Low-Overhead Organizations for the Directory in Future Many-Core CMPs*

Alberto Ros[1] and Manuel E. Acacio[2]

[1]Dpto. de Informática de Sistemas y Computadores
Universidad Politécnica de Valencia, 46022 Valencia (Spain)
[2]Dpto. de Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30100 Murcia (Spain)
aros@gap.upv.es, meacacio@ditec.um.es

Abstract. If current trends continue, today's small-scale general-purpose CMPs will soon be replaced by multi-core architectures integrating tens or even hundreds of cores on-chip. These many-core CMPs will implement the hardware-managed, implicitly-addressed, coherent caches memory model. Cache coherence in these designs will be maintained through a directory-based cache coherence protocol implemented in hardware. The organization of the directory structure will be a key design point due to the requirements in area that it will pose. In this work we study the effects on performance, network traffic and area that the use of compressed sharing codes for the directory will have in many-core CMPs. In particular, we select two compressed sharing codes previously proposed by us in the context of large-scale shared-memory multiprocessors that have very small area requirements. Simulation results of 32-core CMPs show that degradations of up to 32% in performance and 350% in network traffic are experienced. Additionally, since some proposals for efficient multicast support in on-chip networks have recently appeared, we also consider the case of using this kind of support in combination with the compressed sharing codes. Unfortunately, we found that multicast support is not enough to remove all the performance degradation that the compressed sharing codes introduce and barely can reduce network traffic.

1 Introduction

In the last years we have witnessed the substitution of single-core processors by multi-core ones. Following the Moore's Law that establishes that the number of transistors doubles every 18 months, it is expected that current small-scale general-purpose chip-multiprocessors (CMPs) will soon be followed by multi-core architectures integrating tens or even hundreds of cores on-chip [3]. Architectures of this type are usually known as many-core CMPs.

Many-core CMPs will be probably designed as arrays of identical or close-to-identical building blocks (tiles) connected over a switched direct network [12, 16]. Tiled architectures provide a scalable solution for supporting families of products with varying computational power, managing the design complexity, and effectively using the resources available in advanced VLSI technologies. As an example, Intel has recently announced the 48-core Single-chip Cloud Computer [1], an experimental research microprocessor that has been developed in the context of the Tera-scale Computing Research Program. More specifically, the Single-chip Cloud Computer consists of 24 tiles with two IA cores per tile, which are interconnected by means of a 24-router mesh network providing 256 GB/s bisection bandwidth.

* This research was supported by the Spanish MEC and MICINN, as well as European Commission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04, and PROMETEO from Generalitat Valenciana (GVA) under Grant PROMETEO/2008/060.

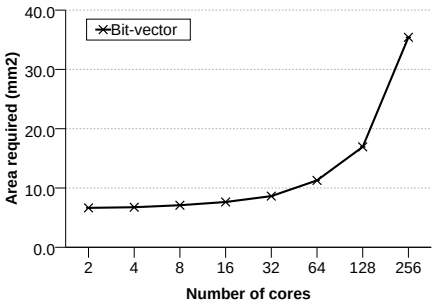


Fig. 1: Area (mm^2) required for a 1MB cache module when the bit-vector sharing code is used

On the other hand, if current trends continue, future many-core CMP architectures will implement the hardware-managed, implicitly-addressed, coherent caches memory model [6]. With this memory model, all on-chip storage is used for private and shared caches that are kept coherent in hardware by using a cache coherence protocol. In this way, each tile contains at least one level of cache memory that is private to the local core (the L1 in this work), and the first level of shared cache (commonly, the L2 cache) is physically distributed between the tiles of the system.

The cache coherence protocol will be a key design issue in these architectures since it will add requirements of area and energy consumption to the final design, and therefore, could restrict severely its scalability. When the number of cores is large, as is the case of many-core CMPs, the best way of keeping cache coherence is by implementing a directory-based protocol, which reduces energy consumption compared to broadcast-based protocols by keeping track of the caches that hold copies of each block in a directory structure. In tiled CMPs, the directory structure is distributed between the L2 cache banks, usually included into the L2 tags' portion [16]. In this way, each tile keeps the sharing information of the blocks mapped to the L2 cache bank that it contains. This sharing information comprises two main components[1]: the *state bits* used to codify one of the three possible states the directory can assign to the line (*Uncached*, *Shared* and *Private*), and the *sharing code*, that holds the list of current sharers. Most of the bits of each directory entry are devoted to codifying the sharing code. Since the directory must be stored as part of the on-chip L2 cache, it is desirable that its size be kept as low as possible. Moreover, a hard to scale directory organization could require to re-design the L2 cache to adapt the tile to the range of cores that is expected for the CMP.

In a traditional directory organization, each directory entry keeps track of the sharers of the corresponding memory block through a simple bit-vector (one bit per private cache). In Figure 1, we plot the area (in mm^2) that one 1MB 4-way L2 module would take as the number of cores grows from 2 to 256 (area estimations are based on CACTI. Refer to Section 4 for more details). As it can be seen, while the number of cores keeps below 16 the bit-vector sharing code barely impacts area requirements. However, from 16 cores on, the use of bit-vectors would entail too much area overhead and more area efficient sharing codes would be required.

One approach for reducing directory area requirements in the context of traditional shared-memory multiprocessors is the use of compressed sharing codes. Compressed sharing codes store the full directory information in a compressed way to use fewer number of bits, introducing a loss of precision compared to *exact* ones[2]. This means that when this information is reconstructed, some of the cores codified in the sharing code are real sharers and must receive the coherence

[1] Apart from other implementation-dependent bits.
[2] Bit-vector is an example of *exact* sharing code.

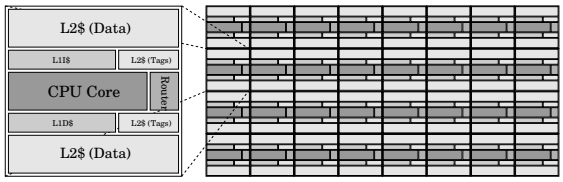


Fig. 2: Organization of the tile assumed in this work and a 4×8 tiled CMP

messages, whereas some other cores are not sharers actually and *unnecessary* coherence messages will be sent to them. *Unnecessary* coherence messages lead to increased miss latencies, since more messages are required to resolve caches misses. Moreover, *unnecessary* coherence messages also entail extra traffic in the interconnection network and useless cache accesses, which will increase energy consumption. Conversely, a bit-vector directory does not generate unnecessary coherence messages and thus shows the best results in terms of both performance and energy consumption.

In this work we study the effects on performance, network traffic and area required by the directory structure that the use of compressed sharing codes will have in many-core CMPs. In particular, we select two compressed sharing codes previously proposed by us in the context of large-scale shared-memory multiprocessors, Binary Tree (BT) and Binary Tree with Symmetric Nodes (BT-SN) [2], and that have very small area requirements. Simulation results of 32-core CMPs show that degradations of up to 32% in performance and 350% in network traffic are experienced. Additionally, since some proposals for efficient multicast support in on-chip networks have recently appeared [11], we also consider the case of using this kind of support in combination with the compressed sharing codes. Unfortunately, multicast support is not enough to remove completely the performance degradation that the compressed sharing codes introduce (performance degradations of 10% on average are still observed when BT is used) and barely can reduce network traffic.

The rest of the paper is organized as follows. First of all, we will give more details regarding the target CMP architecture in Section 2. Subsequently, in Section 3 we will present a couple of compressed sharing codes based on the concept of multilayer clustering that have small overhead in terms of area. Next, in Section 4, we will describe the evaluation environment that we are assuming, and the results of the evaluation will be shown in Section 5. Finally, Section 6 closes the work and points future directions to be explored.

2 Base Architecture

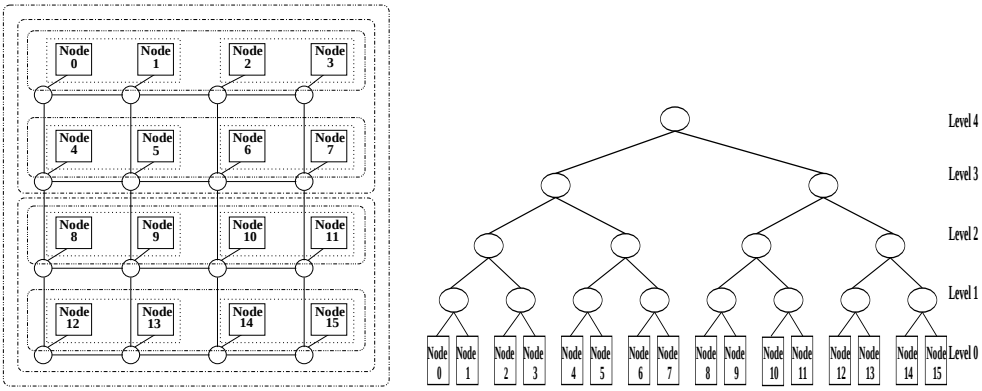
A tiled CMP architecture consists of a number of replicated *tiles* connected over a switched direct network. Each tile contains a processing core with primary caches (both instruction and data caches), a slice of the L2 cache, and a connection to the on-chip network. Cache coherence is maintained at the L1 caches. In particular, it is employed a directory-based cache coherence protocol, with directory information stored in the tags' part of the L2 cache modules. The L2 cache is shared among the different processing cores, but it is physically distributed between 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 [5]). Moreover, for simplicity the L1 and L2 caches are inclusive, that is to say, all the blocks included in any L1 cache keep an entry in the L2 cache. Figure 2 shows the organization of a tile (left) and a 16-tile CMP (right). From now on, we will use the terms tile and node interchangeably.

3 Multi-layer Clustering Concept

This section presents two compressed sharing code organizations based on the *multi-layer clustering* approach previously proposed in [2].

Multi-layer clustering assumes that nodes are recursively grouped into clusters of equal size until all nodes are grouped into a single cluster. Compression is achieved by specifying the smallest cluster containing all the sharers (instead of indicating *all* the sharers). Compression can be increased even more by indicating only the level of the cluster in the hierarchy. In this case, it is assumed that the cluster is the one containing the home node for the memory block. This approach is valid for any network topology.

Although clusters can be formed by grouping any integer number of clusters in the immediately lower layer of the hierarchy, we analyze the case of using a value equal to two. That is to say, each cluster contains two clusters from the immediately lower level. By doing so, we simplify binary representation and obtain better granularity to specify the set of sharers. This recursive grouping into layer clusters leads to a logical binary tree with the nodes located at the leaves.



(a) *Physical* system (b) *Logical* system

Fig. 3: Multi-layer clustering approach example

As an application of this approach, two compressed sharing codes were previously proposed in [2]. The sharing codes can be shown graphically by considering the distinction between the *logical* and the *physical* organizations. For example, we have a 16-tile CMP with a mesh as the interconnection network, as shown in Figure 3(a), and we can imagine the same system as a binary tree (multi-layer system) with the nodes located at the leaves of this tree, as shown in Figure 3(b). Note that this tree only represents the grouping of nodes, not the interconnection between them. In this representation, each subtree is a cluster. Clusters are also shown in Figure 3(a) by using dotted lines. It can be observed that the binary tree is composed of 5 layers or levels ($\log_2 N + 1$, where N is a power of 2). From this, the following two compressed sharing codes were derived in [2]: *Binary tree* (BT) and *Binary tree with symmetric nodes* (BT-SN).

3.1 Binary Tree (*BT*)

Since nodes are located at the leaves of a tree, the set of nodes (sharers) holding a copy of a particular memory block can be expressed as the minimal subtree that includes the home node

Table 1: System parameters

32-core CMP			
GEMS Parameters		**SICOSYS Parameters**	
Processor frequency	4 GHz	Network frequency	2 GHz
Cache hierarchy	Inclusive	Topology	8x4 Mesh
Cache block size	64 bytes	Switching technique	Wormhole, Multicast
Split L1 I & D caches	128KB, 4 ways,	Routing technique	Deterministic X-Y
	4 hit cycles	Message size	4 flits data, 1 flit control
Shared unified L2 cache	1MB/tile, 4 ways,	Routing time	2 cycles
	7 hit cycles	Link latency (one hop)	2 cycles
Memory access time	300 cycles	Link bandwidth	1 flit/cycle

and all the sharers. This minimal subtree is codified using the level of its root (which can be expressed using just $\lceil \log_2 (\log_2 N + 1) \rceil$ bits). Intuitively, the set of sharers is obtained from the home node identifier by changing the value of some of its least significant bits to *don't care*. The number of modified bits is equal to the level of the above mentioned subtree. It constitutes a very compact sharing code (observe that, for a 128-node system, only 3 bits per directory entry are needed). For example, consider a 16-node system such as the one shown in Figure 3(a), and assume that nodes 1, 4 and 5 hold a copy of a certain memory block whose home node is 0. In this case, node 0 would store 3 as the tree level value, which is the one covering all sharers (see Figure 3(b)). Unfortunately, this would include as well nodes 0, 2, 3, 6 and 7 that do not have copy of such memory block and that, thus, would receive unnecessary coherence messages on a subsequent coherence event.

3.2 Binary Tree with Symmetric Nodes (*BT-SN*)

We also introduce the concept of symmetric nodes of a particular home node. Assuming that 3 additional symmetric nodes are assigned to each home node, they are codified by different combinations of the two most-significant bits of the home node identifier (note that one of these combinations represents the home node itself). In other words, symmetric nodes only differ from the corresponding home node in the two most significant bits. For instance, if 0 were the home node, its corresponding symmetric nodes would be 4, 8 and 12. Now, the process of choosing the minimal subtree that includes all the sharers is repeated for the symmetric nodes. Then, the minimum of these subtrees is chosen to represent the sharers. The intuitive idea is the same as before but, in this case, the two most significant bits of the home identifier are changed to the symmetric node used. Therefore, the size of the sharing code of a directory entry is the same as before plus the number of bits needed to codify the symmetric nodes (for 3 sym-nodes, 2 bits). In the previous example, nodes 4, 8 and 12 are the symmetric nodes of node 0. The tree level could now be computed from node 0 or from any of its symmetric nodes. In this way, the one which encodes the smallest number of nodes and includes nodes 1, 4 and 5 is selected. In this particular example, the tree level 3 must be used to cover all sharers, computed from node 0 or node 4.

4 Evaluation environment

We perform the evaluation using the full-system simulator Virtutech Simics [8] extended with Multifacet GEMS 1.3 [9], that provides a detailed memory system timing model. Since the

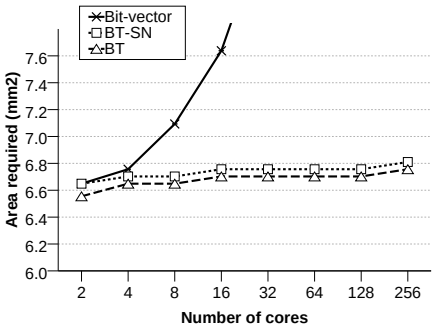


Fig. 4: Area (mm^2) required for a 1MB cache module when bit-vector, BT or BT-SN are used

network modeled by GEMS 1.3 is not very precise, we have extended it with SICOSYS [10], a detailed interconnection network simulator. We simulate a 32-tile CMP architecture as the one described in Section 2. The values of the main parameters used for the evaluation are shown in Table 1. Cache latencies have been calculated using the CACTI 5.3 tool [13] for 45nm technology. We also have used CACTI to measure the area of a 1MB 4-way L2 cache bank that includes the different sharing codes assumed in this work. In this study, we assume that the length of the physical address is 44 bits, like in the SUN UltraSPARC-III architecture [4].

The ten applications used in our simulations cover a variety of computation and communication patterns. *Barnes* (8192 bodies, 4 time steps), *FFT* (256K points), *Ocean* (258x258 ocean), *Radix* (1M keys, 1024 radix), *Raytrace* (teapot), *Volrend* (head) and *Water-Sp* (512 molecules, 4 time steps) are scientific applications from the SPLASH-2 benchmark suite [15]. *Unstructured* (Mesh.2K, 5 time steps) is a computational fluid dynamics application. *MPGdec* (525_tens_040.m2v) and *MPGenc* (output of *MPGdec*), are multimedia applications from the APLBench suite [7]. We account for the variability in multithreaded workloads by doing multiple simulation runs for each benchmark in each configuration and injecting random perturbations in memory systems timing for each run.

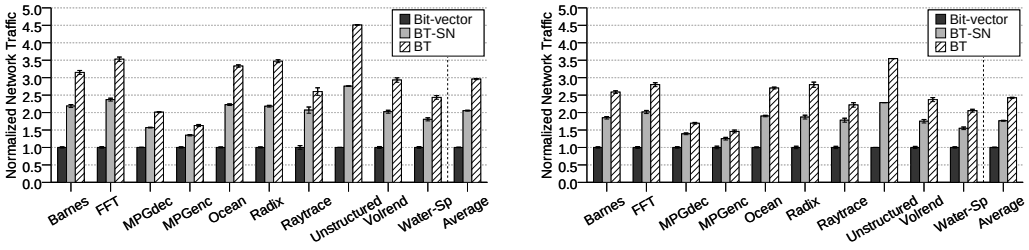
5 Evaluation results

We start this section by comparing the area overhead introduced by the different organizations for the sharing code considered in this work (i.e., bit-vector, BT and BT-ST). Next, we study the impact that the compressed sharing codes have on network traffic. For that, we consider both an interconnection network with and without multicast support. Finally, we end with a comparison between the three directory organizations in terms of the execution times that they obtain for the ten applications described in the last section.

5.1 Impact on area overhead

Figure 4 plots the total area (in mm^2) that would be required by a 1MB 4-way cache module when bit-vector, BT and BT-SN sharing codes are used. Due to the limited number of cores used in our simulations (32), we evaluate BT-SN assuming only one symmetric node. In this way, the size of BT-SN is equal to the size of BT plus 1 bit to codify whether the home node or the symmetric node is being used in the codification.

As shown in Figure 4 (and discussed in the introduction of this work), the area overhead that the bit-vector sharing code entails does not scale with the number of cores. Obviously,



(a) Without multicast (b) With multicast

Fig. 5: Normalized network traffic for bit-vector, BT and BT-SN

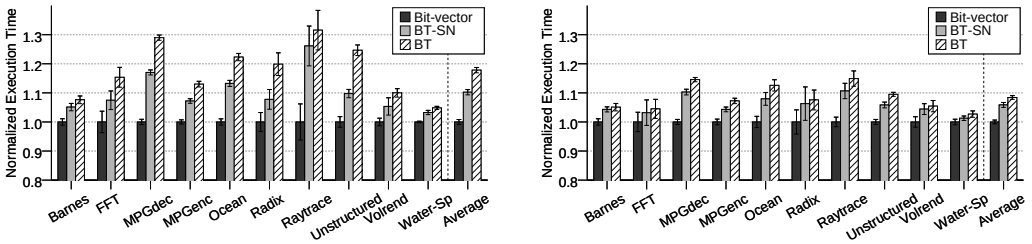
the size of the bit-vector (in bits) increases linearly with the number of cores. For this reason, the bit-vector could be a good option for a small number of cores. However, from 16 cores on the increase in area that the bit-vector conveys makes it unfeasible (the area overhead becomes almost 100% for the 64-core configuration). On the other hand, the size of BT and BT-SN barely increases with the number of cores. Moreover, the total number of bits needed by BT and BT-SN is very small in all cases ($\lceil \log_2 (\log_2 N + 1) \rceil$ bits and $\lceil \log_2 (\log_2 N + 1) \rceil + 1$ bits, respectively). In this way, the area overhead of BT and BT-SN is very low (less than 5% for the 256-core configuration) and keeps almost constant with the number of cores. This makes that BT and BT-SN can be considered as promising alternatives to bit-vector for future may-core CMPs, since besides introducing very small overheads in terms of area, these sharing codes would allow to support families of CMPs with varying number of cores and using exactly the same tile (without requiring any modifications in the directory structure).

5.2 Impact on network traffic

Although compressed sharing codes can drastically reduce the size of the directory, their counterpart is that they could increase the number of coherence messages as a consequence of the in-excess codification of the sharers that they perform. Increasing the number of coherence messages leads to more traffic being injected in the interconnection network of the CMP. Since previous works have identified the interconnection network as one of the most important elements of the CMP from the point of view of energy consumption (consuming almost 40% of the total energy budget in the Raw processor [14]), more traffic at the end means more energy.

Figure 5 shows the amount of network traffic that would be generated for bit-vector, BT and BT-SN for the 32-core CMP configuration assumed in this work. 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 bit-vector case. We present results considering both a network with unicast support (a) and with multicast support (b).

As shown in Figure 5(a), the use of BT has severe impact on the amount of network traffic and degradations ranging from approximately 50% for *MPGenc* to 350% for *Unstructured* are found. The problem with BT is that when one of the sharers is far from the home node in the logical tree structure illustrated in Figure 3(b), the root of the tree is selected as the minimum tree level covering both the home node and the sharer, which results in all cores being actually codified. We have found that this situation occurs frequently in most applications, which explains the significant amount of extra traffic for BT. In particular, the average number of coherence



(a) Without multicast

(b) With multicast

Fig. 6: Execution time for 32 cores

messages that are sent on a coherence event[3] increases from 2 in bit-vector to more than 20 in BT. On the contrary, when BT-SN is considered the tree level that covers all the sharers can be computed from either the home node or its symmetric node. This leads to noticeable reductions in the average number of coherence messages (12 in BT-SN), which leads to important savings in network traffic when compared with BT. Unfortunately, BT-SN does not mitigate completely the extra traffic introduced by BT and degradations of approximately 100% on average are still observed. Again, when two or more cores, distant in the logical tree, share a memory block, the root of the tree would be codified by BT-SN.

Obviously, the provision of multicast support at the interconnection network level can alleviate the levels of extra traffic. More specifically, in Figure 5(b) we show the results obtained when we take advantage of multicast support for sending coherence messages (invalidations and cache-to-cache transfer commands). Efficient implementations of such kind of multicast support in on-chip networks have recently been proposed [11]. Unfortunately, using multicast support for factorizing efficiently also the response messages is not a trivial issue. So, in this work we assume that responses to coherence commands are unicast messages. As it can be seen, the use of multicast support is an step forward in achieving the network traffic levels obtained by bit-vector, and it is especially useful when BT is considered (average traffic overhead is reduced from 200% without multicast support to 150%). Anyway, the fact that multicast support is available just for the coherence commands and not for their associated responses limits its benefits.

5.3 Impact on execution time

The degradations previously reported in terms of network traffic finally translate into increases in terms of execution time. In Figure 6 we show how the use of BT and BT-SN impacts applications' execution times, considering an interconnection network with and without multicast support, (a) and (b) respectively. Again, all results have been normalized with respect to the bit-vector case.

As observed in Figure 6(a), the use of BT without multicast support has important consequences on performance. In particular, the execution time grows from less than 10% for *Barnes* and *Water-Sp* to more than 30% for Raytrace (19% on average). In general, the greater number of messages that are needed with BT to resolve every coherence event leads to longer cache miss latencies, and therefore, execution times. Obviously, the extent of the degradation in execution time will depend on the particular characteristics of each application (L1 cache miss rate, average number of coherence messages per cache miss, kind of synchronization used, etc.). This is why there is no direct correlation between the amount of extra traffic reported in Figure 5(a) and the

[3] By coherence event we refer to a situation where the home node must use the sharing code to send coherence messages (invalidations or cache-to-cache transfer commands).

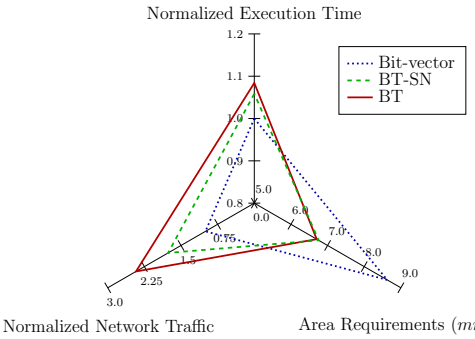


Fig. 7: Trade-off between area, performance and network traffic for BT, BT-SN and bit-vector (32 cores and multicast support are assumed)

degradation in execution time shown in Figure 6(a). On the other hand, when BT-SN is used instead of BT, the average overhead in terms of execution time is reduced to a half (10%). In this case, significant reductions in execution time are observed for most applications. The exceptions are *Barnes* and *Water-Sp*, that hardly see their execution times reduced when BT-SN is used, even when significant savings in terms of network traffic were reported.

The effects of using multicast support with BT and BT-SN are analyzed in Figure 6(b). As before, multicast support has significant impact on execution time when BT is assumed. In this case, average degradation falls from 19% to less than 10%. Although all applications benefit from multicast support, *FFT*, *MPGdec*, *Radix*, *Raytrace* and *Unstructured* are the most affected (in all these cases performance degradation entailed by BT is reduced to more than a half). Finally, and as it was reported for network traffic, multicast support does not help much in reducing performance overhead when BT-SN is considered. In this case, what dominates cache miss latencies is the time taken to collect all responses to a coherence event, which is not optimized with the assumed multicast support.

6 Conclusions and Future Work

The organization of the directory needed to maintain cache coherence will be a key design point in future many-core CMPs. In this work we have analyzed the effects that the BT and BT-SN compressed sharing codes have on area, network traffic (as representative of the energy consumed in the interconnection network) and performance in the context of many-core chip-multiprocessors. In particular, we have found that although very area-efficient directories could be derived based on these two sharing codes (with area overheads of less than 5%), the degradations in terms of network traffic (200% for BT and 100% for BT-SN) as well as execution time (20% for BT and 10% for BT-SN) that they entail could preclude them from being employed in future many-core CMPs. Moreover, we have studied the case of having an interconnection network with multicast support, and have found that although BT can significantly benefit from such kind of support (degradations in execution time and network traffic are reduced to 8% and 150% respectively), BT-SN barely finds any benefits from it. The reasons why multicast support is unable to hide the degradation that BT and BT-SN introduce are two. First, multicast support is only used for sending coherence commands but not for collecting the responses. An second, even when an efficient mechanism able to provide combined responses were used, more destinations for the coherence commands still implies more traffic and longer cache miss latencies. As a summary of the results, Figure 7 shows the trade-off between area, performance and network traffic for the sharing codes evaluated in this work.

Our future work includes new organizations for the sharing code aimed at reducing the amount of unnecessary coherence messages that BT-SN entails but having similar requirements in terms of area. Additionally, we are studying the possibility of including support in the interconnection network for discarding unnecessary coherence messages as they travel to their destination. Finally, we are extending our simulation tools to compare the different directory organizations in terms of their energy requirements (considering both static and dynamic energy consumption).

References

1. Single-chip Cloud Computer. *http://techresearch.intel.com/articles/Tera-Scale/1826.htm*.
2. M. E. Acacio, J. González, J. M. García, and J. Duato. A new scalable directory architecture for large-scale multiprocessors. In *7th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, pages 97–106, Jan. 2001.
3. S. Borkar. Thousand core chips: A technology perspective. In *44th Annual Design Automation Conference*, pages 746–749, June 2007.
4. T. Horel and G. Lauterbach. UltraSPARC-III: Designing third-generation 64-bit performance. *IEEE Micro*, 19(3):73–85, May 1999.
5. C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th Int. Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, pages 211–222, Oct. 2002.
6. J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, and C. Kozyrakis. Comparing memory systems for chip multiprocessors. In *34th Int'l Symp. on Computer Architecture (ISCA)*, pages 358–368, June 2007.
7. M.-L. Li, R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The ALPBench benchmark suite for complex multimedia applications. In *Int'l Symp. on Workload Characterization*, pages 34–45, Oct. 2005.
8. P. S. Magnusson, M. Christensson, and J. Eskilson, et al. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.
9. M. M. Martin, D. J. Sorin, and B. M. Beckmann, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, 33(4):92–99, Sept. 2005.
10. 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.
11. S. Rodrigo, J. Flich, J. Duato, and M. Hummel. Efficient unicast and multicast support for CMPs. In *41st IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, pages 364–375, Nov. 2008.
12. M. B. Taylor, J. Kim, and J. Miller, et al. The raw microprocessor: A computational fabric for software circuits and general purpose programs. *IEEE Micro*, 22(2):25–35, May 2002.
13. S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. Technical Report HPL-2008-20, HP Labs, Apr. 2008.
14. 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)*, pages 105–111, Dec. 2003.
15. 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)*, pages 24–36, June 1995.
16. M. Zhang and K. Asanović. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *32nd Int'l Symp. on Computer Architecture (ISCA)*, pages 336–345, June 2005.