On the Evaluation of Dense Chip-Multiprocessor Architectures

Francisco J. Villa, Manuel E. Acacio, José M. García
Departamento de Ingeniería y Tecnología de Computadores
University of Murcia, 30080 Murcia (Spain)
Email: {fj.villa,meacacio,jmgarcia}@ditec.um.es

***Abstract*— Chip-multiprocessors (CMPs) have been revealed as the most promising way of making efficient use of current improvements in integration scale. Nowadays, commercial CMP releases integrate at most 8 processor cores onto the chip. However, 16 or more processor cores are expected to be offered in near future Dense-CMP (D-CMP) systems. In this way, these architectures impose new design restrictions, and some topics, such as the cache-coherence problem, must be reviewed.**

In this paper we present an exhaustive performance evaluation of two recently proposed D-CMP architectures, making special emphasis on the solution to the cache-coherence problem that each one of them introduces. The Shared Bus Fabric architecture (SBF) features a snoop cache-coherence protocol and is based on a high-performance bus fabric interconnection network. The second architecture follows a directory-based approach and integrates a bi-dimensional mesh as the interconnection network. Our results show that the performance achieved by the SBF architecture is hard-limited by the bandwidth restrictions of the bus fabric. On the other hand, the directory-based architecture outperforms the SBF one, but presents some performance inefficiencies due to the additional indirection that the directory structure stored in the L2 cache level introduces.

I. Introduction

Recent advances in integration scale have enabled chip-multiprocessor architectures (or CMPs), where multiple processor cores, as well as some other structures such as the cache hierarchy and the interconnection network, are placed on a single die [1], [2]. These architectures provide higher performance than huge monolithic superscalar processors, and at the same time, they simplify the process of designing and verifying the architecture. Besides, chip-multiprocessor architectures are specially suited for the embedded computing domain, as these architectures provide the computational concurrency required to handle real-world events in real time, so that most microprocessors vendors are turning their attention to CMP architectures [3].

Some recent commercial small-scale chip-multiprocessor releases [4], [5] integrate at most 8 processor cores. However, these implementations do not match with future dense-CMPs (or D-CMPs), in which 16 processor cores or more are expected to be integrated on a single chip [6]. Unfortunately, D-CMPs impose new restrictions (physical space limitations, latencies on the critical path are even more crucial, new architectural bottlenecks) that are not found in current chip-multiprocessors, and thus, it is necessary to evaluate the problems that arise when designing such architectures.

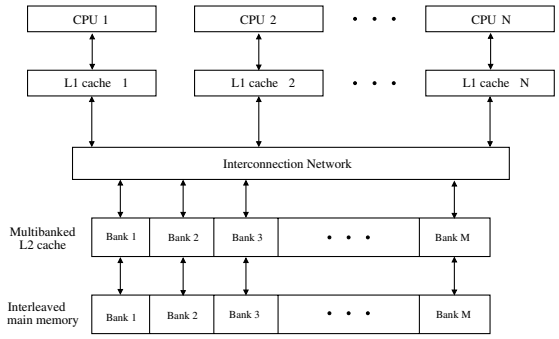


Fig. 1. Anatomy of a typical CMP architecture.

One of the main issues to be assessed in D-CMPs is the problem of cache-coherence when parallel workloads are executed. This represents a well-known problem in traditional multiprocessors, and a large body of literature dealing with it can be found. However, the particular characteristics of D-CMPs enforce researchers to review the classical solutions to the cache-coherence problem in order to adapt them to this new design space.

Most of the state-of-the-art CMPs implement the architecture shown in Figure 1. In this architecture the communication is based on a bus, each core has a private L1 cache and all the cores share the L2 cache, so that it is necessary to specify a coherence solution to maintain the cache coherence at the first level caches.

One possibility for avoiding the cache coherence problem in this kind of architectures consists in using the L1 caches to store only private data, while shared data is only cached at the second level cache, which is shared among all the processor cores. In this way, there are not private copies of the shared data, so no coherence protocol is needed. Although this solution is very simple and easily implementable, it is however expected to perform poorly when compared to snoop-based cache coherence implementations.

This fact is corroborated from the results presented in Figure 2. In this Figure we compare the solution that does not use the L1 caches to store shared data (*Bus-WithoutCoherenceProtocol*, or *Bus-WCP*), an implementation featuring a simple split-transaction bus in which requests and responses use the same physical interconnection and that models a MOESI-like cache-coherence protocol (*Bus*), and finally,

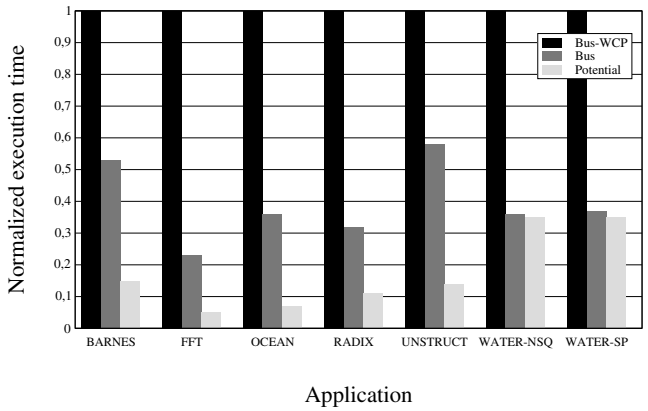


Fig. 2. Normalized execution time for several snoop-based cache-coherence implementations.

an idealistic architecture that behaves like a bus-snoop based implementation from a logical point of view, but includes a 2D-mesh point-to-point interconnection network (*Potential*). The *Potential* architecture, which is not implementable, gives a theoretical performance limit for these proposals that implement a cache-coherence mechanism. All the implementations assume 16 processor cores and an architecture like the one shown in Figure 1. The execution time is normalized with respect to *Bus-WCP*. More details about the evaluation methodology that we have used can be found in Section III.

As we can see, the *Bus-WCP* implementation is outperformed by a simple bus-based cache-coherent architecture, so that it becomes mandatory to implement a cache coherence protocol in this kind of architecture if high-performance is the goal. At the same time, as the *Potential* architecture shows, there is much room for improvement when considering the implementation of the cache-coherence protocol, so that researchers are expected to provide novel solutions that take into consideration the particularities of D-CMP systems.

Recently, Kumar *et al.* have proposed a CMP architecture that implements a snoop-based protocol on a high-performance Shared Bus Fabric (*SBF*) interconnection network [7]. This interconnection network comprises four separate pipelined buses used for sending/receiving requests/replies, inserting the requests in the snoop queue of each processor and collecting the snoop responses for all the processors. This architecture, which combines a snoop-based cache-coherence protocol with a complex shared bus fabric, appears as a reasonable high-performance solution to the problem of cache-coherence in D-CMPs.

Alternative solutions to the one previously described are those based on the Non-Uniform Cache Architecture (NUCA) model [8]. This technique allows that those cache banks that are nearer to one processor have lower access latencies than those that are far away. Two architectures based on this kind of cache organization have been proposed in [9] and [10] respectively. Both implementations are based on a point-to-point interconnection network and a directory cache-coherence protocol embedded in the L2 cache.

These two families of architectures (*SBF* and NUCA-based implementations) constitute the most recent high-performance proposals dealing with the problem of cache-coherence in dense-CMPs. This paper evaluates and compares the advantages and drawbacks of these two architectures using a detailed performance simulator and parallel scientific workloads. The main contributions of the paper are the following:

- A detailed performance evaluation of two recently proposed Dense-CMP architectures, making emphasis on the cache-coherence protocols implemented by each one and the implications of these protocols in the results. To the best of our knowledge, this is the first comparison between these two alternative organizations.
- Identification of the main architectural bottlenecks for these two architectures. The performance achieved by the *SBF* architecture is hard-limited by the bandwidth restrictions of the bus fabric, while the directory-based implementations present some inefficiencies related to the additional indirection that this structure introduces.

The rest of the paper is organized as follows. Section II describes the D-CMP architectures that have been evaluated. In Section III, we present a detailed performance evaluation of the architectures. Section IV summarizes the related work. Finally, Section V outlines the main conclusions of this work and points out some future ways.

II. Architecture of Future D-CMPs: SBF vs. NUCA

In this Section, we describe the two architectures that have been recently proposed for organizing Dense-CMP designs and that we have evaluated in this work. In both cases, we consider a CMP composed of 16 out-of-order processor cores, a first level of private caches, a second level of shared, multibanked cache and an interconnection network connecting the two level of caches.

A. A SBF-based CMP architecture

The first architecture evaluated features a snoop-based MOESI-like cache-coherence protocol based on a shared bus fabric (*SBF*) similar to the one presented in [7], although in the referred paper the authors assume private L2 caches and the shared bus fabric connects the L2 caches with the higher level of the memory hierarchy.

The *SBF* was originally proposed as a high-speed link in order to communicate data between processors, caches, IO and memory within a CMP system and it is the on-chip equivalent to the bus employed in snoop-based shared memory multiprocessors. As we can see in Figure 3, the *SBF* comprises four different, pipelined buses:

- *Address Bus* (AB): when the requester is granted access to the bus, it inserts the corresponding request in this bus.
- *Snoop Bus* (SB): the requests put in the AB are taken off the end of the address bus and inserted in a snoop queue connected to this bus.
- *Response Bus* (RB): each snooping node places its response to the snooped request in this bidirectional bus. The response logic placed at the end of the bus is

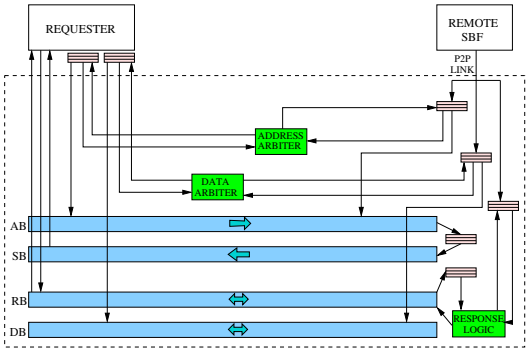


Fig. 3. The shared bus fabric implemented.

responsible for collecting all the responses and generating a broadcast message identifying the action that each structure must take.

- *Data Bus* (DB): the data is sent over this bidirectional bus to the requester.

When there are several bus fabrics in the system (for example, when the number of cores is greater than 8, as it is the case in this paper), they are connected by means of a point-to-point link. This link is able to transfer the three types of transactions (request, response and data) and is terminated with multiple queues at each end, as shown in Figure 3. These queues are arbitrated (together with the local ones) in order to grant access to each bus. We refer the interested reader to [7] for a detailed description of the *SBF* proposal.

B. CMP design Implementing the NUCA Model

The second dense-CMP architecture that we have evaluated is similar to those proposed in [9], [10] and features a point-to-point 2D-mesh interconnection network. In order to maintain coherence at the first level caches, a directory-based protocol is used. The directory structure is integrated into the chip and located at the same level as the L2 cache banks. The directory is implemented using a full-map bit-vector scheme and only keeps track of the lines stored at the L2 cache (the inclusion property is maintained between the two levels of cache, so the information about the L1 local copies is precise at every moment). When the directory receives a data request from the L1 caches, it decides if the request must be satisfied by another L1 cache, one of the L2 cache banks or main memory. If the request is an upgrade, the directory sends invalidation messages to other L1 caches that have a copy of the requested block. When a line is replaced at the L2 cache, the directory also sends invalidation messages to all the L1 caches sharing the evicted line to ensure the inclusion property. Finally, MESI states are used in the L2 caches.

This directory-based architecture has a layout similar to the one proposed by Beckmann and Wood in [9] (see Figure 4). The total number of L2 cache banks is sixteen. We have seen in our simulations that a lower number of banks hurts performance, while a higher number does not provide noteworthy benefits. In order to reduce the distance between the cores and the L2 cache banks, cache banks in the proposed layout are placed in the center of the chip, with the processor cores around them. Once again, we have evaluated other possible layouts, but the one that we have chosen presents the lowest network latencies, and consequently, shows the best performance numbers.

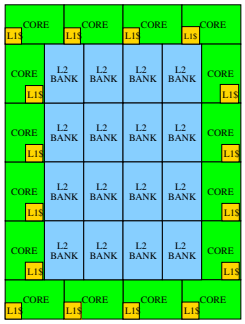


Fig. 4. Layout for a D-CMP based on a 2-D mesh.

The network implemented is a 2-dimensional bi-directional mesh including separate networks for requests and replies. The system implements pipelined switches, and hence the flit delay of multiple flits can be incurred in a pipelined way. The network routes packets using dimension-ordered routing, and each switch provides wormwhole routing.

III. Experimental Methodology and Results

A. Simulation Environment

In this section, we present a detailed performance evaluation of the D-CMP architectures described in the previous Section. In all the configurations, the processor model that has been simulated is similar to the MIPS R10000 processor [11], with an issue width of 4 instructions and a reorder buffer of 64 entries. The memory consistency model is an optimized implementation of the sequential consistency memory model that includes load speculation and allows stores to graduate before completion.

We have extended the Rice Simulator for ILP Multiprocessors (RSIM) [12] in order to model the D-CMP architectures evaluated [13]. We compare the two architectures described in the previous Section with an ideal (but unimplementable) architecture that behaves like a bus-based, snoop-coherent D-CMP from a logical point of view, but features a 2D mesh interconnection network instead of a bus. Thus, the *Potential* architecture features the characteristics of snoop-based cache-coherence (protocol simplicity, lower overhead) but at the same time takes advantage of the bandwidth provided by a point-to-point interconnection network. In Table I we can see the system parameters used in this work.

Table II describes the benchmarks that we have used in our experiments. This set of parallel scientific applications covers a variety of computation and sharing patterns. BARNES-HUT, FFT, OCEAN, RADIX, WATER-NSQ and WATER-SP belong to the SPLASH-2 benchmark suite [14]. BARNES-HUT application simulates the interaction of a system of bodies in three dimensions over a number of time steps,

TABLE I

SYSTEM CONFIGURATION.

Base Configuration	
Number of cores	16
Clock frequency	2 Ghz
L1 size	64KB
L1 associativity	4-way
L1 latency	1 cycle tags + 1 cycle data
L2 size	4MB (total size)
Number of L2 banks	16 (256KB per bank)
L2 associativity	8-way
L2 latency	6 cycles tags + 9 cycles data
Line size	32 bytes
Memory latency	200 cycles
Shared Bus Fabric	
Arbitration	2 cycles
Bandwidth	4 GBytes
2D-Mesh	
Size	6x6
Arbitration	2 cycles
Link Bandwidth	4 GBytes

TABLE II

APPLICATIONS AND INPUT SIZES USED IN THIS WORK.

Application	Input size
BARNES-HUT	4096 bodies, 4 time steps
FFT	64K complex doubles
OCEAN	130x130 ocean
RADIX	512K keys, 1024 radix
UNSTRUCTURED	Mesh.2K, 5 time steps
WATER-NSQ	512 molecules, 4 time steps
WATER-SP	512 molecules, 4 time steps

using the Barnes-Hut hierarchical N-body method. The FFT kernel is a complex 1-D version of the radix-$\sqrt{n}$ six-step FFT algorithm, which is optimized to minimize interprocessor communication. The data set consists of the n complex data points to be transformed, and another n complex data points referred to as the roots of unity. The OCEAN application studies large-scale ocean movements based on eddy and boundary currents. The RADIX program sorts a series of integers, called keys, using the popular radix sorting method. WATER-NSQ performs an N-body molecular dynamics simulation of the forces and potentials in a system of water molecules. WATER-SP solves the same problem as WATER-NSQ, but uses a more efficient algorithm for large numbers of molecules. Finally, UNSTRUCTURED is a computational fluid dynamics application [15]. The application sizes have been chosen taking into account the recommendations of [14] as well as the number of cores and the L1 cache size in our architecture. We have tried to maintain the L1 cache hit rates higher than 90% when possible.

B. Experimental Results

The first performance metric that we present is the execution time (see Figure 5(a)). The results are normalized with respect to *SBF*. As we can see, for all the applications (with the exception of WATER-NSQ and WATER-SP) the directory based architecture (from now on, *Directory*) clearly outperforms the *SBF* implementation. All the architectures perform similarly in the case of WATER-NSQ. When comparing *Directory* and *Potential* both architectures present very similar results, with the exceptions of UNSTRUCT and WATER-SP (the approximate differences are 20% and 10% respectively). These results reveal the directory-based architecture as a very competitive choice when designing a high-performance D-CMP, although there is still room for improvement in this configuration.

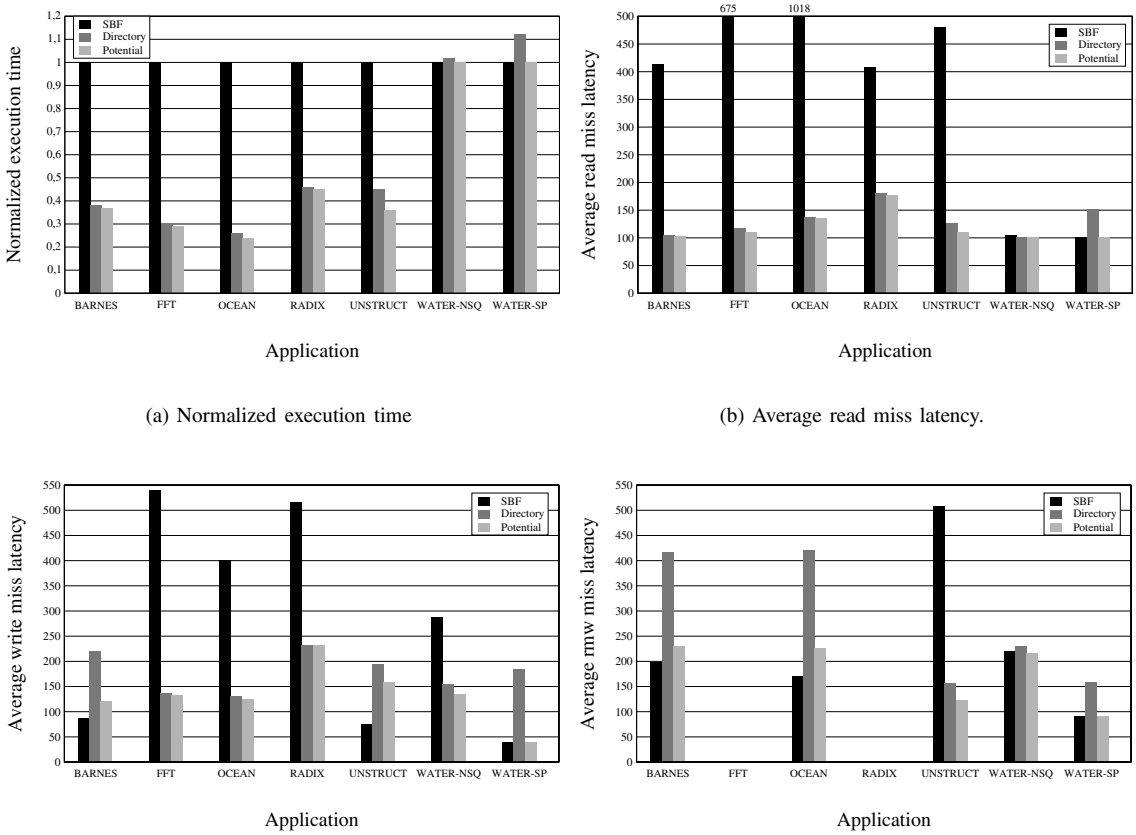
Figures 5(b), 5(c) and 5(d) show the average miss latency expressed in processor cycles for read, write and read-modify-write operations respectively. For read misses, the results maintain the same tendency than in the case of the execution time. However, the reductions for the *Directory* architecture are much more impressive, reaching a factor of 7 in OCEAN. Again, the directory-based approach performs worse than the snoop-based one for WATER-SP.

For write misses, the *SBF* architecture presents shorter latencies for three of the applications (BARNES-HUT, UNSTRUCT and WATER-SP), and the benefits of the *Directory* architecture are less pronounced than the obtained in read misses. For write operations, the directory must invalidate the local copies of the line stored in some of the private caches when the line is in Shared state, and wait for the acknowledgment replies of each sharer, so this kind of miss requires more indirection to be completed than in the case of a snoop-based protocol. The invalidation process is more efficient in the *SBF* architecture, as the L1 cache controllers invalidate the shared line as soon as they observe the request in the *Snoop Bus* and the miss requires a single access to the L2 cache in order to be satisfied. The same reasoning is valid for read-modify-write operations, although in this case the *Directory* architecture only presents shorter latencies for UNSTRUCT. The codes of FFT and RADIX do not contain read-modify-write operations.

Finally, in Figure 6 we can see the average miss latency from another perspective. Misses are split into four categories in terms of which memory structure provides data [13]. In *$-to-$ misses*, the line is in a single cache, or the line is in several caches and one of them has the line in Owned state (only for snoop-based implementations). If no L1 cache can provide the line, and the L2 cache has a valid copy of it, we categorize the miss as a *Hit L2 miss*. When the only valid copy is in main memory, the miss is named *Mem miss*. Finally, *Inv misses* appear when the faulting cache has a valid copy of the line in Shared state but permission for writing is wanted.

For these four categories, we distinguish three latency components, corresponding to the cycles spent at the L1 cache controller (including the time spent until the request is inserted in the network and the time spent processing the corresponding reply), the shared interconnection network (including the time spent at the different queues and buses as well as the time waiting for the response on the response bus in the case of the *SBF*) and those spent retrieving data ($T_{controller}$, T_{net} and T_{mem} respectively). Miss latency has an additional component when the directory is used, corresponding to the time spent at this structure for processing the request (T_{dir}).

Figure 6(a) shows that the main bottleneck of the *SBF* architecture is the network. The time spent at this structure highly dominates the L1 miss latency for *$-to-$*, *Hit L2* and



(a) Normalized execution time

(b) Average read miss latency.

(c) Average write miss latency.

(d) Average rmw miss latency.

Fig. 5. Performance results for the D-CMP architectures evaluated.

Mem misses, so we can conclude that a shared bus fabric like the one evaluated in this work is not able to process efficiently the memory traffic that will be generated in 16-way D-CMP architectures.

When comparing the *SBF* latencies with the *Directory* ones, we see once again how *Directory* presents shorter miss latencies than *SBF*, except in the case of *Inv* misses and for WATER-NSQ and WATER-SP, as in these applications the *SBF* interconnection does not become saturated. For *Inv* misses, the directory must invalidate the local copies at the L1 private caches and wait for the positive replies to the invalidation requests from the sharers, which implies much more indirection than in the case of the *SBF* architecture. We can also see that *$-to-$* miss latency is higher than *Hit L2* miss latency for the *Directory* architecture. In this case, the directory protocol also introduces additional indirection for providing data.

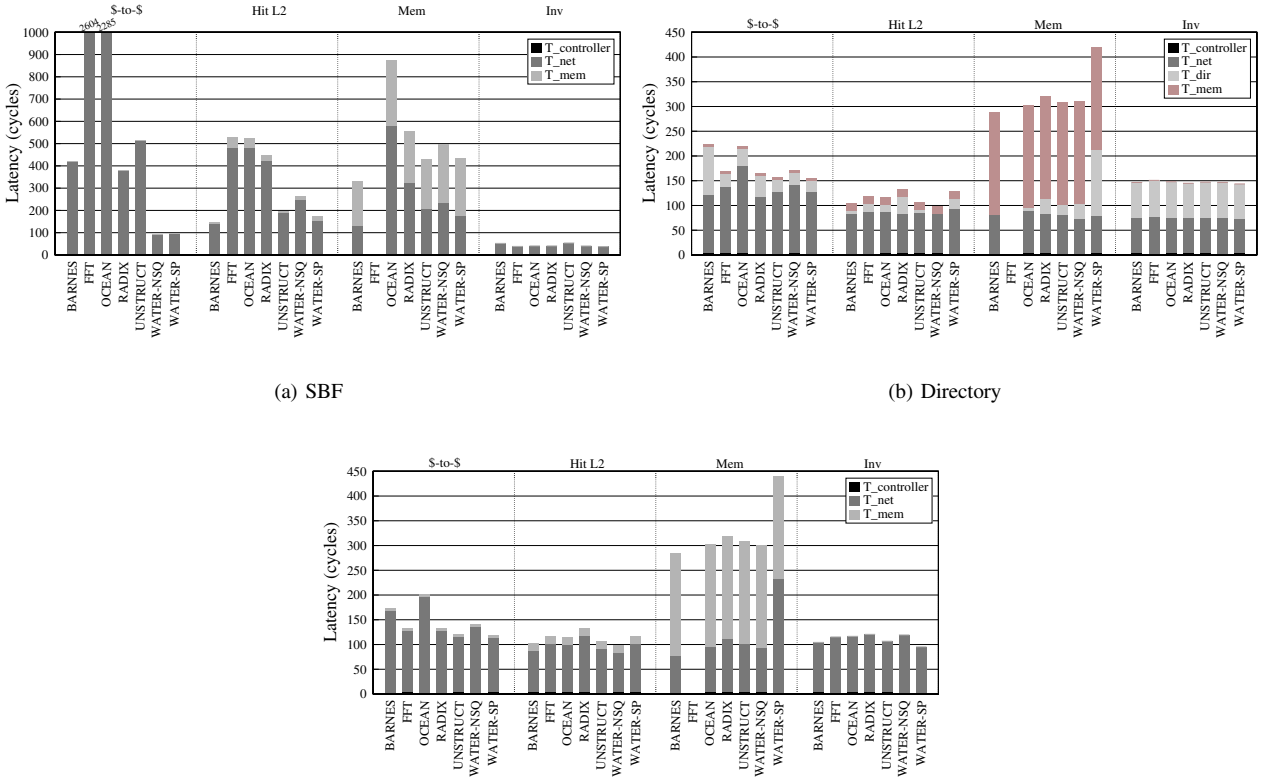
These two kind of misses hurt the performance obtained with the *Directory* architecture, and present the greatest difference when comparing the results with those obtained with the *Potential* architecture. We are currently working on optimizing these inefficiencies.

Finally, in order to provide sufficient insight into the trade-offs with respect to various design paramateres, we have performed some experiments using different L1 cache sizes (ranging from 16 KB to 128 KB). We have obtained similar results to those presented in this work, although the performance gap decreases with L1 cache sizes of 128 KB. However, we think that this is a side effect of the small working sets that the applications that we have used present, so we think that the L1 cache size is not a key paramater for this evaluation.

Another parameter that can affect the results is the number of cores. When the architectures integrate 4 processor cores, both implementations present similar results, as the Shared Bus Fabric does not get saturated, so the *SBF* architecture appears as a high-performance alternative for small-scale chip-multiprocessors.

IV. Related Work

One of the papers that most directly deals with the cache-coherence problem in CMPs is [16]. In this paper, the authors propose a hierarchical protocol for multiple-CMP systems that separates the intra-CMP coherence protocol from the inter-CMP protocol and is based on the token coherence protocol [17]. However, their design space differs substantially from those ones evaluated in this paper, as the authors consider CMPs composed of four processor cores.



(a) SBF

(b) Directory

(c) Potential

Fig. 6. Average latency for $-to-$, Hit L2, Mem and Inv misses.

The Piranha CMP [1] was one of the first CMP proposals integrating 8 cores onto the chip. To maintain intra-chip coherence, they propose a mechanism similar to a full-map centralized directory-based coherence protocol. The L2 cache controllers are responsible for enforcing coherence, so that each controller has complete and exact information about the on-chip cached copies for the subset of lines that map to it, as in the directory-based architecture we have evaluated.

Huh *et al.* [10] propose an organization for the on-chip memory subsystem of a CMP composed of 16 processors. The L2 cache is organized as a non-uniform cache architecture (NUCA) array. They use a switched network and a directory protocol, so the architecture they propose is very similar to the one we have evaluated, with some differences in the organization of the second level cache.

There are some other recent works in the literature dealing with the performance of the memory hierarchy in CMP architectures. Liu *et al.* [18] study several L2 cache organizations in order to increase the utilization (and in last term, the performance of the memory hierarchy) of this structure. They propose a mechanism that dynamically assigns L2 splits to each processor. In this way, they obtain fair use of the L2 cache, taking into account the demands of each processor at every moment. However, they do not consider other possible bottlenecks, such as the presence of the shared bus. As in the previous work, they simulate a CMP composed of 8 cores.

In [19], the authors consider tiled CMPs where each tile contains a slice of the total on-chip L2 cache storage and tiles are connected by an on-chip mesh network. They propose a hybrid cache management policy which combines the advantages of both private and shared L2 schemes.

In [20], the authors propose a central coherence unit and a new cache coherence protocol in order to reduce shared-bus transaction time. They evaluate several configurations with a variable number of processors, ranging from 2 to 8. However, the integration scale and memory subsystem latencies assumed in the paper differ highly from those found nowadays, so their results are not comparable with ours.

Finally, some other authors have studied the memory hierarchy when combining chip-multiprocessing with speculative multithreading, although most of them focus on the support for thread-level memory speculation [21], [22], [23]. Among them, Yanagawa *et al.* [24] perform a complexity analysis of a cache controller designed by extending a MSI controller to support thread-level memory speculation. They use a directory-based mechanism to maintain coherence, and find that the main component of memory latency is the delay incurred when accessing the directory.

V. Conclusions

This paper presents an exhaustive performance evaluation of two recent proposals for the organization of high-performance dense-CMP architectures, making special emphasis on the influence that the cache-coherence protocol has on performance. Simulation results show that a D-CMP implementation based on a bi-dimensional mesh interconnection network and a directory-based cache-coherence protocol outperforms an architecture featuring a shared bus fabric and a MOESI-like snoop protocol.

The study also identifies the structural bottlenecks of these two architectures. In the *SBF* implementation, the performance achieved is hard-limited by the buses. Even although four buses conform the *SBF*, it becomes saturated when 16 processor cores are connected, increasing the latency of the misses found at the L1 private caches.

In the case of the directory-based architecture, the interconnection network tolerates the coherence traffic originated in this kind of architecture. However, the indirection incurred when accessing the directory structure increases the overall latency for $-to-$ and Invalidation misses. We have seen that there is still room for improvement in this kind of architecture by reducing the latency for these two types of misses, as the results obtained for the *Potential* architecture demonstrate.

As part of our future work, we are currently developing prediction-based cache-coherence protocols to avoid the accesses to the directory information for *cache-to-cache* transactions and *invalidation* misses, which will bring therefore performance results closer to those of the *Potential* architecture.

Acknowledgments

This work has been supported by the Spanish Ministry of Educación y Ciencia and the European Union (Feder Funds) under grant TIC2003-08154-C06-03.

References

[1] 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 *Proc. of 27th Int'l Symp. on Computer Architecture*, June 2000, pp. 282–293.

[2] L. Hammond, B. A. Hubbert, M. Siu, M. K. Prabhu, M. Chen, and K. Olukotun, "The Stanford Hydra CMP," *IEEE Micro*, vol. 20, no. 2, pp. 71–84, March 2000.

[3] D. Geer, "Chip Makers Turn to Multicore Processors," *IEEE Computer*, vol. 38, no. 5, pp. 11–13, May 2005.

[4] R. Kalla, B. Sinharoy, and J. M. Tendler, "IBM Power5 Chip: A Dual-Core Multithreaded Processor," *IEEE Micro*, vol. 24, no. 2, pp. 40–47, March-April 2004.

[5] J. M. Tendler, S. Dodson, S. Field, and H. L. B. Sinharoy, "POWER4 System Architecture," IBM Server Group, Tech. Rep., 2001.

[6] K. Krewell, "Sun's Niagara pours on the cores," *Microprocessor Report*, vol. 18, no. 9, pp. 11–13, September 2004.

[7] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling," in *Proc. of 32nd Int'l Symp. on Computer Architecture*, June 2005.

[8] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Dominated On-Chip Caches," in *Proc. of 10th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2002, pp. 211–222.

[9] B. Beckmann and D. Wood, "Managing Wire Delay in Large Chip-Multiprocessor Caches," in *Proc. of 37th Int'l Symp. on Microarchitecture*, December 2004, pp. 319–330.

[10] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," in *Proc. of 10th Int'l Conf. on Supercomputing*, June 2005, pp. 31–40.

[11] K. C. Yeager, "The MIPS R10000 Superscalar Processor," *IEEE Micro*, vol. 16, no. 2, pp. 28–40, March-April 1996.

[12] C. J. Hughes, V. S. P. Pai, P. Ranganathan, and S. V. Adve, "RSIM: Simulating Shared-Memory Multiprocessors with ILP Proccesors," *IEEE Computer*, vol. 35, no. 2, pp. 68–76, February 2002.

[13] F. J. Villa, M. E. Acacio, and J. M. García, "Memory Subsystem Characterization in a 16-core Snoop-Based Chip-Multiprocessor Architecture," in *2005 Int'l Conf. on High-Performance Computing and Communications*, September 2005.

[14] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and Methodological Considerations," in *Proc. of 22nd Int'l Symp. on Computer Architecture*, June 1995, pp. 24–36.

[15] S. S. Mukherjee, S. D. Sharma, M. D. Hill, J. R. Larus, A. Rogers, and L. Saltz, "Efficient Support for Irregular Applications on Distributed-Memory Machines," in *Proc. of 5th Int'l Symp. on Principles and Practice of Parallel Programming*, July 1995, pp. 68–79.

[16] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood, "Improving Multiple-CMP Systems Using Token Coherence," in *Proc. of 11th Int'l Symp. on High-Performance Computer Architecture*, February 2005.

[17] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token Coherence: Decoupling Performance and Correctness," in *Proc. of 30th Int'l Symp. on Computer Architecture*, June 2003, pp. 182–193.

[18] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs," in *Proc. of 10th Int'l Symp. on High Performance Computer Architecture*, February 2004, pp. 176–185.

[19] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proc. of 32nd Int'l Symp. on Computer Architecture*, June 2005.

[20] M. Takahasi, H. Takano, E. Kaneko, and S. Suzuki, "A Shared-bus Control Mechanism and a Cache Coherence Protocol for a High-performance On-chip Multiprocessor," in *Proc. of 2nd Int'l Conference on High-Performance Computer Architecture*, February 1996, pp. 314–322.

[21] L. Hammond, M. Willey, and K. Olukotun, "Data Speculation Support for a Chip Multiprocessor," in *Proc. of 8th Int'l Symp. on Architectural Support for Parallel Languages and Operating Systems*, October 1998, pp. 58–69.

[22] V. Krishnan and J. Torrellas, "A Chip-Multiprocessor Architecture with Speculative Multithreading," *IEEE Transactions On Computers*, vol. 48, no. 9, pp. 866–880, September 1999.

[23] J. G. Steffan, C. B. Colohan, A. Zhai, and T. C. Mowry, "A Scalable Approach to Thread-Level Speculation," in *Proc. of 27th Int'l Symp. on Computer Architecture*, June 2000, pp. 1–12.

[24] Y. Yanagawa, L. D. Hung, C. Iwama, N. D. Barli, S. Sakai, and H. Tanaka, "Complexity Analysis of A Cache Controller for Speculative Multithreading Chip Multiprocessors," in *Proc. of 10th Int'l Conference on High Performance Computing*, December 2003, pp. 393–404.