PS Directory: A Scalable Multilevel Directory Cache for CMPs

Joan J. Valls, Alberto Ros, Julio Sahuquillo and María E. Gómez

***Abstract*—As the number of cores increases in current and future chip–multiprocessor (CMP) generations, coherence protocols must rely on novel hardware structures in order to scale in terms of performance, power, and area. Systems that use directory information for coherence purposes are currently the most scalable alternative.**

This paper studies the important differences between the directory behavior of private and shared blocks, which claim for a separate management of both types of blocks at the directory. We propose the PS directory, a two-level directory cache that keeps the reduced number of frequently accessed shared entries in a small and fast first-level cache, namely *Shared* cache, and uses a larger and slower second-level *Private* cache to track the large amount of private blocks. Entries in the Private cache do not implement the sharer vector, which allows important silicon area savings.

Speed and area reasons suggest the use of eDRAM technology, much denser but slower than SRAM technology, for the Private cache, which in turn brings energy savings. Experimental results for a 16-core CMP show that, compared to a conventional directory, the PS directory improves performance by 14% while reducing silicon area and energy consumption by 34% and 27%, respectively. Also, compared to the state-of-the-art Multi-Grain Directory, the PS directory apart from increasing performance, it reduces power by 18.7%, and provides more scalability in terms of area.

I. Introduction

As the number of cores increases in both current and future shared-memory chip–multiprocessor (CMP) generations, coherence protocols must scale to sustain performance. Directory–based coherence is the commonly preferred approach over snoop–based coherence, because the former keeps track of cached blocks to avoid the use of broadcast messages. Two main design choices have been used in both research proposals [9], [18] and commercial processors [4], [8], [26] to implement CMP directories: *Duplicate Tags* and *Sparse Directories*.

Duplicate-tag based directories keep track of all cached blocks without invalidating any of them due to directory space constraints, thus without hurting the cache performance. Nevertheless, this approach may become prohibitive even for a relatively small number of cores because of the high energy consumed by the highly associative lookups, which are required to build the *sharer vector* on each directory access. On the other hand, sparse directories use a cache–like structure, referred to as directory cache, to keep track of the cached blocks. In this approach, when a directory entry is evicted, all copies of the tracked block in the processor caches are invalidated, even if the block is being used by the processor so rising the so–called coverage misses. Mainly due to power reasons (i.e. low associativity degree), sparse directories are the preferred design choice for a medium to high number of cores.

Each entry of a conventional directory cache mainly stores the owner of the block and the sharer vector, whose size grows linearly with the number of cores. As a consequence, it is expected that directories in future CMP generations will have important on–chip area and leakage overheads [34]. Therefore, large many-core CMPs demand for directory designs that scale in terms of area and power. Several attempts have addressed these issues by focusing on reducing (shortening or compressing) the sharer vector length [1], [6], [7], [22].

The directory cache proposed in this work follows a different approach and is based on the different behaviors exhibited by cache blocks in parallel workloads. We found that shared and private blocks present important differences from the directory cache point of view, which translate to different directory cache requirements (e.g., number of ways, cache size, or access time). Most of the accessed blocks are private [9], [35] and each of them uses an entry in the directory cache. But these blocks do not require coherence actions, that is, the sharer vector field is not used at all, which implies the entry size could be reduced for most of the directory entries. Moreover, since private entries will not be accessed again after the block is fetched, their access time does not affect system performance. On the other hand, most of the directory accesses concentrate on a reduced number of entries that track shared blocks, which require low associativity as shown in Section II-A. So, most of the accesses to the directory could be solved by looking up a reduced number of ways and therefore bringing power savings.

This paper proposes the PS (Private Shared) directory, which relies on empirical findings on the block behavior from the directory cache perspective. The PS directory consists of two independent caches, referred to as Private cache and Shared cache, each one tuned to the behavior exhibited by each block type. The Shared cache is designed with much less entries and associativity than a typical directory cache, attending to the low fraction of expected shared blocks. Reducing associativity yields to significant energy savings for an important amount of directory lookups. Due to its low access latency, it can resolve

J.J. Valls, J. Sahuquillo and M.E. Gómez are with the Department of Computer Engineering, Universitat Politècnica de València, Spain
E-mail: joavalmo@fiv.upv.es, jsahuqui@disca.upv.es, megomez@disca.upv.es

A. Ros is with the Computer Engineering Department, Universidad de Murcia, Spain
E-mail: aros@ditec.um.es

indirections early. Moreover, most indirections are solved by this small cache. In contrast, the Private cache implements a larger number of entries and associativity, but its entries do not include the sharer vector field, thus enabling scalability. Directory entries from the Private cache can be moved from the Private cache to the Shared cache at run–time in case the tracked block becomes shared.

The PS directory can be implemented in typical SRAM (6T cells) technology. Nevertheless, in order to address power and area scalability for a high number of cores we also study the benefits of using eDRAM technology [20], which has been already used to implement large caches in recent commercial processors like the IBM Power7 [13]. SRAM technology is used for speed to implement the small fast Shared cache with low associativity while eDRAM is used for area and power in the much larger Private cache, in which access time is not a concern.

The proposal has been compared against a conventional directory with the same number of entries and the state-of-the-art Multi-Grain Directory. Experimental results for a 16-core CMP show that, compared to the conventional directory, the PS directory improves performance by 14% due to the separate treatment of private and shared blocks, while reducing area by 26.35%. In addition, when eDRAM technology is considered, this reduction is as high as 33.98%. In terms of energy, the PS directory allows energy savings by 27% with SRAM technology. On the other and, when compared to the Multi-Grain Directory, the PS directory allows power savings by 18.7% and speedups the performance by 16.7% while requiring less silicon area. Concerning scalability, the PS directory is able to reduce up to 84.3% the area required by the conventional directory cache for a 1024–core system with the same number of entries.

This paper presents two major contributions with respect to existing directory proposals:

- We propose a sparse directory scheme which provides scalability in terms of area and energy, which is the major shortcoming of sparse directories.
- The proposed directory presents minimal performance degradation with respect to a perfect directory.

In short, the proposal achieves similar performance as the duplicate tags approach, but with a feasible implementation for future many core CMPs that provides major energy gains. A short and preliminary version of this approach can be found in [30]. This paper refines the proposal and evaluates it in greater detail.

The rest of the paper is organized as follows. Section II analyzes the behavior of shared and private blocks and discusses the main technology reasons that led us to do this work. Section III describes the assumed baseline processor. Section IV presents the proposed approach. Section V describes the simulation environment. Section VI analyzes the performance, area and energy consumption. Section VII discusses the related work. Finally, Section VIII presents some concluding remarks.

II. Motivation

This section focuses on the two main pillars that support this work. First, the behavior of shared and private blocks is analyzed, showing key observations that guided us to the final design. Second, the key technology issues that enable the proposed design to scale are discussed.

A. Analyzing the Behavior of Private and Shared Blocks from the Directory Point of View

The PS directory relies on the fact that private and shared blocks present different behavior from the directory point of view, which can be outlined in four key observations and one finding. As explained below, these five key points advocate to organize directory caches in two independent structures, one for tracking private blocks and the other for shared blocks.

- **Observation 1:** *Directory entries keeping track of private blocks do not require the sharer vector field.*
- **Observation 2:** *Most data blocks in parallel workloads are private.*

According to these two observations, the Private cache should be designed narrower and taller than the Shared cache, that is, with shorter entries but with higher number of them. Due to the smaller entry size in the Private cache important area savings can be achieved, especially for systems with a large number of cores, thus offering scalability. Notice that the larger the Private cache is (in comparison with the Shared cache), the more area savings can be obtained, thanks to the missing sharer vector field.

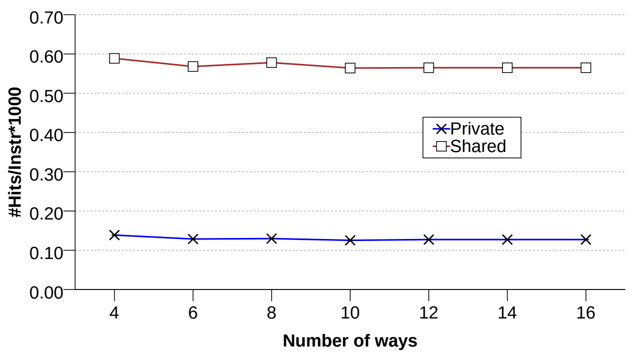
- **Observation 3:** *Most directory hits concentrate on shared entries.*
- **Observation 4:** *Almost all directory entries for private blocks are accessed only once.*

These observations emphasize that private blocks access the directory either when they are not stored in the processor cache (e.g., the first access to a block or invalidations due to directory evictions) or when a write-back is performed (e.g., due to space constraints in the processor cache). The first case will cause a directory miss, while the second case will hit in the private directory cache and will invalidate the corresponding entry. On the other hand, shared entries are accessed more times due to several cores accessing the same block. Thus, most directory hits are due to shared blocks. According to this reasoning, the PS directory scheme accesses the Shared cache first so preventing likely useless accesses to the Private cache (which has higer associativity), which will result in energy savings.

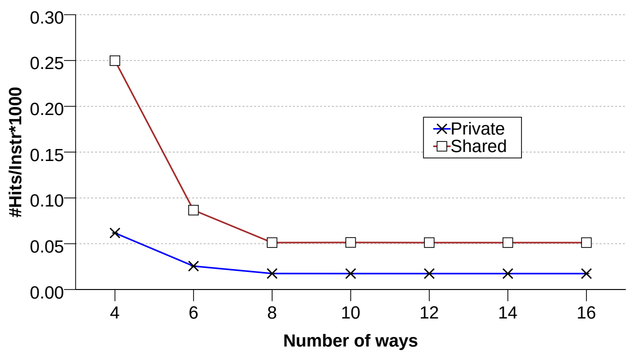
Figure 1 depicts the number of directory entries hits (differentiating between shared and private) per kilo instructions committed, varying the number of ways in the directory cache and keeping constant the number of sets[1]. Two benchmarks, *Barnes* from the SPLASH-2 benchmark suite [31] and *Blackscholes* from PARSEC [5], have been used to illustrate these observations.

As can be seen in Figure 1(a), the number of hits in entries tracking shared blocks is about $5\times$ larger than that in entries

[1]Experimental conditions are defined in Section VI.

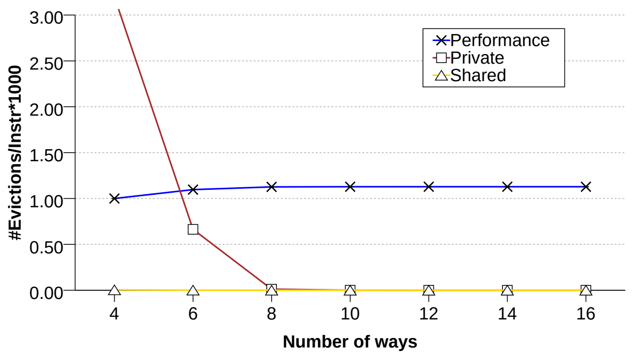


(a) Barnes

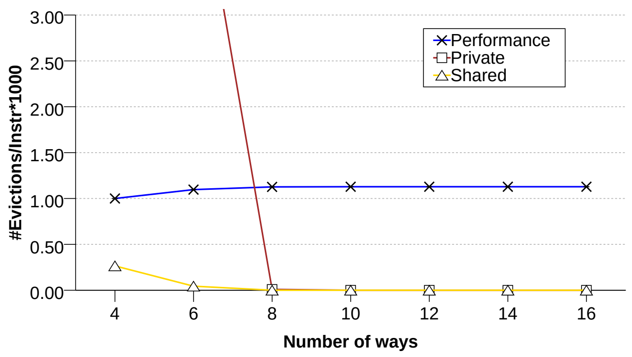


(b) Blackscholes

Figure 1. Number of hits to private and shared entries per kilo instruction in a conventional directory.



(a) Barnes



(b) Blackscholes

Figure 2. Number of evictions of private and shared entries per kilo instruction in a conventional directory and its effect on performance.

tracking private blocks in Barnes. Entries of private blocks are only looked up again in case a block is replaced either from the directory or from the processor cache, and then asked again by the processor. In both cases, the directory entry is removed, thus when the corresponding private block is looked up in the directory, a miss will occur. Private entries are scarcely accessed in spite of being the number of them much larger than that of shared entries. Results for Blackscholes (1(b)) show minor differences for higher number of ways because the number of directory evictions is noticeably reduced in this benchmark as the directory capacity increases. With a lower number of evictions, the number of L1 coverage misses will also decrease. Hence the directory will be accessed less frequently. These results suggest that while shared blocks should have a reduced directory access time for performance, this time is not so critical for private blocks. Keeping this observation in mind, we study the potential benefits of using a power and area aware technology to implement the private cache.

- **Finding 1:** *Shared directory entries have much less associativity requirements than private directory entries.*

To quantify the proper associativity degree, we ran experiments with a conventional (or single–cache) directory varying the number of ways. We identified and quantified the number of evicted directory entries that cause subsequent misses in the processor caches, and classified them into private and shared according to the type of the block that was being tracked. Then, the effect of both block types on performance was measured. Misses in the processor caches that occur due to a directory entry eviction will be referred to as *coverage misses* as also done is some recent works [9], [24].

We found that private and shared entries have different associativity requirements. Figure 2 illustrates the results for two different workloads. Results reveal that the number of evicted shared blocks provoking coverage misses slightly varies with the number of ways, while the number of private blocks drops dramatically. The number of evicted private blocks is really high for a low associativity degree, which translates to significant performance degradation.

Assuming a typical LRU replacement policy and taking into account that entries in the directory tracking private blocks are not accessed again, entries tracking a private block work out like a FIFO policy, which means that, in absence of locality, the impact of private blocks on performance mainly depends on the number of ways available to them. The larger the number of ways is, the longer the time for each entry in the directory cache.. If it is too low, it is likely that the block will be forced to leave the processor cache, even though it is still being used, thus increasing the number of coverage misses. On the other hand, with higher number of ways, we give them more chances before eviction. It can be observed that around 8 ways is enough to stabilize the number of evictions of private blocks as well as the system performance.

Based on these results, the Private cache will be designed with around 8 ways (see Section V for further details), whereas the Shared cache will implement a lower number of ways (i.e., 2 ways), since the impact of shared blocks on performance mainly depends on their access locality and are not benefited from such a large complexity.

B. Dealing with Scalability in Future CMPs

CMP systems must be designed to accommodate specific area and power budgets. Both technological constraints represent major design concerns since they prevent future manycore CMPs from scalability with future increasing core counts. Power consumption is mainly distributed between cores and large on-chip cache memories in current designs. Caches occupy a large percentage of the on-chip area to mitigate the huge penalties of accessing the off-chip main memory. Giving more silicon area and power to the cache hierarchy and related structures (e.g. directory caches) leaves less space and power for cores, which could force CMP designs with simpler cores so yielding to lower performance, especially harmful for single-threaded applications [19].

Many efforts have been carried out in both the industry and academy to deal with power and area focusing on the cache subsystem, including processor caches, off-chip caches and directory structures. Regarding the latter structures, directory caches have been proven to provide effectiveness and scalability, both in terms of power and area, for a small to medium number of cores. However, these design issues must be properly faced by future systems since the pressure on achieving good cache performance increases with the core count. There are two main ways to tackle these issues: architectural solutions to achieve a good tradeoff among performance, area and power, and mingling disparate technologies in a power and/or area aware design. Both ways can be applied independently or together, as proposed in this work.

This paper presents architectural innovations to track separately private and shared blocks in two independent directory caches. The main aim of this two-cache approach is to tailor each cache structure to the requirements of each block type with architectural solutions. In addition, alternative technologies can be used for implementing the different caches. For instance, a power aware technology can be used for one cache while a fast technology can be employed for the other one.

The cache hierarchy has been typically implemented with SRAM technology (6 transistors per cell) which incurs in important power and area consumptions. A few years ago, technology advances have allowed to embed DRAM (eDRAM) cells in CMOS technology [20]. An eDRAM cell integrates a trench DRAM storage into a logic circuit technology. Table I highlights the main properties of these technologies regarding the design issues addressed in this work. Compared to SRAM, eDRAM cells have both less power consumption and higher density but lower speed. Because of the reduced speed, eDRAM cells have not been used in manufactured first-level high-performance processor caches. In short, both technologies present diverse features regarding density, speed, and power.

Table I
COMPARING TECHNOLOGICAL FEATURES OF SRAM VERSUS EDRAM.

Technology	**Density**	**Speed**	**Power**
SRAM	low	fast	high
eDRAM	high	slow	low

These CMOS compatible technologies have been used both in the industry and the academia to implement processor caches. For instance, in some modern microprocessors [13], [27], [28] SRAM technology is employed in L1 processor caches while eDRAM cells are used to allow huge storage capacity in last level caches. Regarding academia, some recent works [29], [32] mingle these technologies in several cache levels. In short, both technologies properly combined at different (or even the same) cache structures can be used to address speed, area, and power in the cache subsystem.

In this paper, and to the best of our knowledge, this is the first time that both technologies are combined to implement the directory cache. We use SRAM for speed in the frequently accessed Shared cache while eDRAM is employed for power and area savings in the much larger Private cache. Therefore, scalability and performance are provided by design thanks to the joined use of architectural techniques and the choice of the appropriate technology for each cache structure.

III. BASE ARCHITECTURE

A tiled CMP architecture consists of a number of replicated *tiles* connected by a switched direct network. Different tile organizations are possible so, to focus the research, this work assumes that 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, a directory-based cache coherence protocol is employed with a directory cache storing coherence information. Both the L2 cache and the directory cache are shared among the different processing cores but they are physically distributed among them, that is, it is implemented as a NUCA architecture [14]. Therefore, a fraction of accesses to the L2 cache is sent to the local slice while the rest is serviced by remote L2 slices. In addition, L1 and L2 caches are non-inclusive, that is, some blocks stored in the L1 caches may not have an entry in the L2 cache (but in the directory). Figure 3 shows the organization of a tile (left side) and a 16-tile CMP (right side), which is used as baseline for experimental purposes.

IV. THE PS DIRECTORY SCHEME

The main goal of the proposed approach is to take advantage of the different behavior exhibited by shared and private directory entries to design scalable directory caches while, at the same time, improving their performance. Figure 4 depicts the proposed two-level organization consisting of the Private

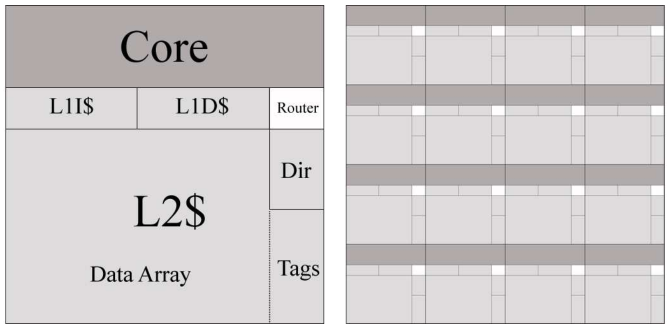


Figure 3. Organization of the tile assumed in this work and a 4×4 tiled CMP.

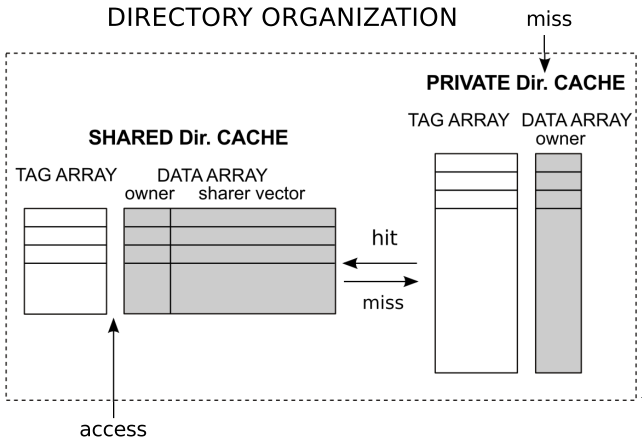


Figure 4. Private-Shared directory organization.

cache and the Shared cache. As previously exposed, the Private cache is designed with narrower entries since they do not require the sharer vector and with a larger number of entries because of the expected high number of private blocks. On the other hand, the Shared cache has a reduced number of entries, thus the sharer vector is only implemented in a small fraction of directory entries.

When an access to a memory block misses in the processor cache, it is looked up on the directory for coherence maintenance. Then, if the access results in a directory miss, the block is provided by the corresponding NUCA slice (or by the main memory) to the processor cache, and an entry is allocated in the directory cache to track that block. In the PS directory this entry is allocated in the Private cache since the block is held at this point of time by a single cache. Then, the core identifier is stored in the owner entry field.

On subsequent accesses to that memory block by the same processor, it will find the block in its L1 cache, so no additional access to the directory cache will be done. On the other hand, when that block is evicted from the processor cache, two main actions are carried out: i) the data block is written back in the NUCA cache, and ii) the directory cache is notified in order to invalidate the entry of that block (stored in the Private cache). Thus, a subsequent access to that block will result in a directory cache miss. This means that the Private cache access time does not affect the performance of private blocks since these blocks are provided directly to cores by the NUCA cache

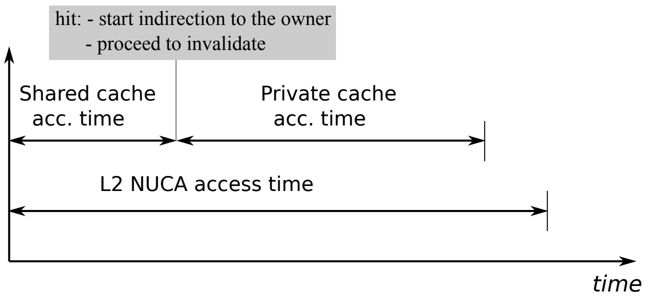


Figure 5. Parallel access of the Shared cache and the NUCA cache. Private cache is only accessed on a miss in the Shared cache.

or main memory.

If a block tracked by the Private cache is accessed by a core other than the owner, the block becomes shared and its entry is moved to the Shared cache updating the sharer vector accordingly. From then until eviction, coherence of this block is tracked in the Shared cache. That is, the proposal allows only unidirectional movements from the Private to the Shared cache. Bidirectional transfers of entries among both caches have been also explored but the extra hardware cost does not justify the scarce benefits.

Regarding timing, directory caches are typically accessed in parallel with the NUCA cache. On a directory hit, the data block can be provided either by the NUCA cache or by a remote processor cache (i.e. the owner). In case that the data block must be provided by a remote processor cache, the NUCA access is canceled. Analogously, the PS directory could access both directory cache structures simultaneously, however since most directory accesses concentrate on shared blocks, the PS scheme only accesses the Shared cache in parallel with the NUCA slice. This way provides major energy savings with minimal performance penalty. Figure 5 depicts this design choice. Depending on the protocol, specific coherence actions can start as soon as a hit rises in the Shared cache; for instance, read requests can be forwarded to the owner of the block, or invalidation requests can be issued to the caches sharing the block in case of write requests. On a miss in the Shared cache, the Private cache is accessed. As mentioned, this access could be also performed in parallel with the Shared cache but at expenses of power while bringing minimal benefits on performance. On a miss in this cache, which is the most frequent case, there will be no energy or performance gains or penalties by accessing both directory structures in parallel instead of sequentially, since both structures have to be accessed and the sum of their access time is still lower than the NUCA access time that is accessed in parallel. The main difference appears on a Private cache hit. By accessing both directory structures in parallel the directory access time would be slightly reduced on a private directory hit, but at the expense of higher and unnecessary energy consumption on a shared directory hit. Since hits on the shared directory are more frequent, making this access sequential was the preferred design choice.

Figure 6 summarizes the actions carried out by the directory controller on a coherence access, which works as follows:

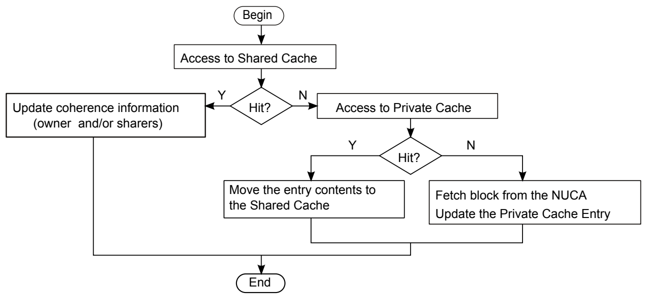

Begin
Access to Shared Cache
Hit?
Y
N
Update coherence information (owner and/or sharers)
Access to Private Cache
Hit?
Y
N
Move the entry contents to the Shared Cache
Fetch block from the NUCA
Update the Private Cache Entry
End

Figure 6. Directory controller flow-diagram.

- When a coherence request reaches the directory, the directory controller looks up first the Shared cache since it is more likely that the access results in a hit in this cache due to the higher fraction of accesses to shared entries. On a hit, the controller updates (if needed) the sharer vector, performs the associated coherence actions, and cancels the NUCA access (depending on the block state). On a miss in the Shared cache, the controller looks up the Private cache. This sequential timing has, on average, negligible impact on performance since most directory accesses are to shared blocks, and most accesses to private blocks provide the block from the NUCA cache.
- A hit in the Private cache means that the block is shared because another core already has a copy of it in its cache. The processor that accessed it the first time will not access the directory again because its cache already holds the block, unless a data cache or directory eviction occurs and then the entry will miss in the directory again. Hence, the directory entry is moved to the Shared cache. This way ensures that entries for private blocks are retained in the Private cache while shared entries are filtered and moved to the Shared cache.
- On a directory miss, the corresponding block entry is allocated in the Private cache to keep track of the missing block. As there is no coherence information stored for that block in the two directory caches, then the block is not being currently cached by any processor. Thus, the block is assumed to be private to the core accessing it and the owner information (requesting processor) is updated with the core identifier.
- In the proposed implementation, when an entry is replaced from any of both directory caches it leaves the directory after performing the corresponding invalidations, and no movement to the other cache is allowed.

The proposal reduces area by design with respect to conventional caches implemented with the same number of entries since directory entries in the Private cache are much narrower. In addition, power is also reduced by accessing smaller cache structures sequentially. Nevertheless, the use of two independent organizations with different design goals, speed for the Shared and capacity for the Private, suggests that using specific technologies addressing these design issues could provide the proposal further energy and area savings.

Table II
SYSTEM PARAMETERS

Memory Parameters	
Cache hierarchy	Non-inclusive
Cache block size	64 bytes
Split L1 I & D caches	64KB, 4-way (256 sets)
L1 cache hit time	2 cycles
Shared single L2 cache	512KB/tile, 8-way (1024 sets)
L2 cache hit time	2 (tag) and 6 (tag+data) cycles
Single directory cache	256 sets, 4 ways (same as L1)
Single directory cache hit time	2 cycles
Memory access time	160 cycles
Network Parameters	
Topology	2-dimensional mesh (4x4)
Routing technique	Deterministic X-Y
Flit size	16 bytes
Data and control message size	5 flits and 1 flit
Routing, switch, and link time	2, 2, and 2 cycles

Low-leakage technologies or transistors with low leakage currents could be used in the Private cache, whose number of entries is much higher and its access time is not critical for performance. This work explores the use of eDRAM technology in the Private cache which provides, as experimental results show, important area and leakage savings.

V. SIMULATION ENVIRONMENT

The proposed cache scheme has been evaluated with full-system simulation using Virtutech Simics [16] along with the Wisconsin GEMS toolset [17], which enables detailed simulation of multiprocessor systems. In order to evaluate systems with higher number of cores (see Figure 9), we have implemented a Pintool [15] that simulates both the base system and the behavior of our PS directory scheme. The interconnection network has been modeled using GARNET [3], a detailed network simulator included in the GEMS toolset. We simulate a 16-tile CMP architecture like the described in Section III. The values of the baseline system parameters used in the evaluation are shown in Table II. We used the CACTI 6.5 tool [21] to estimate access time, area requirements and power consumption of the different cache structures for a 32nm technology node.

Different configurations for the PS directory have been evaluated with a $1\times$ *coverage ratio* if not stated a different ratio. This ratio indicates the number of directory entries per processor cache entry. For instance, in the $1\times$ ratio, each directory cache slice has the same amount of entries as an L1 cache (i.e. $1\times$). Two PS directory configurations have been evaluated varying its shared-to-private ratio (1:3 and 1:7), that is, the number of entries in the Private cache is three and seven times greater, respectively, than that of the Shared cache. These two directory configurations have been chosen for comparison purposes, because they have the same number of entries (computed as the sum of entries in both directory caches) as the conventional directory cache. Additionally, we perform a sensitivity study with lower coverage ratios for our PS directory in order to show the significant reduction in directory area and power that it can achieve without degrading application performance (Section VI-C).

Table III shows the access time and characteristics of the studied directory structures. The first row, labeled as single cache, refers to the conventional single-cache approach (sparse directory) used as baseline. Then, two different PS architectures are presented. Values for the Private cache were calculated both for SRAM and eDRAM technologies and for different coverage ratios. Since, CACTI provides latencies in *ns*, we rounded these values to obtain an integer number of processor cycles. The L2 cache access time was assumed to be 6 cycles, and the remaining access times were scaled accordingly. Notice that eDRAM latency is much longer than SRAM latency.

A key finding (as discussed in Section II-A) is that private blocks are more benefited from higher number of ways than shared blocks. This finding suggests that the Private cache should implement higher associativity degree. Based on this finding, the baseline design proposes a 6-way Private cache (two ways over the conventional baseline cache). However, for a fair comparison, the associativity of the Shared cache is lowered to only 2-ways (i.e. two ways less than the conventional one). Notice that, considering both cache structures, the average number of ways per set matches that of the conventional cache, but skewed to the Private cache in the PS directory.

Apart from comparing the PS directory with a conventional directory cache with as many entries as the sum of the Private and the Shared caches, the PS directory has been also compared against the recently proposed Multi-Grain Directory (MGD) scheme [33]. MGD uses different entry formats of same length and tracks coherence at multiple different granularities in order to provide scalability. Each MGD entry tracks either a temporarily private memory region, or a single cache block with any number of sharers. By using a single entry instead of using one entry per block in the private region, the coherence directory size can be reduced. Region entries rely on a presence vector to indicate which blocks of the region are allocated in the private L1 cache. On a directory miss, a region entry is allocated in the directory. When a second private cache tries to access a block from a private region, the appropriate bit in the region's presence vector is reset and a block entry is allocated in the directory. Block entries work the same way as they do in conventional sparse directories. In the presented results, the associativity of the MGD is 4 ways as in our baseline directory cache, the memory interleaving is 1KB, and the number of entries is $0.5\times$ that of the conventional and PS directories. This coverage ratio has been chosen for the MGD as suggested by their authors with the aim of providing scalability in terms of area and power by grouping blocks in regions.

We evaluate the aforementioned directory schemes with a wide range of scientific applications. *Barnes* (16K particles), *FFT* (64K complex doubles), *Ocean* (514×514 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Radix* (512K keys, 1024 radix), *Raytrace* (teapot –optimized by removing locks for unused ray ids–), *Volrend* (head), and *Water-Nsq* (512 molecules) are from the SPLASH-2 benchmark suite [31]. *Blackscholes* (simmedium) and *Swaptions* (simmedium) belong to PARSEC suite [5]. The experimental results reported in this work correspond to the parallel phase of the evaluated benchmarks.

VI. Experimental Evaluation

A. Performance

This section analyzes the performance of the proposed PS directory for a 16-tile CMP compared to the conventional *sparse* directory and to a multi-grain directory (MGD). The performance of the directory cache must be addressed because it may significantly affect the system performance. Effectively, every time a directory entry is evicted, invalidation messages are sent to the corresponding processor caches for coherence purposes. These invalidations will cause *coverage misses* upon a subsequent memory request to those blocks, therefore impacting on the final performance.

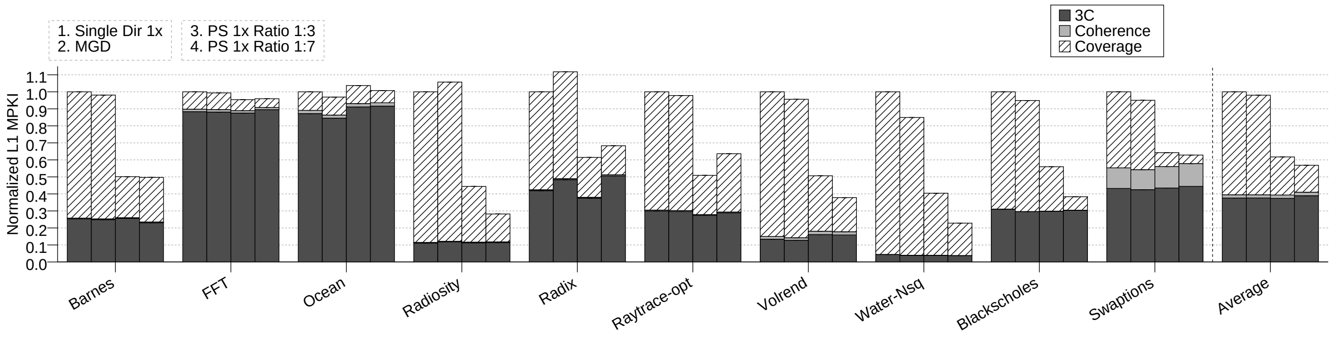
Figure 7(a) shows the L1 MPKI (Misses per KiloInstruction) classified in *3C* (i.e., cold or compulsory, capacity and conflict), *Coherence*, and *Coverage*. As observed, the PS directory cache is able to remove most coverage misses caused by a *single cache* or sparse directory approach with the same number of entries (by 84.2% and 68.2% for 1:7 and 1:3 private-to-shared ratios, respectively). Essentially, this reduction in coverage misses comes from removing conflict misses in the directory cache, which are mainly caused by private directory entries, as shown in Section II-A. Therefore, by adding two additional ways to the Private cache (at the cost of reducing the number of sets, so the number of entries remains the same) most directory conflict misses can be avoided. To illustrate where benefits come from, lets study the 1:3 ratio. This ratio provides the same number of sets as the Shared and as the Private cache, with 2-way and 6-way associativity, respectively. In other words, this PS organization has exactly the same number of sets as the 4-way single cache, and on average, the same number of ways per set. Thus, this scenario clearly shows that critical *private* sets are efficiently handled by the PS scheme. To sum up, performance benefits mainly come from identifying that the private entries suffer from conflict misses and selectively adding associativity to specific structures depending on the requirements of the type of the entries.

The MGD directory reduces the L1 coverage misses by 3.2% with respect to the single conventional directory. Notice that the MGD is able to reduce the number of coverage misses with half the number of entries than the sparse directory. Nevertheless, this reduction is much lower than the one achieved by the PS directory.

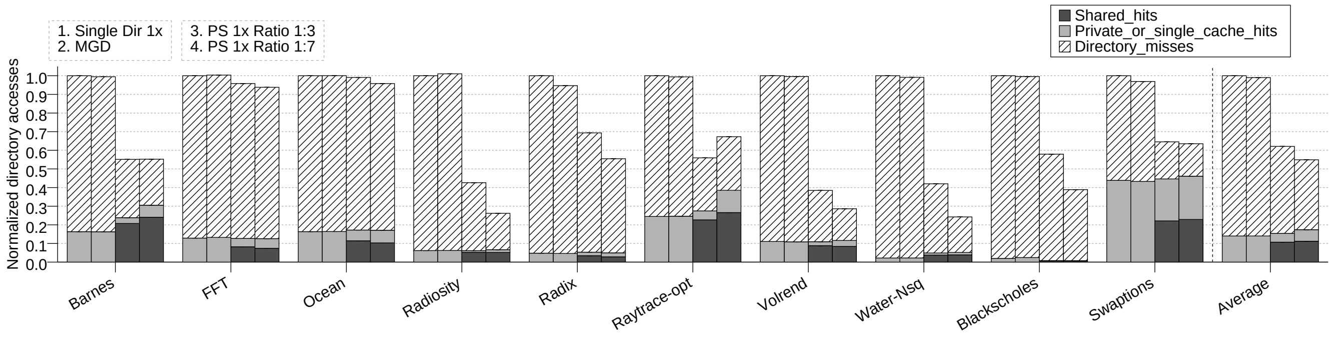
An observation that can confuse the reader is that, in certain applications, by reducing the number of misses, the number of executed instructions rises. This is a side effect that can occur in multi-core systems, like the one studied in this work, when running parallel workloads. The main reason is due to *spin-waiting* instructions. More precisely, in parallel workloads,

Table III
DIRECTORY LATENCIES

Directory cache	# Ways	1× # Sets	**Latency** 1×	0.5×	0.25×	0.125×
Single cache	4	256	2	2	2	-
Shared dir 1:3	2	128	2	2	2	2
Private dir 1:3 SRAM / eDRAM	6	128	2 / 4	2 / 4	2 / 3	2 / 3
Shared dir 1:7	2	64	2	2	2	2
Private dir 1:7 SRAM / eDRAM	7	128	2 / 4	2 / 4	2 / 3	2 / 3



(a) Normalized L1 MKPI.



(b) Normalized directory accesses.

Figure 7. Normalized misses with respect to a conventional single-cache directory.

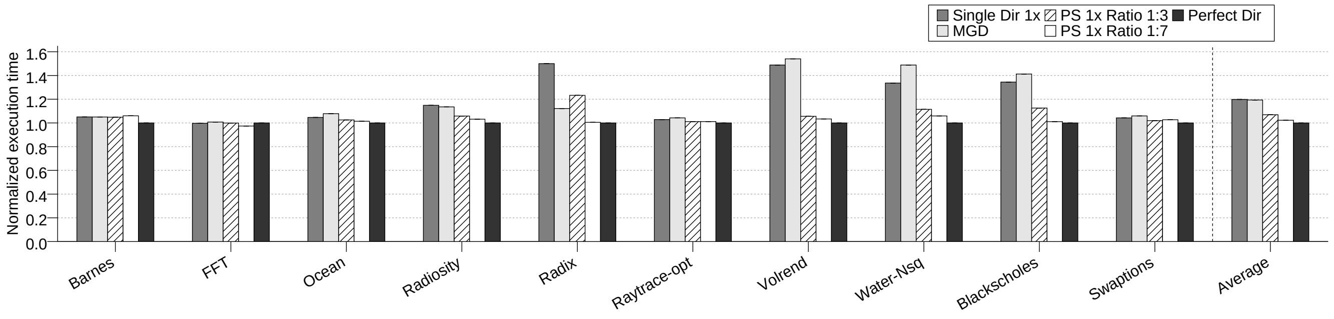


Figure 8. Normalized execution time with respect to a perfect directory.

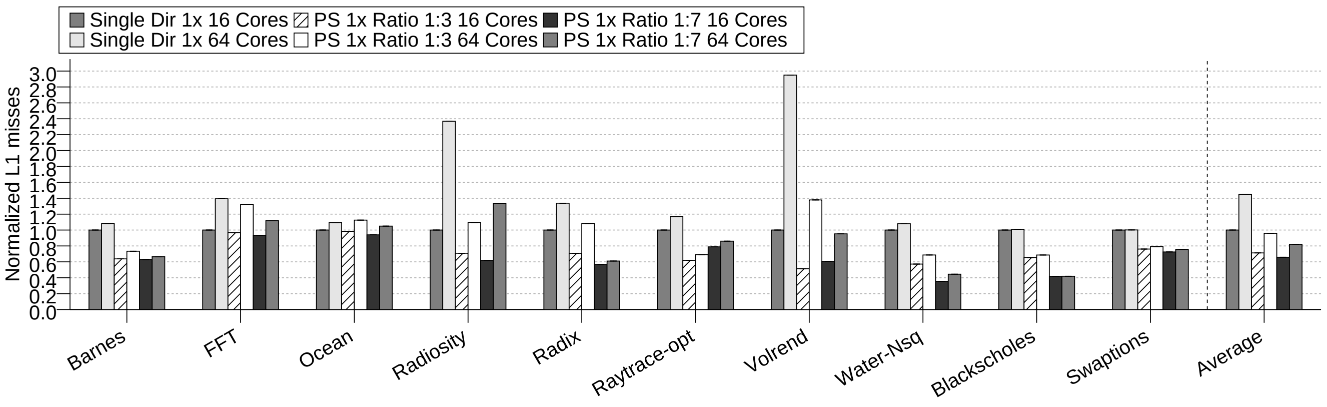


Figure 9. Normalized L1 misses with respect to a single-cache directory with 16 cores.

due to synchronization instructions (e.g., barriers) some cores (the faster ones) have to wait for the slower ones to reach the synchronization point. This causes additional spin-waiting instructions that hit the cache and affect the MPKI values by further reducing them. Moreover execution times do not benefit from it. Some examples of this occurrence is Barnes, where the PS directory increases the instructions executed by 10% and Ocean, where MGD increases instructions executed by 2% and shows a lower MPKI although it slightly increases the number of misses.

Performance of a multilevel directory cache can be defined as the number of coherence requests that find the required coherence information in the directory, that is, as the overall directory hit ratio regardless of the directory structure that provides such information. Figure 7(b) presents the accesses to each PS directory cache classified in misses and hits. In case of a hit, it is also classified in the directory structure that currently has the entry (Private or Shared caches).

Notice that, as expected by design, the Private cache shows on average a poor hit ratio despite the much higher number of entries (3× and 7× times the entries of the Shared cache), and most directory hits concentrate on the Shared cache, which corresponds to the smaller and faster directory structure. Remember that each hit in the Private cache refers to a private block that becomes shared. Although the 1:7 ratio could seem to have a Shared cache too small, it provides on average better results than the 1:3 ratio reducing the number of accesses to the directory. *Ratio 1:3* and *ratio 1:7* reduce the number of accesses to the directory by 37.9% and by 45.1%, respectively, while the MGD directory only reduces this number by 1%.

Reducing both the number of coverage misses in the processor caches and the access latency to the directory cache translate into improvements in execution time as shown in Figure 8. This figure compares the performance of the studied directory schemes with that of a perfect directory cache. A directory cache is referred to be perfect when it does not incur in performance degradation, that is, there are no coverage misses. Therefore, a perfect directory cache provides the same performance as a duplicate tags approach but it offers more scalability. Nevertheless, unlike the proposed scheme, there is no realizable implementation of a duplicate tag approach. Benchmarks with high coverage miss values (i.e. Radix or Blackscholes) are the ones that benefit the most from our proposal or similar ones like MGD. The higher the reduction of coverage misses, the shorter the execution time. Compared to the single directory cache, the PS directory reduces execution time on average by 13.6% and 11.1% for the 1:7 and 1:3 shared-to-private ratios, respectively. Compared to the perfect cache, the Single cache increases the execution time on average by 22.3%, yielding in some case to unacceptable performance (e.g. by 60% in Radix). However, performance drops of this proposal with respect to the perfect cache are by 6.4% and 2.9% for the ratios 1:3 and 1:7, respectively.

The small reduction of coverage misses achieved by MGD also brings, on average, small performance gains (by 3.9%) over the conventional single-cache directory. Compared to the PS directory, the MGD presents a slow-down of 11.6% and 16.7% considering the 1:3 and 1:7 ratios, respectively. This is due to the fact that shared blocks are more frequently accessed at the directory; thus, a shared cache with shorter access time can positively impact on cache miss latency.

To study the scalability of the proposal, we compare the cumulative L1 misses, that is the number the directory accesses (the lower the better), of the considered directory schemes with 16 and 64 cores. Figure 9 shows the results normalized with respect to the L1 misses of the single directory with 16 cores. As observed, when increasing the number of cores from 16 to 64, the single directory increases its L1 misses (i.e., the amount of directory accesses) by 44.8%. An interesting observation is that PS configurations with 64 cores present less L1 misses (by 4.2% and 18% for ratios 1:3 and ratio 1:7) than a single directory with four times less cores. The reason behind this is, as explained before, the important reduction in coverage misses that the proposal achieves. Hence, we can conclude that the PS directory is able to scale in performance better than more conventional schemes, mainly due to the different treatment of block types, which brings significant reductions in coverage misses.

In short, results present the PS directory as a simple and effective design, which is able to reach performance close to a perfect directory with reduced hardware complexity.

B. Area and Energy Analysis

This section shows how the PS directory is able to reduce area and energy consumption while increasing performance.

Table IV shows the area required for different PS schemes and the single directory cache. Both SRAM and eDRAM technologies (as stated in the *Private (Tech.)* column) have been considered for the Private cache design, while the smaller Shared cache is always implemented with fast SRAM technology. As expected, all the PS configurations are able to reduce area, even those entirely implemented with SRAM technology. In particular, compared to the single cache, the PS configurations with SRAM Private caches save by 18.51% and 25.48% of area for 1:3 and 1:7 shared-to-private ratios, respectively. These savings come because the Private cache does not include the sharer vector field. In addition, when eDRAM technology is considered, these reductions grow up to 25.02% and 33.12% for 1:3 and 1:7 shared-to-private ratios, respectively.

Figure 10 depicts the required silicon area per-core for the studied directory configurations. As observed, the single cache directory and the MGD require more area than any of the PS configurations. Additionally, their area requirements grow faster with the number of cores. Notice that in spite of using half the number of entries of a PS directory, the MGD scales poorer than the PS directory. The PS directory is able to reduce by 84.3% (ratio 1:7) and 73.3% (ratio 1:3) the area required by the conventional directory for a 1024–core system, even though all of them have the same number of entries. Thus,

Table IV
AREA (IN $mm^2 * 1000$) OF THE DIFFERENT PS CONFIGURATIONS FOR 16 CORES COMPARED WITH THE SINGLE CACHE DIRECTORY.

Directory	Shared	Private (Tech.)	Total	Area (%)
Single	19.51	–	19.51	100.00%
PS 1:3	6.40	9.50 (SRAM)	15.90	81.49%
	6.40	8.22 (eDRAM)	14.63	74.98%
PS 1:7	3.45	11.08 (SRAM)	14.54	74.52%
	3.45	9.60 (eDRAM)	13.05	66.88%

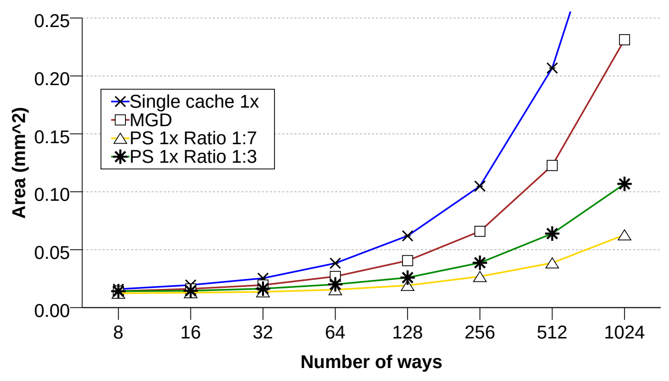


Figure 10. Scalability analysis in terms of area.

the PS directory overcomes one of the biggest problems that sparse directories present, namely, their scalability.

On the other hand, the PS directory attacks energy consumption by design, especially leakage, since it uses two low hardware complexity structures with less storage capacity than a single conventional directory cache. Figure 11 shows the total energy consumed during the benchmarks execution, normalized with respect to the single cache directory. SRAM technology has been assumed in the Private cache of the PS directory. We can observe that a PS directory with the same number of entries as a single cache directory can save around 27% and 20.5% of the energy consumption of the single cache directory for the 1:7 and the 1:3 ratios, respectively, while MGD *only* reduces by 8.9%. This means that a PS directory, with either 1:3 or 1:7 ratio, is able to improve the multi-grain scheme in terms of energy. In short, the PS directory reduces energy consumption by 18.7% with respect to MGD. Moreover, when taking eDRAM technology in the Private directory cache into consideration, the savings are as high as 87.3% and 81.3% for the 1:7 and the 1:3 ratios, respectively, with respect to the single cache directory.

C. Directory Coverage Ratio Analysis

This section evaluates the impact on performance of reducing the directory coverage ratio, that is, the number of entries in the PS directory cache. As the number of entries is reduced in the directory cache, a degradation in performance is expected, but at the same time area and energy consumption will improve. The ideal directory cache size is the one that entails negligible impact on performance while at the same time allows area and energy savings.

Figure 12(a) shows the L1 MPKI (as Figure 7(a)) classified in *3C*, *Coherence*, and *Coverage* for different coverage ratios. As shown, with the only exception of a $0.125\times$ coverage ratio, the proposal still incurs in less L1 cache misses than a single conventional directory cache, on average, allowing a significant reduction in directory cache area. For a $0.125\times$ coverage ratio the increase in the number of cache misses is roughly 20%, on average. This increase in coverage misses translates into a degradation in execution time with respect to a $1\times$ coverage ratio PS directory. However, with respect to a single directory, the execution time is still shortened, even for a $0.125\times$ coverage ratio, as shown in Figure 12(b). Therefore, if reducing silicon area is a target design goal, which would be the main reason for a lower coverage ratio, one can opt for reducing the area overhead of the directory without losing performance with respect to a conventional directory. The PS directory is able to improve the performance of a conventional single directory cache while using 8 times less entries.

Table V shows the area required for different PS schemes with different coverage ratios[2] and the single directory cache. As expected, all the PS configurations are able to reduce area, even those with the same number of entries $(1\times)$ as the conventional directory cache. This is due to the fact that the Private cache does not implement the sharer vector field.

[2]Results for $0.125\times$ are not shown because CACTI is not able to provide results for so small caches.

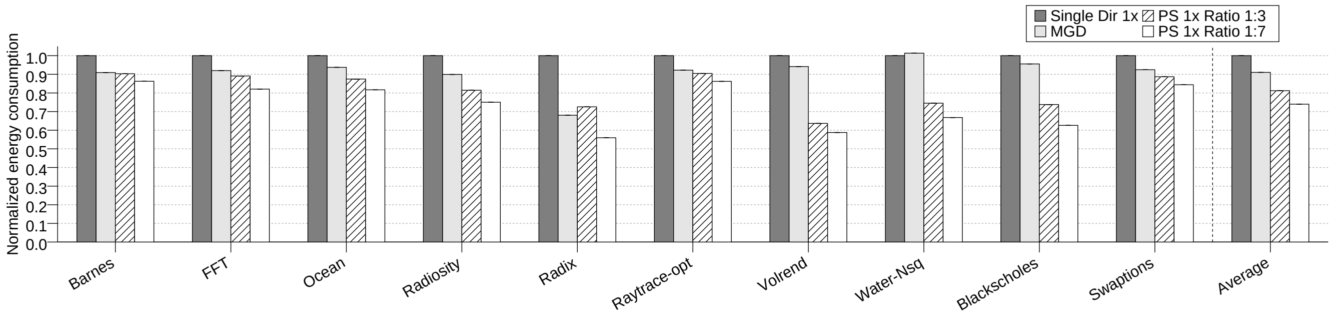
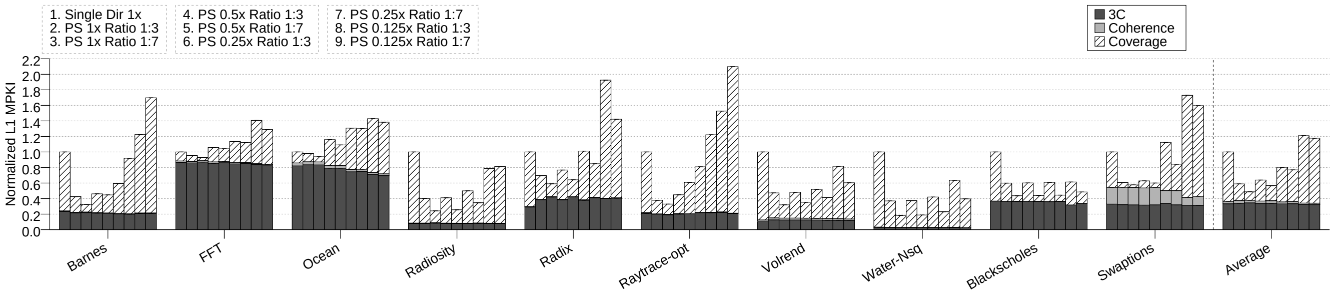
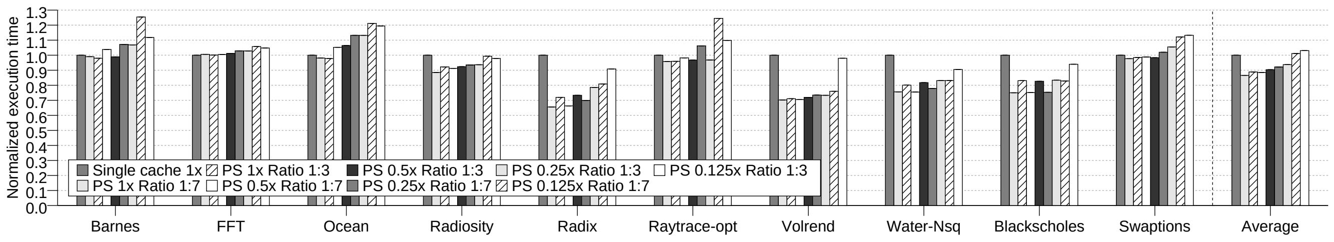


Figure 11. Normalized energy consumed by the directory with respect to a single-cache directory. eDRAM technology is used in the Private directory cache.



(a) Normalized L1 MPKI.



(b) Normalized execution time.

Figure 12. Normalized performance with respect to a conventional single-cache directory.

When the directory coverage ratio is reduced ($0.5\times$ and $0.25\times$ coverage ratios), area savings significantly increase up to $80,24\%$ for the $0.25\times$ 1:7 configuration, while still improving the system performance (as shown previously). Comparing the results for both shared-to-private ratios, we can see that configurations with 1:7 ratio are more area efficient since they are able to reduce area from 12% up to 26% (depending on the directory coverage ratio) over configurations with 1:3 ratio, while providing similar performance results.

Table VI shows the energy (dynamic and static) consumed by the PS directory cache with different coverage ratios and compared to the $1\times$ single directory cache. As observed, the $1\times$ and $0.5\times$ PS configurations consume more dynamic energy per access than the conventional cache, but this is highly offset by the much lower leakage consumed by the PS configurations, which is highly reduced even using SRAM technology in the Private cache. Leakage is reduced from 19% for the SRAM $1\times$ 1:3 configuration up to 86% in the eDRAM $0.25\times$ 1:7 configuration. Comparing 1:3 and 1:7 shared-to-private ratios, the 1:7 configurations are able to reduce leakage consumption from 5% up to 15% with respect to the 1:3 configurations. Taking into account these values, Figure 13 shows the energy consumed during the execution of the benchmarks by the PS directory normalized with respect to the energy consumption by the single-cache directory. Lower coverage ratios lead to less energy consumed at the cost of performance degradation.

Figure 14 depicts the area per core scalability for the studied directory configurations. As observed, the conventional directory cache exhibits the worst area behavior with significant area differences with the PS directory configurations. These differences increase with the number of cores. Even it requires for 128 cores more area than all the PS configurations with up to 1024 cores, with the only exception of the PS $1\times$ 1:3

Table V
AREA (IN $mm^2 * 1000$) OF THE DIFFERENT PS CONFIGURATIONS FOR 16 CORES COMPARED WITH THE 1× SINGLE CACHE DIRECTORY.

Coverage	Directory	Shared	Private	(Tech.)	Total	Area (%)
1×	Single	19,51		–	19,51	100,00%
	PS 1:3	6,33	9,50	(SRAM)	15,83	81,15%
	PS 1:7	3,28	11,08	(SRAM)	14,37	73,65%
	PS 1:3	6,33	8,22	(eDRAM)	14,56	74,61%
	PS 1:7	3,28	9,60	(eDRAM)	12,88	66,02%
0.5×	PS 1:3	3,28	4,80	(eDRAM)	8,09	41,47%
	PS 1:7	1,74	4,80	(eDRAM)	6,55	33,60%
0.25×	PS 1:3	1,74	3,01	(eDRAM)	4,76	24,39%
	PS 1:7	0,84	3,01	(eDRAM)	3,85	19,76%

Table VI
STATIC AND DYNAMIC ENERGY CONSUMPTION OF THE DIFFERENT PS CONFIGURATIONS FOR 16 CORES COMPARED WITH THE 1× SINGLE CACHE DIRECTORY.

Configurations		Pleakage (mW)				Eread (pJ)			
Coverage	Directory	Shared	Private	(Tech.)	Total	Shared	Private	(Tech.)	Total
1×	Single	4,2346	–		4,2346	0,0048	–		0,0048
	PS 1:3	1,1877	2,2572	(SRAM)	3,4450	0,0027	0,0028	(SRAM)	0,0055
	PS 1:7	0,6404	2,6334	(SRAM)	3,2739	0,0016	0,0032	(SRAM)	0,0049
	PS 1:3	1,1877	0,5123	(eDRAM)	1,7001	0,0027	0,0067	(eDRAM)	0,0094
	PS 1:7	0,6404	0,5977	(eDRAM)	1,2382	0,0016	0,0078	(eDRAM)	0,0094
0.5×	PS 1:3	0,6404	0,4114	(eDRAM)	1,0518	0,0016	0,0035	(eDRAM)	0,0052
	PS 1:7	0,3650	0,4799	(eDRAM)	0,8450	0,0010	0,0041	(eDRAM)	0,0052
0.25×	PS 1:3	0,3650	0,3276	(eDRAM)	0,6927	0,0010	0,0027	(eDRAM)	0,0037
	PS 1:7	0,2181	0,3822	(eDRAM)	0,6003	0,0007	0,0032	(eDRAM)	0,0039

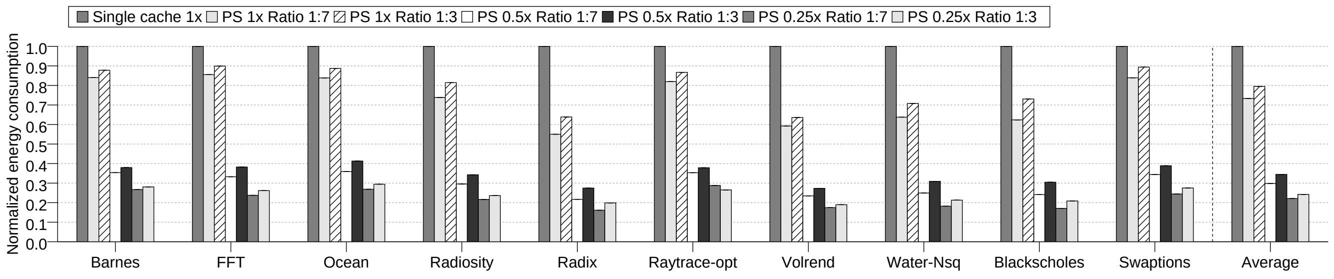


Figure 13. Normalized energy consumed by the directory with respect to a single-cache directory.

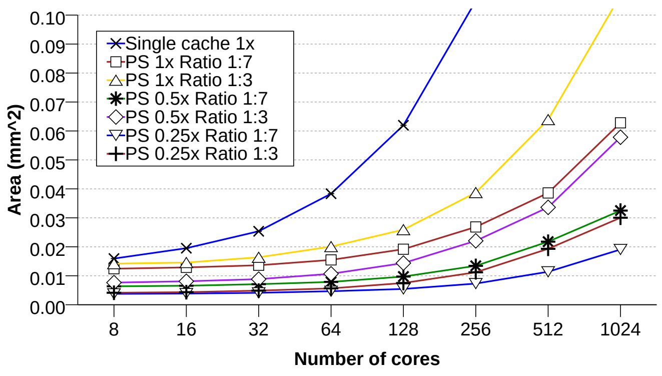


Figure 14. Scalability analysis in terms of area.

configuration.

As stated in previous section for a 1× coverage configuration, the PS directory is able to reduce by $26,71\%$ (ratio 1:7) and $15,71\%$ (ratio 1:3) the area required by the conventional directory cache for a 1024–core system using both the same number of entries. Of course, the area is further reduced with smaller coverage ratios. In particular, for the 0.5× PS configurations, the PS directory requires only by $14,47\%$ (ratio 1:3) and $8,13\%$ (ratio 1:7) the area required by the single cache directory, and for the 0.25× PS configurations only $7,52\%$ (ratio 1:3) and $4,77\%$ (ratio 1:7) the area required by the single cache directory.

VII. RELATED WORK

Cache coherence is needed in shared memory systems where multiple cores are allowed to access the same memory blocks. This work focuses on directory-based protocols, which are the commonly adopted solution for a medium to large core count. These protocols use a coherence directory to track which private (e.g. L1) processor caches share each block. The directory structure is accessed to carry out coherence actions such as sending invalidation requests to serialize write operations, or asking a copy of the block to the owner (e.g. the last processor that wrote it).

Traditional directory schemes does not scale with the core count, which is the current trend in the microprocessor industry. Thus, implementing directories that scale to hundreds of cores in terms of power and area is a major design concern. Directory implementations, both in academia and industry, follow two main approaches: duplicate-tag directories and sparse directories.

Duplicate-tag directories maintain a copy of the tags of all tracked blocks in the lower cache level (e.g. the L1 core cache). Therefore, this approach does not raise directory induced invalidations. The sharer vector is obtained by accessing the highly associative directory structure. This approach has been implemented in modern small CMP systems [4], [26] and is the focus of recent research works [23], [34]. The main drawback of this approach is the required associativity of the directory structure, which must be equal to the product of the number core caches by the associativity of such caches. This means that a directory access requires a 512 associative search for 64 8-way L1 caches. Duplicate-tag directories are area–efficient, however, the highly associative structures yield to a non-scalable quadratic growth of the aggregated energy consumption [10], so this approach becomes prohibitive for a medium to large core count.

The high power consumption incurred by duplicate-tag directories has led some research to focus on providing high associativity with a small number of ways. Cuckoo Directory [10] uses a different hash function to index each directory way, like skew–associative caches. Hits require a single lookup but replacements require from multiple hash functions to provide multiple candidates, so giving the illusion of a cache with higher associativity but at the expense of higher consumption and latency.

Sparse cache directories [12] are organized as a set-associative cache like structure indexed by the block address. Reducing the directory associativity makes this approach more power–efficient than duplicate–tag directories. Each cache directory entry encodes the set of sharers of the associated tracked block. Conventional approaches use a bit vector, that is, a bit per-core cache, to encode the sharers. In this scheme, the per-core area grows linearly with the core count and the aggregated directory area grows quadratically, since the number of directory structures increases with the number of cores. Previous research works have focused on reducing directory area by focusing on the entry size.

To shorten the entry size some approaches use compression [1], [6], [7], [22]. In [1], [2] a two-level cache directory is proposed. The first-level stores the typical sharer vector while the second-level uses a compressed code. When using compression, area is saved at expenses of using an inexact representation of the sharer vector, thus yielding to performance losses. Hierarchical [11] representation of the sharer vector has been also used for entry size reduction purposes. However, hierarchical organizations impose additional lookups on the critical path so hurting latency. Sparse directories may reduce area by reducing the number of directory entries but at the expense of performance since directory evictions force invalidations at the core caches of the blocks being tracked.

Unlike typical sparse directories, SCD [25] uses different entry formats of the same length. Lines with one or a few sharers use a single directory entry while widely shared lines employ several cache lines (multi-tag format) using hierarchical bit vectors. This scheme requires extra complexity and accesses for managing dynamic changes (expanding/contracting) in the format.

Multi-grain directories (MGD) [33] also uses different entry formats of same length and tracks coherence at multiple different granularities in order to achieve scalability. Each MGD entry tracks either a temporarily private memory region or a single cache block with any number of sharers. Differently from the PS directory, this proposal is limited to a range of directory interleavings (those higher or equal to the size of a memory region) in order to achieve maximum benefits. MGD has been evaluated and compared to our PS directory, as shown in the evaluation section.

Finally, other proposals [9] focus on reducing the number of entries implemented in the cache directory instead of focusing on the sharer vector. While this approach does not affect the performance, it requires modifying the OS, the Page Table, the processor TLBs and the coherence protocol.

VIII. Conclusions

This work identifies five key characteristics that clearly differentiate the behavior of private and shared blocks from the directory point of view. Based on these observations, we introduce the PS directory, a directory cache that uses two different cache structures, each one tailored to one type of block (i.e., private or shared). The Shared directory cache, which tracks shared blocks is small, with low associativity and fast. The Private directory cache is aimed at tracking private blocks, which are highly dominant in current workloads. This structure does not store the sharer vector, is larger than the shared cache, and it is implemented with higher associativity.

Experimental results for a 16-core CMP show that, compared to a single directory cache with the same number of entries, the PS directory improves performance by 14% due to the separate treatment of private and shared blocks. Additionally, directory area is reduced by 26.35% mainly due to not storing the sharer vector for the private blocks, and by 33.98% when eDRAM technology is considered for the Private cache. Regarding energy consumption, reductions about 27% are achieved. Compared to the state-of-the-art MGD scheme, the PS directory increases the performance by 16.7% and reduces energy by 18.7%, being also much more scalable in terms of area. Thus, the proposal provides noticeable scalability in terms of area and energy with respect to the single-cache and the MGD directories, while also being able to surpass a single-cache directory in terms of performance scalability.

Finally, we would like to remark that the mentioned benefits are obtained with almost the same performance as the duplicate tags approach (i.e., perfect directory) but with a feasible implementation that scales in performance better than conventional approaches.

Acknowledgments

This work has been jointly supported by the MINECO and European Commission (FEDER funds) under the project TIN2012-38341-C04-01 and the *Fundación Seneca-Agencia de Ciencia y Tecnología de la Región de Murcia*, under the project *Jóvenes Líderes en Investigación* 18956/JLI/13.

References

[1] 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)*, Jan. 2001, pp. 97–106.

[2] M. E. Acacio, J. González, J. M. García, and J. Duato, "A two-level directory architecture for highly scalable cc-NUMA multiprocessors," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 16, no. 1, pp. 67–79, Jan. 2005.

[3] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *IEEE Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.

[4] L. A. Barroso, K. Gharachorloo, and R. McNamara, et al, "Piranha: A scalable architecture based on single-chip multiprocessing," in *27th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2000, pp. 12–14.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *17th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Oct. 2008, pp. 72–81.

[6] D. Chaiken, J. Kubiatowicz, and A. Agarwal, "LimitLESS directories: A scalable cache coherence scheme," in *4th Int'l Conf. on Architectural Support for Programming Language and Operating Systems (ASPLOS)*, Apr. 1991, pp. 224–234.

[7] G. Chen, "Slid - a cost-effective and scalable limited-directory scheme for cache coherence," in *5th Int'l Conf. on Parallel Architectures and Languages Europe (PARLE)*, Jun. 1993, pp. 341–352.

[8] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the AMD opteron processor," *IEEE Micro*, vol. 30, no. 2, pp. 16–29, Apr. 2010.

[9] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *38th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2011, pp. 93–103.

[10] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *17th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2011, pp. 169–180.

[11] S.-L. Guo, H.-X. Wang, Y.-B. Xue, C.-M. Li, and D.-S. Wang, "Hierarchical cache directory for cmp," *Journal of Computer Science and Technology*, vol. 25, no. 2, pp. 246–256, Mar. 2010.

[12] A. Gupta, W.-D. Weber, and T. C. Mowry, "Reducing memory traffic requirements for scalable directory-based cache coherence schemes," in *Int'l Conf. on Parallel Processing (ICPP)*, Aug. 1990, pp. 312–321.

[13] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "POWER7: IBMs next-generation server processor," *IEEE Micro*, vol. 30, no. 2, pp. 7–15, Apr. 2010.

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

[15] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *2005 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*, Jun. 2005, pp. 190–200.

[16] P. S. Magnusson, M. Christensson, and J. Eskilson, et al, "Simics: A full system simulation platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[17] M. M. Martin, D. J. Sorin, and B. M. Beckmann, et al, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sep. 2005.

[18] M. R. Marty and M. D. Hill, "Virtual hierarchies to support server consolidation," in *34th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2007, pp. 46–56.

[19] M. R. Marty and M. D. Hill, "Virtual hierarchies," *IEEE Micro*, vol. 28, no. 1, pp. 99–109, Jan. 2008.

[20] R. E. Matick and S. E. Schuster, "Logic-based eDRAM: Origins and rationale for use," *IBM Journal of Research and Development*, vol. 49, no. 1, pp. 145–165, Jan. 2005.

[21] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0," HP Labs, Tech. Rep. HPL-2009-85, Apr. 2009.

[22] B. W. O'Krafka and A. R. Newton, "An empirical evaluation of two memory-efficient directory methods," in *17th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 1990, pp. 138–147.

[23] A. Ros, M. E. Acacio, and J. M. García, "A scalable organization for distributed directories," *Journal of Systems Architecture (JSA)*, vol. 56, no. 2-3, pp. 77–87, Feb. 2010.

[24] A. Ros, B. Cuesta, R. Fernández-Pascual, M. E. Gómez, M. E. Acacio, A. Robles, J. M. García, and J. Duato, "Extending magny-cours cache coherence," *IEEE Transactions on Computers (TC)*, vol. 61, no. 5, pp. 593–606, May 2012.

[25] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *18th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2012, pp. 129–140.

[26] M. Shah, J. Barreh, and J. Brooks, et al, "UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC," in *IEEE Asian Solid-State Circuits Conference*, Nov. 2007, pp. 22–25.

[27] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner, "Power5 system microarchitecture," *IBM Journal of Research and Development*, vol. 49, no. 4/5, pp. 505–521, Jul. 2005.

[28] J. M. Tendler, J. S. Dodson, J. S. Fields, H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–25, Jan. 2002.

[29] A. Valero, J. Sahuquillo, S. Petit, V. Lorente, R. Canal, P. López, and J. Duato, "An hybrid eDRAM/SRAM macrocell to implement first-level data caches," in *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009, pp. 213–221.

[30] J. J. Valls, A. Ros, J. Sahuquillo, M. E. Gómez, and J. Duato, "PS-Dir: A scalable two-level directory cache," in *21st Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2012, pp. 451–452.

[31] 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)*, Jun. 1995, pp. 24–36.

[32] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *36th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2009, pp. 34–45.

[33] J. Zebchuk, B. Falsafi, and A. Moshovos, "Multi-grain coherence directories," in *46th IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2013, pp. 359–370.

[34] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *42nd IEEE/ACM Int'l Symp. on Microarchitecture (MICRO)*, Dec. 2009, pp. 423–434.

[35] H. Zhao, A. Shriraman, S. Dwarkadas, and V. Srinivasan, "SPATL: Honey, i shrunk the coherence directory," in *20th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2011, pp. 148–157.