identity of the owner, which could obtain the benefits of the *DiCo-hints* policy with lower memory storage (no information regarding the set of frequent sharers would be needed) and network traffic (hint messages to update the L1C$ on every ownership change would be avoided). Additionally, another topic of interest is to study the combination of DiCo-CMP and a heterogeneous interconnection network. Note that DiCo-CMP increases the number of messages that are not in the critical path of cache misses (for example, the hints) compared with a directory protocol. Since these messages could be sent using low-power wires without hurting performance [10], DiCo-CMP would make more extensive use of these wires than a directory protocol, thus resulting in lower power consumption.

8. Acknowledgements

This work has been jointly supported by Spanish MEC under grant "TIN2006-15516-C04-03", European Comission FEDER funds under grant "Consolider Ingenio-2010 CSD2006-00046" and European Comission funds under Network of Excellence HiPEAC. A. Ros is supported by a research grant from Spanish MEC under the FPU national plan (AP2004-3735).

References

[1] M. E. Acacio, J. González, J. M. García, and J. Duato. Owner prediction for accelerating cache-to-cache transfer misses in cc-NUMA multiprocessors. In *SC2002 High Performance Networking and Computing*, Nov. 2002.

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

[3] A. R. Alameldeen and D. A. Wood. Variability in architectural simulations of multi-threaded workloads. In *9th Int'l Symp. on High Performance Computer Architecture (HPCA-9)*, pages 7–18, Feb. 2003.

[4] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets,

and B. Verghese. Piranha: A scalable architecture based on single-chip multiprocessing. In *27th Int'l Symp. on Computer Architecture (ISCA'00)*, pages 12–14, June 2000.

[5] B. M. Beckmann, M. R. Marty, and D. A. Wood. ASR: Adaptive selective replication for cmp caches. In *39th Int'l Symp. on Microarchitecture (MICRO'06)*, pages 443–454, Dec. 2006.

[6] M. Bjorkman, F. Dahlgren, and P. Stenstrom. Using hints to reduce the read miss penalty for flat COMA protocols. In *28th Int'l Conference on System Sciences*, pages 242–251, Jan. 1995.

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

[8] L. Censier and P. Feautrier. A new solution to coherence problems in multicache systems. *IEEE Transactions on Computers*, 27(12):1112–1118, Dec. 1978.

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

[10] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter. Interconnect-aware coherence protocols for chip multiprocessors. In *33th Int'l Symp. on Computer Architecture (ISCA'06)*, pages 339–351, June 2006.

[11] H.-C. Hsiao and C.-T. King. Boosting the performance of now-based shared memory multiprocessors through directory hints. In *20th IEEE Int'l Conference on Distributed Computing Systems (ICDCS'00)*, pages 602–609, Apr. 2000.

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

[13] R. Kalla, B. Sinharoy, and J. M. Tendler. IBM Power5 Chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, Mar. 2004.

[14] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X)*, pages 211–222, Oct. 2002.

[15] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *32th Int'l Symp. on Computer Architecture (ISCA'05)*, pages 408–419, June 2005.

[16] N. Magen, A. Kolodny, U. Weiser, and N. Shamir. Interconnect-power dissipation in a microprocessor. In *Int'l workshop on System Level Interconnect Prediction (SLIP'04)*, pages 7–13, Feb. 2004.

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

[18] M. M. Martin. *Token Coherence*. PhD thesis, University of Wisconsin-Madison, Dec. 2003.

[19] M. M. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors. In *30th Int'l Symp. on Computer Architecture (ISCA'03)*, pages 206–217, June 2003.

[20] M. M. Martin, M. D. Hill, and D. A. Wood. Token coherence: Decoupling performance and correctness. In *30th Int'l Symp. on Computer Architecture (ISCA'03)*, pages 182–193, June 2003.

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

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

[23] S. Mukherjee, S. Sharma, M. D. Hill, J. Larus, A. Rogers, and J. Saltz. Efficient support for irregular applications on distributed-memory machines. In *5th Int'l Symp. on Principles & Practice of Parallel Programming (PPOPP'95)*, pages 68–79, July 1995.

[24] K. Olukotun, B. A. Nayfeh, L. Hammond, K. G. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *7th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, pages 2–11, Oct. 1996.

[25] V. Puente, J. A. Gregorio, and R. Beivide. SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems. In *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 15–22, Jan. 2002.

[26] A. Ros, M. E. Acacio, and J. M. García. Direct coherence: Bringing together performance and scalability in shared-memory multiprocessors. In *14th Int'l Conference on High Performance Computing (HiPC'07)*, pages 147–160, Dec. 2007.

[27] P. Stenstrom, M. Brorsson, and L. Sandberg. An adaptive cache coherence protocol optimized for migratory sharing. In *20st Int'l Symp. on Computer Architecture (ISCA'93)*, pages 109–118, May 1993.

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

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

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

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