

## DASC-DIR: A Low-Overhead Coherence Directory for Many-Core Processors

Alberto Ros · Manuel E. Acacio

Received: date / Accepted: date

**Abstract** Current trends point towards future many-core processors being implemented using the hardware-managed, implicitly-addressed, coherent caches memory model. With this memory model, all on-chip storage is used for private and shared caches that are kept coherent by hardware. Communication between cores is performed by writing to and reading from shared memory, and a scalable point-to-point interconnection network is in charge of transmitting messages. Cache coherence in this context is guaranteed by means of a directory-based protocol. Unfortunately, it has been previously shown that the directory structure required to keep track of sharers can restrict the scalability of these designs due its excessive area or energy requirements, or for a compressed directory, the increased coherence traffic that in some cases it could cause.

On the other hand, in many-core architectures, memory blocks are commonly assigned to the banks of a NUCA shared cache by following a physical mapping. This mapping assigns blocks to cache banks in a round-robin fashion, thus neglecting the distance between the cores that more frequently access every block and the corresponding NUCA bank for the block. This issue impacts both cache access latency and the amount of on-chip network traffic generated, and causes that some area- and energy-efficient compressed directories significantly increase the number of messages per coherence event, which finally translates into degraded performance.

In this work we propose an efficient and low-overhead coherence directory which is built around two main ingredients: the first is the use of the *distance-aware round-robin* mapping policy, an OS-managed policy which tries to map the pages accessed by a core to its closest (local) bank, at the same time it introduces an upper bound on the deviation of the distribution of memory pages among cache banks, which lessens the number of off-chip accesses. The second is the utilization of a very compressed directory structure which takes

advantage of this mapping policy to represent sharers in a very compact way without increasing coherence network traffic.

Simulation results for a 32-core architecture demonstrate that compared to a full-map directory using the typical round-robin physical mapping policy, our proposal drastically reduces the size of the directory structure (and thus, its area and energy requirements) at the same time it does not increase coherence network traffic and 6% average savings in execution time are achieved.

**Keywords** Many-core CMPs · Dynamic home assignment · Compressed sharing codes · Energy consumption · Execution time · Area overhead · Network traffic

## 1 Introduction

Ever-increasing power consumption and the diminishing returns in performance of single-core architectures led to the advent of general-purpose multi-core chips (or chip-multiprocessors or CMPs) during the last decade [1]. Since then, most processor manufacturers have joined this multi-core wave, developing products with an ever-increasing number of cores. With a growing number of transistors available at each new technology generation, coupled with a reduction in design complexity enabled by its modular design, multi-core chips look set to stay.

Multi-core architectures that integrate several tens of processor cores (usually known as many-core architectures) are already a reality in the commercial arena –an example is the 60-core Intel Xeon Phi processor [2]– and the number of cores is expected to keep growing, which may lead to hundreds and even thousands of cores integrated on a single chip [3]. In order to organize such a big number of cores, tiled multi-core architectures have been advocated as the most feasible organization. Tiled chip-multiprocessors are designed as arrays of identical or close-to-identical building blocks (tiles). In these architectures, each tile is comprised by a core, one or several levels of caches, and a network interface that connects all tiles through a scalable point-to-point interconnection network. Regarding the communication model, current trends point towards future many-core processors being implemented using the hardware-managed, implicitly-addressed, coherent caches memory model. With this memory model, all on-chip storage is used for private and shared caches that are kept coherent by hardware. Communication between cores is performed by writing to and reading from shared memory, and a directory-based cache coherence protocol implemented in hardware is in charge of ensuring the consistency of data stored in private caches. Figure 1 shows the organization of a 16-core tiled CMP with per core private L1 caches for instructions (L1I\$) and data (L1D\$) and a physically distributed but logically shared L2 cache (L2\$). Without loss of generality, this is the cache hierarchy assumed in this work. This way, from now on we will use the terms last-level cache (LLC) and L2 cache interchangeably.

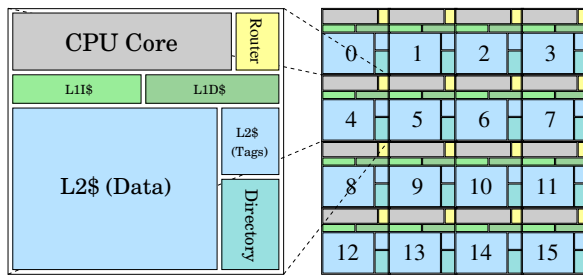


Fig. 1: Architecture of a tiled CMP

The cache coherence protocol in a many-core architecture becomes a key design issue since it adds requirements of area and energy consumption to the final design, and therefore, could restrict severely its scalability. The directory structure is distributed between the last-level shared cache banks, usually within the tags' portion [4]. 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 (apart from other implementation-dependent bits): the *state bits* used to codify one of the three possible states the directory can assign to the block (*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 is commonly 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 full-map sharing code (one bit per private cache). Unfortunately, this sharing code is only feasible for a handful of cores due to the excessive area requirements that it introduces when the number of cores is large. On the other hand, compressed sharing codes, whose size does not grow linearly with the number of cores, drastically reduce area overhead at expense of increasing coherence traffic, and therefore, degrading performance and harming energy consumption.

Besides the organization of the coherence directory, another important design issue in a many-core chip is the distribution of the memory blocks among the different tiles. This aspect directly affects L2 cache access latency, since it depends on the bank wherein the block is allocated, *i.e.*, the *home* bank or tile.

The most straightforward way of distributing blocks among the different tiles is by using a physical mapping policy in which a set of bits in the block address defines the home bank for every block. Most commercial CMPs [5, 6] choose the less significant bits of the block address for selecting the home bank. In this way, blocks are assigned to banks in a round-robin fashion with block-size granularity. This distribution of blocks does not take into account the

distance between the requesting core and the home bank on a L1 cache miss. Moreover, the average distance between two tiles significantly increases with the size of the CMP, which can become a performance problem for many-core CMPs.

On the other hand, page-size granularity seems to be a better choice than block-size granularity for future tiled CMPs because (1) it is more appropriate for new technologies aimed to reduce off-chip latencies, like 3D stacking memory architectures [7], and (2) it provides flexibility to the OS for implementing more efficient mapping policies [8,9], such as *first-touch*, which has been widely used in NUMA architectures to achieve more locality in the memory accesses. The behavior of a first-touch policy is similar to a private cache organization but without replication. One nice aspect of this policy is that it is dynamic in the sense that pages are mapped to cache banks depending on the particular memory access pattern. However, this policy can increase off-chip accesses when the working set of the application is not well-balanced among cores.

Additionally, many-core architectures are very suitable for throughput computing [10] and, therefore, they constitute a highly attractive choice for commercial servers in which several programs are running at the same time using different subsets of the cores available on chip. The use of these architectures as commercial servers emphasize the need of efficient mapping policies because (1) data is shared by cores that are placed in a small region of the chip, but with a round-robin policy they could map to any bank in the chip, and (2) more working set imbalance can occur in these systems since the applications running on them could have very different memory requirements.

In this work, we propose DASC-DIR, an efficient and low-overhead coherence directory which is built around two main ingredients. The first is the use of the *distance-aware round-robin* mapping policy [11], an OS-managed policy which without any extra hardware structures tries to map the pages accessed by a core to its closest (local) bank, at the same time it introduces an upper bound on the deviation of the distribution of memory pages among cache banks, which lessens the number of off-chip accesses.

The second ingredient is the utilization of a very compressed directory structure which takes advantage of this mapping policy to represent sharers in a very compact way without increasing coherence network traffic, and that we call *distance-aware sharing code* or DASC.

This way, this paper extends our previous work [11] by removing the scalability limit imposed by the full-map sharing code assumed in [11] through the use of several compressed sharing codes. Additionally, this paper presents and evaluates for the first time our *distance-aware sharing code*, which is especially designed to be used in conjunction with the *distance-aware round-robin* mapping policy.

Simulation results for a 32-core architecture demonstrate that compared to a full-map directory using the typical round-robin physical mapping policy, our proposal drastically reduces the size of the directory structure (and thus, its area and energy requirements) at the same time it does not increase coherence network traffic and 6% average savings in execution time is achieved.

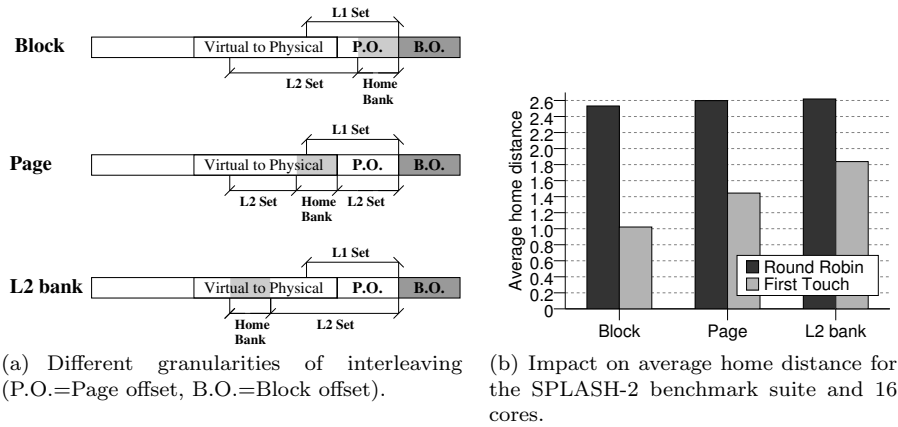


Fig. 2: Granularity of L2 cache interleaving and its impact on average home distance

The rest of the paper is organized as follows. A background on mapping policies for NUCA caches is given in Section 2. In this section we also describe two compressed sharing codes already appeared in the literature. Section 3 describes the distance-aware round-robin mapping policy and the impact of distance-aware mapping policies on private cache miss rate. The distance-aware sharing code is discussed in Section 4. Section 5 introduces the methodology employed in the evaluation. Section 6 shows the performance results. Section 7 presents a review of the related work and, finally, Section 8 concludes the paper.

## 2 Background

### 2.1 Mapping policies in NUCA caches

Non-uniform cache access (NUCA) caches [12] are a set of cache banks distributed across the chip and connected through a point-to-point network. Although cache banks are physically distributed, they constitute a logically shared cache (the L2 cache level in this work). Therefore, the mapping of memory blocks to cache entries is not only defined by the cache set, but also by the cache bank. The cache bank where a particular block maps is called the *home* bank for that block.

Most CMP architectures that implement NUCA caches map memory blocks to cache banks by taking some fixed bits of the physical address of the block [5, 6]. This physical mapping uniformly spreads blocks among cache banks, resulting in optimal utilization of the cache storage. Commonly, the bits taken to select the cache bank for a particular block are the less significant ones, leading to a block-grained interleaving (*Block* diagram in Figure 2(a)). One of

the advantages of this interleaving is that it offers less contention at the home tile by distributing contiguous memory blocks across different cache banks.

Another option is to use an interleaving with a granularity of at least the size of a page (*e.g.*, *Page* or *L2 bank* diagram in Figure 2(a)). As shown in Figure 2(b), when a physical mapping, or round-robin, policy is considered the granularity of the interleaving does not significantly affect the average distance to the home bank. However, this interleaving becomes an important decision when either 3D stacked memory or OS-managed mapping techniques are considered.

A 3D stacked memory design can offer latency reductions for off-chip accesses when a coarse-grained interleaving (at least of page size) is employed. In tiled CMPs with 3D stacking memory, each tile includes a memory controller for the memory bank that it handles [7]. Low-latency, high-bandwidth and very dense *vertical* links [13] interconnect the on-chip controller with the off-chip memory. These vertical links provide fast access to main memory. On a L2 cache miss, it is necessary to reach the memory controller of the memory bank where the block is stored. If the memory controller is placed in a different tile than the home L2 bank, a *horizontal* on-chip communication is entailed. Since blocks in memory are handled at page-size granularity, it is not possible to assign the same mapping for the L2 cache if a block-size granularity is considered. Differently, with a granularity of at least the size of a page the same mapping can be assigned to both memories, thus avoiding the horizontal latency.

The other advantage of a coarse-grained interleaving is that it allows the OS to manage the cache mapping without requiring extra hardware support [8]. The OS maps a page to a particular bank the first time the page is referenced, *i.e.*, a memory miss. At that moment, the OS assigns a physical address to the virtual address of the page. Therefore, some bits in the address of the page are going to change (*Virtual to Physical* field in figure 2(a)). Then, the OS can control the cache mapping by assigning to this page a physical address that maps to the desired bank. For example, a first-touch policy can be easily implemented by assigning an address that physically maps to the tile wherein the core that is accessing the page resides. The OS only needs to keep in software a list of available physical addresses for each memory bank. With a first-touch mapping policy, finer granularity offers shorter average distance between the missing L1 cache and the home L2 bank, as shown in Figure 2(b). Therefore, it is preferable to use a grain size as fine as possible. Since block granularity is not suitable for OS-managed mapping, the finest granularity possible is achieved by taking the less significant bits of the *Virtual to Physical* field, *i.e.*, a page-grained interleaving.

The drawback of a first-touch policy is that applications with a working set not balanced among cores do not make optimal use of the total L2 capacity. This happens more frequently in commercial servers where different applications with different memory requirements run on the same system, or when some applications are running in a set of cores while the other cores remain idle. To avoid this situation, policies like *cache pressure* [8] can be im-

plemented. Cache pressure uses bloom filters to collect cache accesses in order to determine the *pressure* of the different data mapping to cache banks. In this way, newly accessed pages are not mapped to the most pressured caches. However, this approach has several drawbacks. First, it requires extra hardware, (*e.g.*, bloom filters that have to be reset after a timeout period). Second, an efficient function to detect the pressured cache banks can be difficult to implement. Third, this mechanism only considers neighbouring banks, *i.e.*, banks at 1-hop distance. Finally, as far as we know, neither parallel nor multi-programmed workloads have been evaluated using this technique.

## 2.2 BT and BT-SN Compressed Sharing Codes

One approach for reducing area requirements in the context the directory-based cache coherence protocols typically employed in tiled CMPs is the use of compressed sharing codes. Compressed sharing codes store the directory information in a compressed way to use fewer number of bits, introducing a loss of precision compared to *exact* ones (*e.g.*, full-map). 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 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 full-map directory does not generate unnecessary coherence messages and thus shows the best results in terms of both performance and energy consumption.

Among the compressed sharing codes previously proposed in the literature, we consider in this work two organizations (BT and BT-SN) previously proposed in [14]. Both compressed sharing codes are based on the *multi-layer clustering* approach.

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.

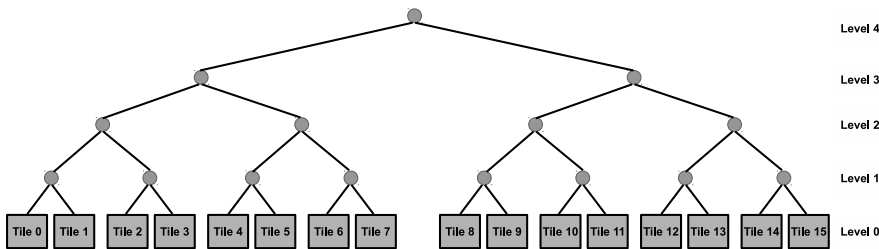


Fig. 3: Multi-layer clustering approach: logical view

As an application of this approach, two compressed sharing codes were previously proposed in [14]. The sharing codes can be shown graphically by considering the distinction between the *logical* and the *physical* organizations. For example, we have the 16-tile CMP with a mesh as the interconnection network previously shown in Figure 1, and we can imagine the same system as a binary tree (multi-layer system) with the tiles (nodes) located at the leaves of this tree, as shown in Figure 3. Note that this tree only represents the grouping of nodes, not the interconnection between them. In this representation, each subtree is a cluster. 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 [14]: *Binary tree* (BT) and *Binary tree with symmetric nodes* (BT-SN).

### 2.2.1 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 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 1, 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). Unfortunately, this would include as well nodes 0, 2, 3, 6 and 7 that do not have a copy of such memory block and thus would receive unnecessary coherence messages on a subsequent coherence event.



### 2.2.2 Binary Tree with Symmetric Nodes (BT-SN)

This sharing code introduces 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.

As it has been shown, in both BT and BT-SN, the set of sharers is calculated from the home of each memory block. This makes these sharing codes especially attractive for use in conjunction with the OS-managed cache mapping strategy proposed in this work.

## 3 DARR: Distance-Aware Round-Robin Mapping

Distance-aware round-robin mapping (DARR) [11] is a simple OS-managed mapping policy for many-core CMPs that assigns memory pages to NUCA cache banks. This policy minimizes the total number of off-chip accesses as happens with a round-robin mapping, and reduces the access latency to a NUCA cache (the L2 cache level in this work) as a first-touch policy does. Moreover, this policy addresses this trade-off without requiring any extra hardware support.

The OS starts assigning physical addresses to the requested pages according to a first-touch policy, i.e, the physical address chosen by the OS maps to the tile of the core that is requesting the page. The OS stores a counter for each cache bank which is increased whenever a new physical page is assigned to this bank. In this way, banks with more physical pages assigned to them will have higher value for the counter.

To minimize the amount of off-chip accesses we define an upper bound on the deviation of the distribution of pages among cache banks. This upper bound can be controlled by the OS through a threshold value. In this way, in case that the counter of the bank where a page should map following a first-

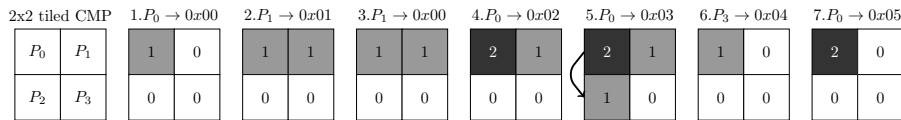


Fig. 4: Example of the distance-aware round-robin mapping policy

touch policy has reached the threshold value, the page is assigned to another bank. The algorithm starts checking the counters of the banks at one hop from the initial placement. The bank with smaller value is chosen. Otherwise, if all banks at one hop have reached the threshold value, then the banks at a distance of two hops are checked. This algorithm iterates until a bank whose value is under the threshold is found. The policy ensures that at least one of the banks has always a value smaller than the threshold value by decreasing by one unit all counters when all of them have values different than zero.

Figure 4 shows, from left to right, the behavior of this mapping policy for a  $2 \times 2$  tiled CMP with a threshold value of two. First, processor  $P_0$  accesses a block within page  $0x00$  which faults in memory (1). Therefore, a physical address that maps to the bank 0 is chosen for the address translation of the page, and the value for the bank 0 is increased. Then, processor  $P_1$  perform the same operation for page  $0x01$  (2). When processor  $P_1$  accesses page  $0x00$  no action is required for our policy because there is a hit in the page table (3). The next access of processor  $P_0$  is for a new page, which is also stored in bank 0, which reaches the threshold value (4). Then, if processor  $P_0$  accesses a new page again, this page must be allocated in another bank (5). The closer bank with a smaller value is bank 2. Finally, when processor  $P_3$  accesses a new page, the page is assigned to its local bank and all counters are decreased (6), allowing bank 0 to map a new page again (7).

The threshold defines the behavior of our policy. A threshold value of zero denotes a round-robin policy in which a uniform distribution of pages is guaranteed, while an unlimited threshold implies a first-touch policy. Therefore, with a small threshold value, our policy reduces the number of off-chip accesses. Otherwise, if the threshold value is high, our policy reduces the average latency of the accesses to the NUCA cache. Note that the threshold value serves as a proxy approximation for the cache pressure since the actual pressure does not directly depend on the uniform distribution of pages, but on the utilization of blocks within pages. However, pages are distributed among all cache banks, thus performing an efficient use of the shared cache. Although, the OS could choose different thresholds depending on the workload, we have found that values between 64 and 256 work well for the workloads considered in this work.

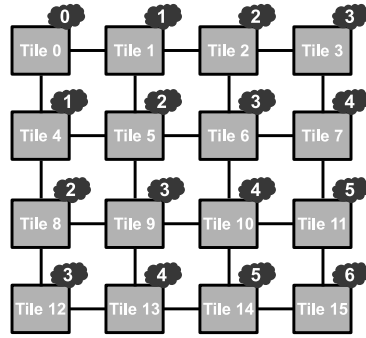


Fig. 5: Distance from tile 0 to the rest of tiles

#### 4 DASC: Distance-Aware Sharing Code

In this section we propose DASC, a new sharing code especially suited for the DARR mapping policy previously described. DASC shares some characteristics with the previously described BT and BT-SN sharing codes. First, DASC is a compressed sharing code with a very extreme compression ratio, thus ideal for scalable systems. In a compressed sharing code more cores than necessary may be codified, *i.e.*, false positives can appear. Due to these false positives unnecessary coherence messages may arise. Second, as in BT and BT-SN, the representation of the set of sharers in DASC is computed from the home node of each memory block.

DASC is especially suited for the DARR mapping policy (and in general for first-touch mapping policies) because it takes advantage of the fact that the DARR mapping policy tends to locate memory blocks closer to the tiles that use them. Note that both BT, BT-SN and DASC compute the sharers from the home tile. If the home tile is one of the frequent sharers, important reductions in extra invalidation messages can be achieved.

The idea behind DASC is to codify the set of sharers as the distance (number of network links) between the home tile and the farthest sharer. Then, all nodes within this distance are considered potential sharers. To implement this, we use a saturating counter with a fixed number of bits (very small) and reserve the greater binary value to denote the situation in which all nodes must be included, *i.e.*, when the counter is saturated.

As an example, Figure 5 shows the distance (in number of links) between home tile 0 and the rest of tiles for a 16-core CMP with a two-dimensional mesh topology. Table 1 depicts the tiles that would be included for the possible different combinations of both a 2-bit and a 3-bit DASC. For example, the fact that nodes 1, 4, and 5 hold a copy of a certain memory block whose home node is 0 would be codified in DASC using value 2. This would include as well nodes 0, 2, and 8 that do not have a copy of such memory block and thus would receive unnecessary coherence messages on a subsequent coherence event.

Table 1: Codification examples using DASC in a 16-tile CMP with a two-dimensional mesh, from the point of view of the home tile 0

2 bits		3 bits	
Value	Covered tiles	Value	Covered tiles
0	{0}	0	{0}
1	{0,1,4}	1	{0,1,4}
2	{0,1,2,4,5,8}	2	{0,1,2,4,5,8}
3	All nodes	3	{0,1,2,3,4,5,6,8,9,12}
		4	{0,1,2,3,4,5,6,7,8,9,10,12,13}
		5	{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14}
		6-7	All nodes

DASC brings three important advantages compared with BT and BT-SN. The first one is that the computation of the value of DASC that includes all the sharers for a particular memory block does not require complex hardware: along its way to the home tile, every request message calculates the DASC value. To do so, every time a link is traversed a saturating counter of the same number of bits than the DASC implementation increases (for example, at the same time the routing logic determines the output port for the message). This information is stored into the request message. Then, when the request reaches the home tile, the computed value will be the new DASC value if it is greater than the one already stored in the directory. On the contrary, BT and BT-SN requires more complex extra hardware at each directory controller to compute the tree level every time a new request for each block is received. This, obviously, increases energy consumption (besides area requirements) and can impact also directory occupancy (which could increase cache miss latencies, and therefore, degrade performance).

The second advantage is that DASC can work for any network topology and it does not requires hard-coded information of the topology, *i.e.*, it is topology agnostic. Differently, BT and BT-SN are more suitable for tree topologies, and for other topologies they may require previous information to reduce the number of false positives.

The third advantage is that multicast coherence messages could be sent in a more efficient way with DASC. In particular, on every coherence event, only one coherence message have to be created in the home tile with DASC, which keeps up propagating through all the ports of every router until the remaining links to be traversed becomes zero. This requires the addition of a field to every multicast coherence message initialized by the directory with DASC value. If the greater binary value is inserted, every router copies the messages along all its output ports (*i.e.*, a broadcast is required and the message must reach all the tiles). A value 0 in this field means that the router has to deliver the message just to the local L1 cache controller. For the rest of values, every router would decrement the value of the field and would spawn and propagate the multicast coherence message through the rest of its output ports (including the one that connects with the local L1 cache controller). Contrarily, in BT

and BT-SN several unicast messages must be created by the directory on every coherence event. Extra logic is needed at each directory controller for computing the destinations of every coherence message, which is sent as several unicast messages through the network. Again, this can result into increased directory occupancy and energy consumption, and also, more network traffic in the interconnect.

## 5 Simulation Environment

We have evaluated our proposals using the Simics full-system multiprocessor simulator [15] extended with both GEMS [16] and SiCoSys [17]. GEMS provides a detailed cache coherent memory system timing model. SiCoSys simulates a detailed interconnection network that allows one 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 modified the GEMS simulator in order to evaluate the three mapping policies evaluated in this work. The first mapping policy, named as *RoundRobin* or *RR*, is an OS-managed policy that assigns physical pages in a round-robin fashion to guarantee the uniform distribution of pages among cache banks. This policy gets similar results as the hardware round-robin policy implemented in GEMS, as shown in [11]. The round-robin policy do not take into consideration the distance from the cores to the home bank. The second mapping policy that we have implemented, named as *FirstTouch* or *FT*, maps memory pages to the local cache bank of the first processor that requested the page. Although this mapping policy is distance-aware, it is not concerned about the pressure on some cache banks. Finally, we also implement the policy proposed in [11], named as *DARR*. We simulate *DARR* with a threshold value of 128, as suggested in [11].

On the other hand, we have also implemented in GEMS the different sharing codes evaluated in this work: BT, BT-SN, and DASC (evaluated both for a saturating counter of 2 and 3 bits). We also run the traditional full-map (FM) sharing code, which keeps one bit per core in the system.

The simulated system is a 32-core tiled CMP connected by a  $8 \times 4$  2-D mesh. Each tile contains an in-order processing core since a large number of simple cores can offer better performance/Watt ratio than a small number of complex cores. Moreover, a memory controller connected to a 3D-stacked memory bank is placed in each tile. Table 2 shows the values for the main parameters of the system evaluated in this work. Memory blocks stored in the private L1 caches are kept coherent by means of a directory-based cache coherence protocol that uses MESI states. We account for the variability in multithreaded workloads [18] by doing multiple simulation runs for each benchmark in each configuration and injecting random perturbations in memory systems timing for each run.

Table 2: System parameters

<b>Memory Parameters: GEMS (3GHz)</b>	
Cache block size	64 bytes
Split L1 I & D caches	32KB, 4-way
L1 cache hit time	2 cycles
Shared unified L2 cache	256KB/tile, 8-way
L2 cache hit time	10 cycles
Memory access time	300 cycles
Page size	4KB
<b>Network Parameters: SICOSYS (1.5GHz)</b>	
Topology	8×4 2-dimensional mesh
Switching technique	Wormhole
Routing technique	Deterministic X-Y
Data and control message size	4 flits and 1 flit
Routing time	1 cycle
Switch time	1 cycle
Link latency (one hop)	2 cycles
Link bandwidth	1 flit/cycle

### 5.1 Benchmarking and characterization

We have evaluated our proposal with both parallel and multi-programmed workloads. Multi-programmed workloads consist of several program instances running at the same time in the system. We classify workloads as either *homogeneous*, *heterogeneous*, or *in-between*. Homogeneous workloads uniformly distribute memory pages among cache banks when a first-touch policy is employed. In contrast, in heterogeneous workloads a few banks allocate more pages than the others when the first-touch policy is considered. In-between workloads are neither extremely heterogeneous nor extremely homogeneous.

Our application set includes ten parallel scientific benchmarks that are representative of both homogeneous and heterogeneous scenarios. *Barnes*, *Cholesky*, *FFT*, *FMM*, *LU*, *Ocean*, *Tomcatv* and *Water-NSQ*, represent the homogeneous workloads. *Unstructured*, *Raytrace*, *Radix*, and *Volrend* constitute the heterogeneous workloads. *Barnes*, *FFT*, *Ocean*, *Radix*, *Raytrace*, *Volrend*, and *Water-NSQ* belong to the SPLASH-2 benchmark suite [19] whereas *Tomcatv* and *Unstructured* are irregular scientific applications. The input size of each application is shown in Table 3.

We also consider multi-programmed workloads. We run the configurations shown in Figure 6, two homogeneous and two heterogeneous workloads. *Radix4* consists of four instances of the *Radix* application, with eight threads each one. *Ocean8* consists of eight instances of the *Ocean* application, with four threads each one. They represent the homogeneous workloads. *Mix4* and *Mix8* run *Ocean*, *Raytrace*, *Water-NSQ* and *Unstructured*. In *Mix4* each application has eight threads. In *Mix8* two instances of each application are run with four threads each. These two workloads represent the heterogeneous and more common multi-programmed scenario. A summary of the multi-programmed applications is also shown in Table 3.

Table 3: Benchmarks, input sizes, and characterization

SPLASH 2 (10 benchmarks)				
Benchmarks	Input size	Average	Deviation	Page footprint
Barnes	16K particles, 4 time steps	30.9688	13.6157	In-between
Cholesky	tk15.O	187.344	272.578	Heterogeneous
FFT	64K complex doubles	30.8125	5.39713	Homogeneous
FMM	16K particles	59.25	119.349	Heterogeneous
LU	512×512 matrix	18.9062	3.83952	Homogeneous
Ocean	258×258 ocean	128.938	18.2651	Homogeneous
Radix	1M integers, 1024 radix	80.25	51.2388	In-between
Raytrace	Teapot	27.2188	23.581	In-between
Volrend	Head	6.78125	10.1409	Heterogeneous
Water-NSQ	512 molecules, 4 time steps	21.5938	11.7885	In-between
Scientific (2 benchmarks)				
Benchmarks	Input size	Average	Deviation	Homogeneous
Tomcatv	256 points, 5 time steps	29.0625	6.60401	Homogeneous
Unstructured	Mesh.2K, 5 time steps	29.0938	48.3545	Heterogeneous
Multi-programmed (4 combinations)				
Benchmarks	Input size	Average	Deviation	Homogeneous
Radix4	Radix×4	530.344	32.3639	Homogeneous
Ocean8	Ocean×8	934.781	29.7549	Homogeneous
Mix4	Ocean, Ray, Water, Unstr.	166.281	198.582	Heterogeneous
Mix8	(Ocean, Ray, Water, Unstr.)×2	313.406	382.611	Heterogeneous

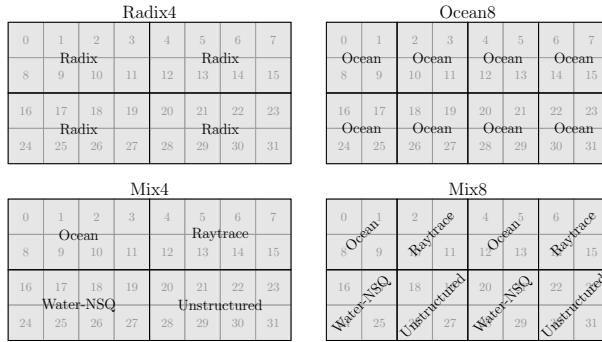


Fig. 6: Multi-programmed workloads evaluated in this work

In order to perform the characterization of the applications evaluated in this work, we first identify the number of pages mapped to each bank for a first-touch policy. Figure 7 shows this number for each of the 32 banks of the simulated NUCA cache. The darker the color of the box is, the more pages are mapped to that bank. We can see that there are applications like *FFT*, *LU*, *Ocean*, *Tomcatv*, *Radix4*, and *Ocean8* that are clearly homogeneous. We can also observe that *Cholesky*, *FMM*, *Unstructured*, *Volrend*, *Mix4*, and *Mix8* are heterogeneous.

In order to give a numeric value to the degree of heterogeneity we employ the coefficient of variation of the pages mapped to NUCA banks in a first-touch policy. The coefficient of variation is calculated as the ratio of the standard deviation to the mean. Table 3 shows these values for every of the applications considered in this work.

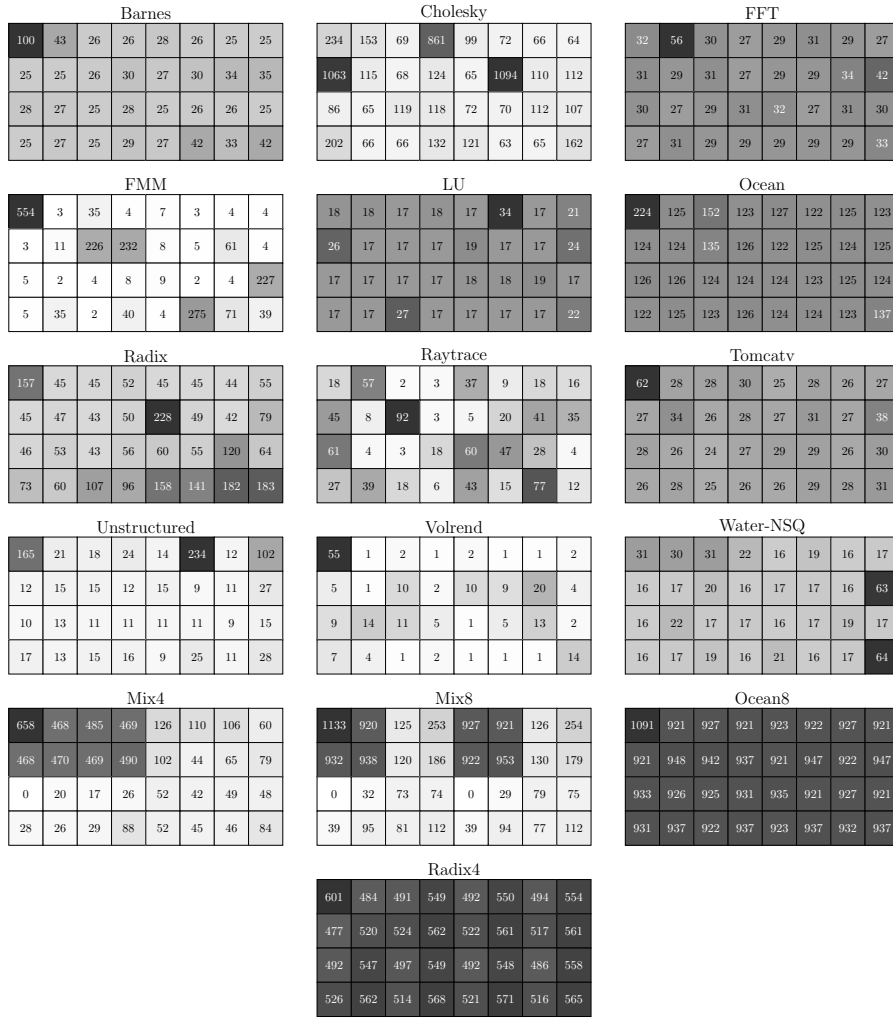


Fig. 7: Number of pages mapped to each cache bank in a first-touch policy for the workloads evaluated in this work

We consider that an application is homogeneous if its coefficient of variation is smaller than 0.4. On the other hand, if the coefficient of an application is greater than 1.0, we consider it as heterogeneous. Applications with a coefficient in between these values are in-between, *i.e.*, neither homogeneous nor heterogeneous. Figure 8 plots graphically this characterization. The y-axis represents the coefficient of variation while the x-axis indicates the average number of pages mapped to each tile. An application like *Volrend*, that is heterogeneous but has a small working set (footprint size) can work well when



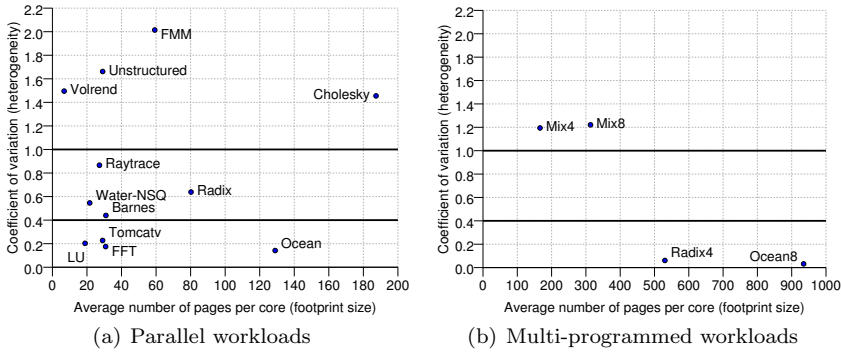


Fig. 8: Characterization of the applications evaluated

the first touch policy is employed, but applications like *Mix8* will incur in a lot of evictions from the NUCA cache, and the consequent off-chip accesses.

In the next section, evaluation results are shown splitting considered applications in these three categories: Homogeneous, Heterogeneous, and In-between.

## 6 Evaluation Results

The main focus of this work is to study the advantages of implementing very compressed sharing codes, like BT, BT-SN, and especially DASC, in systems that employ a distance-aware NUCA mapping policy, like first-touch and DARR. For a detailed evaluation of the DARR mapping policy we refer the interested reader to [11].

In order to understand the effects that the DARR mapping policy can have on the election of the sharing code, we first study the average distance of the sharers to the corresponding home cache bank. Then, we show how shorter distances reduce the number of coherence messages (invalidations and cache-to-cache transfer commands) when compressed sharing codes are used. This leads to reductions in execution time and power consumption when they are used in conjunction with distance-aware mapping techniques, as will be shown later. Finally, we analyze the scalability of the DASC sharing code proposed in this work.

### 6.1 Average distance of sharers to the home banks

The sharing code proposed in this work is very sensitive to the distance from the sharers of a memory block to the corresponding home bank. The shorter the distance is, the better the sharing code works. Fortunately, both the first-touch and the DARR mapping policies reduce the average distance from the

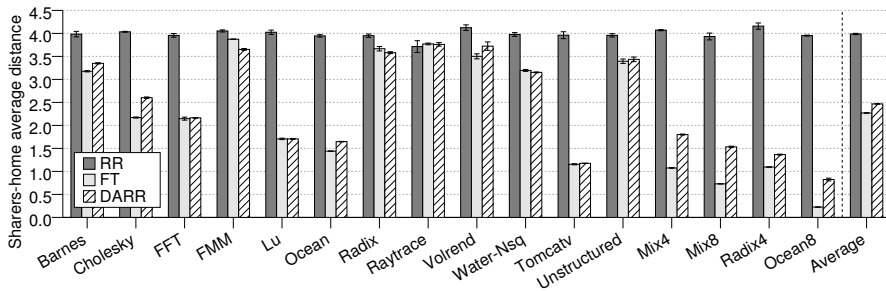


Fig. 9: Average distance of sharers to their home

sharers to the home banks, and this is why DASC reveals as a good option for systems implementing distance-aware mapping policies.

Figure 9 plots the average distance in terms of network hops between the sharers of a memory block and its home bank. Average distances shown in this graph are computed assuming a  $8 \times 4$ -mesh 32-core system and for the three mapping policies considered in this work: round-robin (RR), first-touch (FT), and DARR. We can see that the average distance from sharers to the home banks in RR is around four network links. This is because this policy does not care about the distance from the requesting cores (*i.e.*, the cores that can share the block) to the home node. When the mapping process is performed taking into account distance to the home nodes, average distance can be considerably reduced, especially for scenarios when different applications are running in the same system (*e.g.*, *Mix4*, *Mix8*, *Radix4*, and *Ocean8*). Although the average distance obtained with FT is lower than the observed for DARR (2.3 vs. 2.5), FT causes unbalanced distribution of the load of the cache banks (number of pages assigned to each bank) in heterogeneous applications, which results in extra L2 misses. DARR distributes better the pages among the cache banks in order to reduce off-chip accesses caused by L2 misses, thus increasing the distance from requesters to home banks. However, as it can be seen, this distance is not increased considerably, and consequently, DASC sharing code can work efficiently.

## 6.2 Number of messages per coherence event

By obtaining a short distance between sharers and home banks, compressed sharing codes based on the distance can reduce the number of false positives and work in a more efficient way. Figure 10 plots the number of coherence messages sent on average for any coherence event such as L1 cache misses or directory evictions. Particularly, this figure shows 15 bars, one per each of the configurations considered in this work. They are grouped in sets of 3 bars. From left to right, the first 3 bars are for the full-map sharing code (FM) when it is used in combination with the round-robin (RR), first-touch (FT)

and distance-aware round-robin (DARR) policies, respectively. The next bars are for the BT-SN and BT sharing codes (3+3). The final 6 bars are for the two implementations of DASC: the one using three bits (bars 10, 11 and 12) and the other one using 2 (last 3 bars). This is also applicable to Figures 11 and 12 that will be discussed later on.

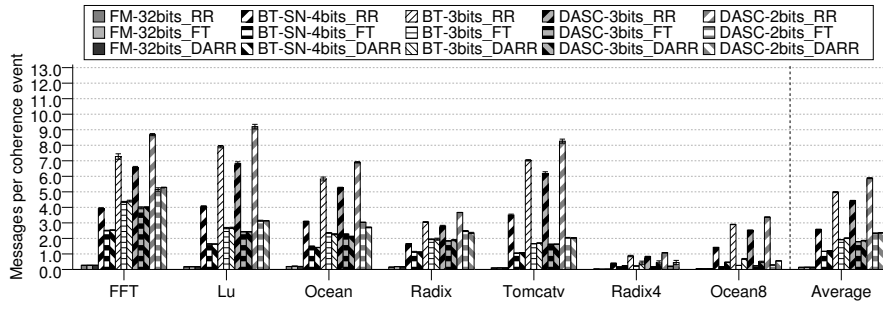
The numbers shown in Figure 10 are for the simulated 32-core system, so in a broadcast-based protocol, *i.e.*, a protocol without a directory, this number would be 32 for write misses and read misses that do not find the data in L2. Differently, we have very compressed sharing codes that require a few bits per entry (from 4 to 2) and require much less coherence messages. We also show the number of messages required by a precise full-map (FM) sharing code. We split the graphs in the three categories mentioned in the characterization: Homogeneous, Heterogeneous, and In-between.

Figure 10(a), shows the results obtained for the homogeneous workloads. In this case, distance-aware mapping policies can reduce distance of sharers to a greater extent, and therefore, we can observe a significant reduction in terms of number of coherence messages when FT or DARR are employed. The lower is the number of bits used for the sharing code, the more false positives and coherence messages are required. However, this is more acute for the RR mapping. Note also that the 3-bit DASC entails less coherence messages than the 3-bit BT sharing code, so it will perform better. This is because two nodes can be very close in the system but far away in the binary tree. For example, in the 16-core example in Figure 5 node 4 and node 8 are separated by only one link, but in the tree structure in Figure 3 their common level is the root of the tree. On the other hand, although the 2-bit DASC sharing code noticeably increases the coherence messages over the 3-bit DASC for RR, when FT or DARR are employed this increase is more acceptable. In particular, the number of messages per coherence event required by DASC-2bits for both FT and DARR is around 2.3, which is a very small number taking into account the size of the sharing code (just 2 bits).

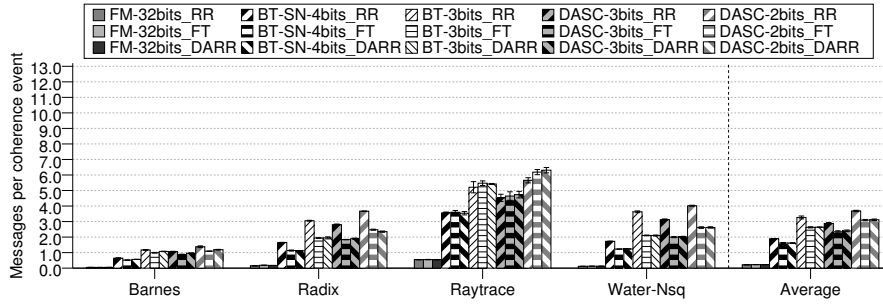
In Figures 10(b) and 10(c), we show workloads where wider distribution of the pages among cache banks is required for DARR. This is why FT requires less coherence messages than DARR (especially for the heterogeneous applications). However, it is important to remember that this comes at the cost of extra off-chip accesses (as shown in [11]), which will increase final execution times as will be discussed in the next section. Overall, the reduction in the number of coherence messages in DASC when moving from RR to DARR is considerable, which confirms the synergy between DASC and DARR (what we call DASC-DIR in this work).

### 6.3 Execution time

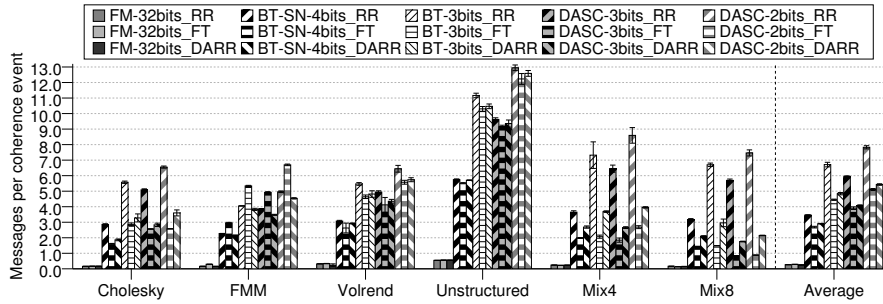
The reductions in execution time that the DARR mapping policy brings along with the low overhead of the DASC sharing code result in improvements in ex-



(a) Homogeneous



(b) In-between



(c) Heterogeneous

Fig. 10: Coherence messages per coherence event

execution time with respect to a non-scalable full-map directory with the typical RR mapping policy.

Figure 11 shows the execution time for the configurations evaluated in this work. We can observe that for the Homogeneous workloads (Figure 11(a)), a 2-bit DASC sharing code performs very well when used in combination with DARR mapping (what we call DASC-DIR), and the increase in execution

time with respect to a non-scalable full-map is only 5.7%. The improvements of this 2-bit sharing code with respect to a traditional full-map employing the typical round-robin policy are 8.4% on average. The 2-bit DASC sharing code performs similarly to the other compressed sharing codes.

Regarding the In-between workloads (Figure 11(b)), a similar pattern can be observed. DASC-DIR performs similarly to our base configuration when the 2-bit DASC sharing code is employed. For the 3-bit version of DASC improvements of 3.7% are obtained. Additionally, it can be also observed that the 3-bit version of DASC performs better than the 3-bit BT sharing code, even when we do not take advantage of the opportunities that the simpler implementation provided by DASC brings (and that were explained in Section 3)<sup>1</sup>. Finally, first-touch and DARR obtain similar results for these workloads.

Finally, for the Heterogeneous workloads (Figure 11(c)), first-touch shows performance degradation with respect to DARR. DARR obtains the lowest execution times out of the three mapping policies. Additionally, the DASC sharing code is efficient enough to not increase execution times, being able to achieve reductions in execution time of 4.9% and 3.1% with respect to the non-scalable base configuration for its 3-bit and 2-bit versions, respectively.

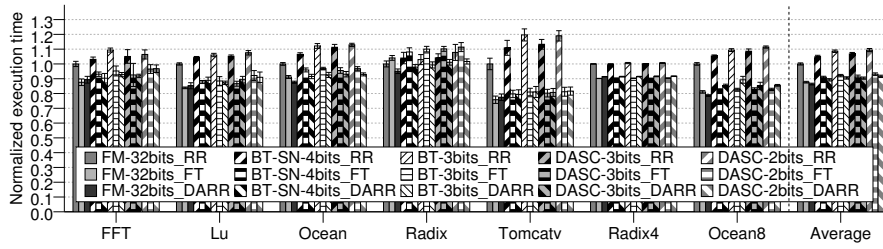
#### 6.4 Network traffic

An increase in the number of coherence messages issued due to false positives when a compressed sharing code is employed, can considerably affect network traffic, and consequently, increase energy consumption. Figure 12, shows the traffic split in the number of control and data flits issued by each switch in the network. In general, we can see that the increase in network traffic due to false positives is lower when distance-aware mapping policies are employed. Also, the traffic generated by the 3-bit version of DASC is lower than the traffic generated by the 3-bit BT sharing code.

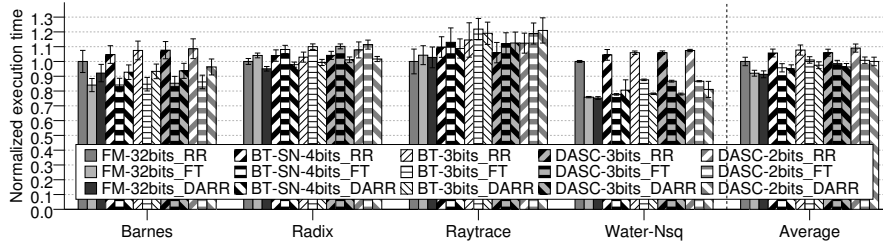
For the Homogeneous workloads (Figure 12(a)), the traffic required by a *DASC-2bits\_DARR* is 17.4% lower than the base configuration (*FM-32bits\_RR*). This is due to two reasons. First, the low distance obtained by the DARR policy that reduces the number of network hops per message issued. Second, the efficiency of the DASC sharing code that do not generate excessive extra coherence messages.

For the In-between workloads (Figure 12(b)), the traffic required by a *DASC-2bits\_DARR* increases by 39.3% with respect to the base configuration, since the distance from sharers to home banks increases for these workloads. A 3-bit DASC sharing code reduces this degradation up to 23.4%. For the Heterogeneous workloads (Figure 12(c)), the traffic is increased by 36.6% and 63.4%, on average, for the 3-bit and the 2-bit versions of DASC, respectively.

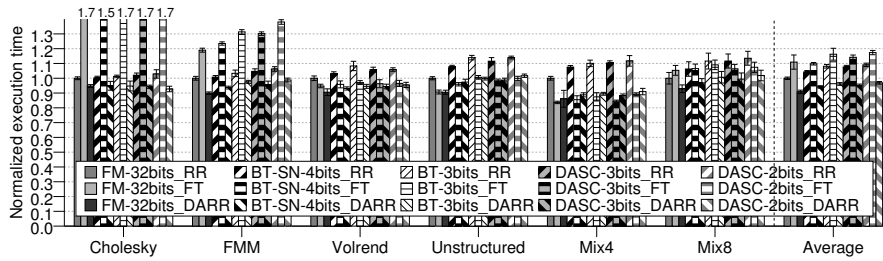
<sup>1</sup> To have a clearer understanding of the impact that the used compressed sharing code has on the results, we concentrate solely on the number of unnecessary coherence messages, leaving implementation-dependant details out of the comparison



(a) Homogeneous



(b) In-between



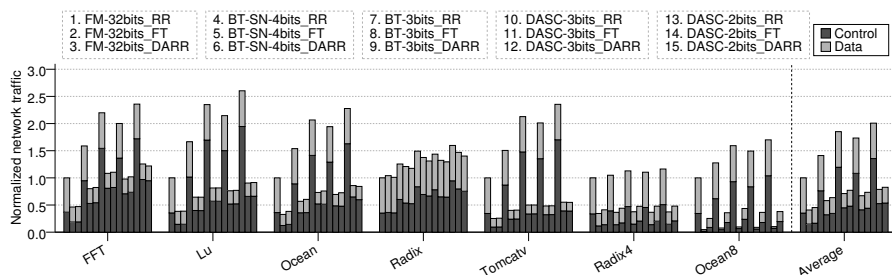
(c) Heterogeneous

Fig. 11: Normalized execution time

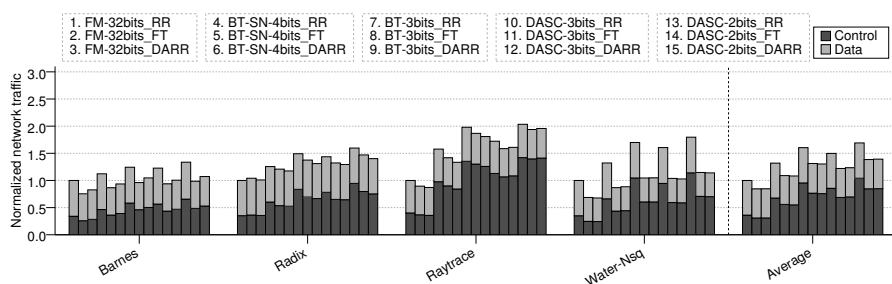
However, for the multi-programmed workloads (*Mix4* and *Mix8*) the traffic keeps low, since the distance among sharers and the home banks keeps low.

### 6.5 Memory requirements and scalability

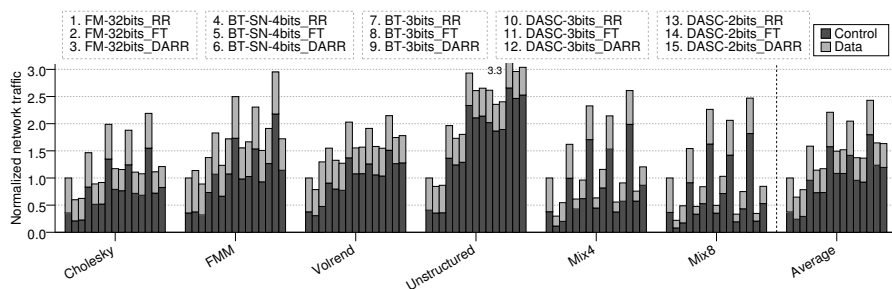
The proposed directory scheme (DASC-DIR) is highly scalable because the size of the sharing code (DASC) it uses does not depend directly on the number of cores in the system. Obviously in larger systems it can be advisable to increase the size of the sharing code to reduce broadcasts, but if system uses a distance-aware mapping policy a few bits are enough to obtain good performance. Figure 13 shows the memory overhead of the sharing code with respect to the



(a) Homogeneous



(b) In-between



(c) Heterogeneous

Fig. 12: Normalized network traffic

memory required by the caches (L1 and L2). This overhead is shown for the different sharing codes: a full-map sharing code, a coarse vector that employs clusters of 4 cores [20], a limited pointer scheme that stores two pointers and when more than two cores share the block employs broadcast [21, 22], a binary tree sharing code both with one symmetric node and without symmetric nodes,

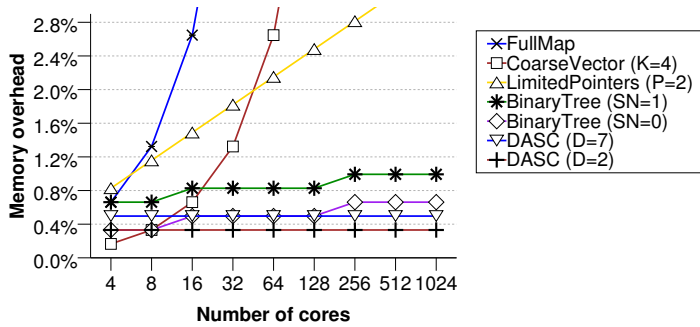


Fig. 13: Memory overhead of several sharing codes schemes

and our DASC sharing code both employing 3 bits (maximum distance 7) and 2 bits (maximum distance 2).

We can observe that only binary tree and DASC are highly scalable, being able to not incur in more than 1% overhead for systems with 1024 cores. Since the size of DASC does not depend on the number of cores, this sharing code is the one offering lowest overheads for large-scale systems. Additionally, as discussed in the previous sections, the combination of DASC with the DARR mapping policy obtains better results than when the binary tree-based counterparts are used.

## 7 Related Work

### 7.1 Proposals to reduce latency in NUCA architectures

There are several ways of reducing cache access latency in NUCA caches. The most relevant ways are data migration, data replication or to perform an intelligent data mapping to cache banks. Next, we comment on the most important works for these approaches.

Kim *et al.*[12] presented non-uniform cache architecture (NUCA) caches. They studied both a static mapping of blocks to caches and a dynamic mapping based on *spread sets*. In such dynamic mapping, a block can only be allocated in a particular *bank set*, but this bank set can be comprised of several cache banks that act as *ways* of the bank set. In this way, a memory block can migrate from a bank far from the processor to another bank closer if the block is expected to be accessed frequently. Chishti *et al.* [23] achieved more flexibility than the original dynamic NUCA approach by decoupling tag and data arrays, and by adding some pointers from tags to data, and vice versa. The tag array is centralized and accessed before the data array, which is logically organized as distance-groups. Again, memory blocks can reside in different banks within the same bank set. Differently from the last two proposals,



Beckmann and Wood [24], considered block migration in multiprocessor systems. They proposed a new distribution of the components in the die, where the processing cores are placed around the perimeter of a NUCA L2 cache. Migration is also performed among cache banks belonging to the same bank set. The block search is performed in two phases, both requiring broadcasting the requests. Unfortunately, these proposals have two main drawbacks. First, there are data placement restrictions because data can only be allocated in a particular bank set and, second, data access requires checking multiple cache banks, which increases network traffic and power consumption.

Zhang and Asanovic [4] proposed victim replication, a technique that allows some blocks evicted from an L1 cache to be stored in the local L2 bank. In this way, the next cache miss for this block will find it at the local tile, thus reducing miss latency. Therefore, all L1 cache misses must look for the block at the local L2 bank before the request is sent to the home bank. This scheme also has two main drawbacks. First, replication reduces the total L2 cache capacity. Second, forwarding and invalidation requests must also check the L2 tags in addition to the L1 tags. Later on, in [25], they proposed victim migration as an optimization that removes some blocks from the L2 home bank when they are frequently requested by a remote core. Now, the drawback is that an extra structure is required to keep the tags of migrated blocks. Moreover, in both proposals, write misses are not accelerated because they have to access the home tile since coherence information does not migrate along with the data blocks.

Differently from all the previous approaches, and closer to ours, Cho and Jin [8] proposed using a page-size granularity (instead of block-size). In this way, the OS can manage the mapping policy, *e.g.*, a first-touch mapping policy can be implemented. In order to deal with the unbalanced utilization of the cache banks, they propose using bloom filters that collect cache access statistics. If a cache bank is *pressured*, the neighbouring banks can be used to allocate new pages. As discussed in Section 2, this proposal has several implementation issues (*e.g.*, it is difficult to find an accurate metric to decide whether a cache is pressured or not) and they do not evaluate the cache pressure mechanism with neither parallel nor multi-programmed workloads. In addition, they only distribute pages among neighbouring banks, *i.e.*, at one-hop distance. In contrast, in our proposal pages are distributed among all banks, if necessary, in an easy way and without requiring any extra hardware. On the other hand, they do not care about the issue of the private cache indexing since they use 16KB 4-way L1 caches, in which the number of bits used to index them is smaller than the number of bits of the offset of the 8KB pages considered in that work, and they can use virtually indexed L1 caches. Lin *et al.* [26] applied Cho and Jin's proposal to a real system. They studied the dynamic migration of pages and the high overheads that it causes. Recently, Awasthi *et al.* [27] and Chaudhuri [28] proposed several mechanisms for page migration that reduce the overhead of migration at the cost of requiring extra hardware structures. Unfortunately, since migration of pages entails an inherent cost (*e.g.*, flushing caches or TLBs), this mechanism cannot be performed

frequently. Although migration can be used along with our proposal, this work focuses on the initial mapping of pages to cache banks. Finally, Awasthi *et al.* [27] do not consider the private cache indexing issue because they use small caches that can be virtually indexed, and Chaudhuri [28] do not take care about the indexing bits despite one bit matches with the home offset bits.

Another direction is to employ a private-shared classification of the accessed data to reduce the NUCA access latency, as described in the *Reactive NUCA* proposal [9]. Private blocks are placed into the local NUCA bank of the requesting core, enabling low-latency accesses for such blocks, while shared blocks are placed across all tiles at the corresponding address-interleaved locations. Further optimizations in the address-interleaved locations for shared blocks were lately studied by García *et al.* [29]. Although in Reactive NUCA the classification is done at page-level by the operating system, compile-time classifications have also been proposed to this end [30,31]. In [30], a data ownership analysis of memory regions is performed at compilation time. This information is transferred to the page table by modifying the behavior of the memory allocator by means of hooks. This proposal is further improved in [31] by considering a new class of data, named as practically-private, which is mapped to the NUCA cache according to a first-touch policy. Differently, our approach do not rely on page classification, it includes pressure metrics to avoid extra off-chip misses, and it achieves low latency even for accesses that would be classified by the other approaches as shared and interleaved across NUCA banks.

## 7.2 Proposals to reduce the size of the directory

Several proposals aimed at reducing the size of the coherence directory have been proposed recently [32–36]. Differently from these proposals, DASC-DIR is based on the use of a very compressed sharing code, DASC. As already explained, DASC can be easily implemented, and do not require complex hardware at the directory controllers. Other compressed sharing codes, as BT, BT-SN or BT-SuT [14], have more cost as they need extra hardware to compress/decompress directory information. Apart from the compressed sharing codes evaluated in this work, others were proposed in the past with a variety of sizes. Some of the most used compressed sharing codes are *coarse vector* [20], which was employed in the SGI Origin 2000 multiprocessor, limited pointers [21,22], employed in FLASH [37] and Alewife [38], *tristate* [39] and *gray-tristate* [40].

## 8 Conclusions

In this work we propose an efficient and low-overhead coherence directory (DASC-DIR) which is built around two main ingredients. The first is the use of the DARR (*Distance-Aware Round-Robin*) mapping policy, an OS-managed

policy which tries to map the pages accessed by a core to its closest (local) bank, at the same time it introduces an upper bound on the deviation of the distribution of memory pages among cache banks, which lessens the number of off-chip accesses. The second is the utilization of a very compressed directory structure which takes advantage of this mapping policy to represent sharers in a very compact way without increasing coherence network traffic. Particularly, the new compressed sharing code introduced in this work, called DASC (*Distance-Aware Sharing Code*), stores the distance between the home node and the farther sharer. Thanks to the use of DARR, this distance keeps usually low, which allows DASC to outperform other compressed sharing codes previously proposed which are unaware of the mapping policy employed in the system. Additionally, contrary to previous proposals, DASC does not require the introduction of extra hardware at the directory controllers for compressing/decompressing sharing information. This way, this work illustrates for the first time the important synergy between the sharing code used for the coherence directory and the mapping policy of the system, two design aspects that should be analyzed together for good system efficiency and scalability.

Results show that the 2-bit version of DASC along with the DARR mapping policy can outperform in terms of execution time and memory requirements (*i.e.*, area and energy required for the directory) a traditional non-scalable full-map directory with the typically used round-robin mapping policy. All of this is achieved without increasing coherence network traffic. Finally, DASC-DIR scales gracefully to larger core counts.

**Acknowledgements** This work has been supported by the Spanish MINECO, as well as European Commission FEDER funds, under grant “TIN2012-38341-C04-03”, and also by the “Fundación Séneca-Agencia de Ciencia y Tecnología de la Región de Murcia” under grant “18956/JLI/13”.

## References

1. J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, B. Sinharoy, POWER4 system microarchitecture, *IBM Journal of Research and Development* 46 (1) (2002) 5–25.
2. Intel Xeon Phi Coprocessor, <http://software.intel.com/en-us/mic-developer> (2013).  
URL <http://software.intel.com/en-us/mic-developer>
3. G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, A. Agarwal, Atac: A 1000-core cache-coherent processor with on-chip optical network, in: 19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), 2010, pp. 477–488.
4. M. Zhang, K. Asanović, Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors, in: 32nd Int'l Symp. on Computer Architecture (ISCA), 2005, pp. 336–345.
5. R. Kalla, B. Sinharoy, W. J. Starke, M. Floyd, POWER7: IBMs next-generation server processor, *IEEE Micro* 30 (2) (2010) 7–15.
6. M. Shah, J. Barreh, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, C. Olson, B. Saha, D. Sheahan, L. Spracklen, A. Wynn, UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC, in: *IEEE Asian Solid-State Circuits Conference*, 2007, pp. 22–25.

7. G. H. Loh, 3d-stacked memory architectures for multi-core processors, in: 35th Int'l Symp. on Computer Architecture (ISCA), 2008, pp. 453–464.
8. S. Cho, L. Jin, Managing distributed, shared L2 caches through OS-level page allocation, in: 39th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 2006, pp. 455–465.
9. N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki, Reactive NUCA: Near-optimal block placement and replication in distributed caches, in: 36th Int'l Symp. on Computer Architecture (ISCA), 2009, pp. 184–195.
10. C. J. Hughes, C. Kim, Y.-K. Chen, Performance and energy implications of many-core caches for throughput computing, *IEEE Micro* 30 (6) (2010) 25–35.
11. A. Ros, M. Cintra, M. E. Acacio, J. M. García, Distance-aware round-robin mapping for large NUCA caches, in: 16th Int'l Conf. on High Performance Computing (HiPC), 2009, pp. 79–88.
12. C. Kim, D. Burger, S. W. Keckler, An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches, in: 10th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS), 2002, pp. 211–222.
13. S. Das, A. Fan, K.-N. Chen, C. S. Tan, N. Checka, R. Reif, Technology, performance, and computer-aided design of three-dimensional integrated circuits, in: Int'l Symp. on Physical Design, 2004, pp. 108–115.
14. M. E. Acacio, J. González, J. M. García, J. Duato, A two-level directory architecture for highly scalable cc-NUMA multiprocessors, *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 16 (1) (2005) 67–79.
15. P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, Simics: A full system simulation platform, *IEEE Computer* 35 (2) (2002) 50–58.
16. M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, D. A. Wood, Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset, *Computer Architecture News* 33 (4) (2005) 92–99.
17. V. Puente, J. A. Gregorio, R. Beivide, SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems, in: 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002, pp. 15–22.
18. A. R. Alameldeen, D. A. Wood, Variability in architectural simulations of multi-threaded workloads, in: 9th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2003, pp. 7–18.
19. S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, A. Gupta, The SPLASH-2 programs: Characterization and methodological considerations, in: 22nd Int'l Symp. on Computer Architecture (ISCA), 1995, pp. 24–36.
20. A. Gupta, W.-D. Weber, T. C. Mowry, Reducing memory traffic requirements for scalable directory-based cache coherence schemes, in: Int'l Conf. on Parallel Processing (ICPP), 1990, pp. 312–321.
21. D. Chaiken, J. Kubiawicz, A. Agarwal, LimitLESS directories: A scalable cache coherence scheme, in: 4th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS), 1991, pp. 224–234.
22. R. Simoni, M. A. Horowitz, Dynamic pointer allocation for scalable cache coherence directories, in: Int'l Symp. on Shared Memory Multiprocessing, 2001, pp. 72–81.
23. Z. Chishti, M. D. Powell, T. N. Vijaykumar, Distance associativity for high-performance energy-efficient non-uniform cache architectures, in: 36th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 2003, pp. 55–66.
24. B. M. Beckmann, D. A. Wood, Managing wire delay in large chip-multiprocessor caches, in: 37th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 2004, pp. 319–330.
25. M. Zhang, K. Asanović, Victim migration: Dynamically adapting between private and shared CMP caches, Tech. rep., Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory (Oct. 2005).
26. J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, P. Sadayappan, Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems, in: 14th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2008, pp. 367–378.
27. M. Awasthi, K. Sudan, R. Balasubramonian, J. Carter, Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within

- large caches, in: 15th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2009, pp. 250–261.
28. M. Chaudhuri, PageNUCA: Selected policies for page-grain locality management in large shared chip-multiprocessor caches, in: 15th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2009, pp. 227–238.
  29. A. García-Guirado, R. Fernández-Pascual, A. Ros, J. M. García, Dapsco: Distance-aware partially shared cache organization, *ACM Transactions on Architecture and Code Optimization (TACO)* 8 (4) (2012) 25:1–25:19.
  30. Y. Li, A. Abousamra, R. Melhem, A. K. Jones, Compiler-assisted data distribution for chip multiprocessors, in: 19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), 2010, pp. 501–512.
  31. Y. Li, R. G. Melhem, A. K. Jones, Practically private: Enabling high performance cmprs through compiler-assisted data classification, in: 21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), 2012, pp. 231–240.
  32. A. Ros, M. E. Acacio, J. M. García, Scalable directory organization for tiled CMP architectures, in: Int'l Conference on Computer Design (CDES), 2008, pp. 112–118.
  33. J. Zebchuk, V. Srinivasan, M. K. Qureshi, A. Moshovos, A tagless coherence directory, in: 42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), 2009, pp. 423–434.
  34. B. Cuesta, A. Ros, M. E. Gómez, A. Robles, J. Duato, Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks, in: 38th Int'l Symp. on Computer Architecture (ISCA), 2011, pp. 93–103.
  35. M. Ferdman, P. Lotfi-Kamran, K. Balet, B. Falsafi, Cuckoo directory: A scalable directory for many-core systems, in: 17th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2011, pp. 169–180.
  36. D. Sanchez, C. Kozyrakis, SCD: A scalable coherence directory with flexible sharer set encoding, in: 18th Int'l Symp. on High-Performance Computer Architecture (HPCA), 2012, pp. 129–140.
  37. J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. A. Horowitz, A. Gupta, M. Rosenblum, J. L. Hennessy, The Stanford FLASH multiprocessor, in: 21st Int'l Symp. on Computer Architecture (ISCA), 1994, pp. 302–313.
  38. A. Agarwal, R. Bianchini, D. Chaiken, D. Kranz, J. Kubiatowicz, B. hong Lim, K. Mackenzie, D. Yeung, The MIT Alewife machine: Architecture and performance, in: 22nd Int'l Symp. on Computer Architecture (ISCA), 1995, pp. 2–13.
  39. A. Agarwal, R. Simoni, J. L. Hennessy, M. A. Horowitz, An evaluation of directory schemes for cache coherence, in: 15th Int'l Symp. on Computer Architecture (ISCA), 1988, pp. 280–289.
  40. S. S. Mukherjee, M. D. Hill, An evaluation of directory protocols for medium-scale shared-memory multiprocessors, in: 8th Int'l Conf. on Supercomputing (ICS), 1994, pp. 64–74.