A Directory Cache with Dynamic Private-Shared Partitioning

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

*Department of Computer Engineering
Universitat Politècnica de València (Spain)
joavalmo@fiv.upv.es, {jsahuqui,megomez}@disca.upv.es

†Dept. de Ingeniería y Tecnología de Computadores
Universidad de Murcia (Spain)
aros@ditec.um.es

***Abstract*—As the core counts increase in each chip multiprocessor generation, coherence protocols should improve scalability in performance, area, and energy consumption to meet the demands of larger core counts. Directory-based protocols constitute the most scalable alternative. A conventional directory, however, suffers from an inefficient use of storage and energy. First, the large, non-scalable, sharer vectors consume unnecessary area and leakage, especially considering that most of the blocks tracked in a directory are cached by a single core. Second, although increasing directory size and associativity could boost system performance, it would come at expenses of energy consumption.**

This paper proposes the Dynamic Way Partitioning (DWP) Directory, a directory structure that exploits three main workload characteristics to achieve area and energy reductions. First, it is widely known that even in parallel workloads most of the accessed cache blocks are private. Second, most directory accesses target the small number of shared blocks. Third, the shared/private ratio of entries in the directory varies across applications and across different execution phases within the applications. To take advantage of these three characteristics, DWP-Directory reduces the number of ways with storage for shared blocks and it allows this storage to be powered off or on at run-time according to the dynamic requirements of the applications.

DWP-Directory is compared to a conventional directory cache with different associativity degrees and with two state-of-the-art schemes: PS-Directory and Hybrid Representation. Experimental results for 32-core CMPs show that DWP-Directory achieves the best of both worlds: similar performance as a traditional directory with high associativity, and similar area as recent state-of-the-art schemes. In addition, DWP-Directory reduces static and dynamic power consumption by 38.0% and 67.4%, respectively compared to conventional sparse directories.

I. Introduction

As transistor technology miniaturizes, silicon resources become more abundant. Consequently, the core count is continually increasing in current chip-multiprocessors (CMP). These systems usually implement a shared memory programming model and a cache coherence protocol to maintain data coherence along the CMP memory hierarchy. Directory-based protocols are the common approach used in current systems over other alternatives such as snoop-based protocols, which generate an important traffic overhead due to the use of broadcast messages. Much research has concentrated on improving the performance and energy of directory protocols, both from the academia [1], [2], [3], [4], [5] and from the industry in modern processors [6], [7], [8], [9], [10]. Directory-based protocols require additional structures to keep track of the cached block. Two main approaches can be followed: *Duplicate Tags* [7] and *Sparse Directories* [11].

Sparse directories, implemented as a cache-like structure with a relatively low associativity degree, are the preferred design choice for a mid to high number of cores, since Duplicate Tags require highly associative lookups to build the *sharer vector* on each directory access and entail a high energy consumption. The limitation of sparse directories is that replacements are needed due to space constraints. Upon a directory entry eviction, all copies of the block —being tracked by such entry— in the cores' private caches are invalidated, regardless of whether the block is being used or not. Therefore, subsequent accesses to these invalidated blocks will rise the so-called *coverage misses* [12], which degrade system performance.

An entry in a conventional sparse directory mainly stores the *owner* of the block, required to find the provider of the block, and a *sharer* vector, required to track all copies of a shared block. While the owner field just requires $log_2 C$ bits, where C represents the number of cores, the sharer vector typically utilizes one bit per core which is set when the core's cache holds a copy of the block. Thus, the size of the sharer vector, and so that of the directory, grows linearly with the number of cores. Consequently, as the current industry trend is to increase the core count in each CMP generation, it is expected that the directory size will present a worth on-chip area and leakage overhead in future CMPs [13]. Therefore, there is a need for new directory schemes that scale in terms of area and power.

The key challenge when addressing scalability in sparse directories lies on reducing the overhead in area and power introduced by the sharer vector. This fact has been effectively addressed based on the characteristics of the blocks being tracked. Some previous works [3], [14] have realized that most blocks are accessed by a single core. That is, a high amount of blocks are fetched into the cache of a given core and then no other core accesses it. These blocks are referred to as *private blocks* in contrast to *shared blocks*, which are accessed by multiple cores. This behavior means that most directory entries keep track of private blocks, which do not require from coherence actions, thus these entries do not use the sharer vector field at all. Based on this finding, recent proposals [15], [16], implement two kinds of entries in the directory: *shared* and *private*. The former include storage for a shared vector and can potentially track shared blocks, while the latter save storage by not including sharing information and are limited to track private blocks. The main drawback of these schemes is that the number of private and shared entries is fixed by design. However, as we show in this work

the requirements of private and shared directory entries widely varies both across applications and intra application. To face the mentioned drawback this paper proposes a directory that adapts the number of shared entries according to the run-time demands of each application.

This paper makes the following contributions:

- We perform a workload characterization and find that the number of shared blocks widely varies at run-time both intra and inter applications.
- We propose the Dynamic Way Partitioning (DWP) directory, to the best of our knowledge, the first directory that dynamically chooses the proper number of shared entries at run-time according to the workload requirements at each phase of its execution.
- DWP-Directory achieves better performance than state-of-the-art directory schemes that exploit asymmetric storage for block tracking. Experimental results for 16- and 32-core CMPs show that compared to conventional directory schemes with the same number of entries, DWP-Directory is able to achieve important area, dynamic and static energy consumption reductions, while having an almost negligible impact on performance.

II. Background and Motivation

A. Asymmetric Storage for Handling Shared and Private Blocks

Different approaches have been proposed to reduce the directory size. Recently, some works [17], [15], [16], [18], [19] have focused on providing *asymmetric storage* for handling shared and private blocks. Area savings come from making the directory narrower by using shorter entries —the sharer vector is not implemented— to track private blocks. These works demonstrate that actively differentiating shared and private entries can yield the system to area and energy improvements over a conventional *one-type entry* directory.

The PS-Directory [15], [18] provides a fast and small (low number of entries) directory in SRAM for the reduced number of frequently accessed *shared* entries, and a larger (more entries) and slower directory in a denser eDRAM cache for infrequently accessed *private* entries. This approach allows entries to move from the *private* directory to the shared one, which is the most frequently accessed. Once one entry becomes shared, however, it does not return to the private directory even if the block being tracked is only stored in a single core. The rationale behind this design feature is that a block that has been shared has a high probability of being shared again, and moving it from one directory to the other consumes extra energy and does not translate into performance improvements.

The Hybrid Representation directory [16], [19] also considers a different representation for private and shared entries. The key difference, however, is that the latter approach proposes a single-cache directory and both types of entries are mingled in the same cache structure. Unlike the previous scheme, the contents of a private entry are permitted to move to a shared one and vice versa, according to the state of the block.

Both aforementioned approaches conclude that, based on the average workload behavior, the most efficient directory is that providing a quarter of its entries to track shared blocks and three quarters to track private ones. The main drawback in both approaches is that both shared and private entries are limited by design, that is, private blocks compete among them for private ways and analogously shared blocks for shared ways. Therefore, if the run-time requirements of a given application exceeds the budget of ways available for a given type of block, that requirement cannot be satisfied by design so yielding the system to performance drops.

In summary, although discerning among shared and private entries can bring important benefits in terms of area and energy, static designs like PS-Directory or Hybrid Representation cannot adapt to the different shared-private ratio of parallel applications and to every execution phase within the application, thus providing sub-optimal performance.

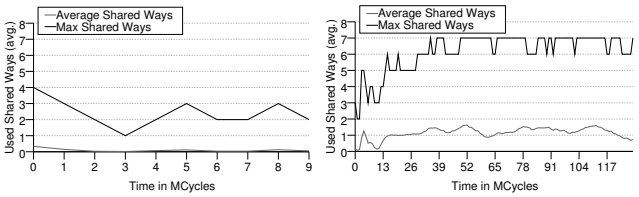
B. Motivation

This section characterizes the applications used in our evaluation (Section IV) by studying the dynamic requirements of shared entries at run-time. The study shows that while at some point in time some applications may require a single shared entry in a set, some others may require almost all the entries in a set to track shared blocks. To deal with this behavior, this paper proposes a flexible structure that dynamically varies the number of *active* shared entries according to the run-time demands of the workloads.

As a first design step, we analyze the dynamic requirements of shared directory entries across a representative subset of parallel workloads in order to find out how many shared entries should be supported to achieve the same performance as a conventional directory. For this purpose, we ran parallel workloads and for each of them we measured the number of entries actually tracking shared blocks along the execution time (see Section IV for simulation details).

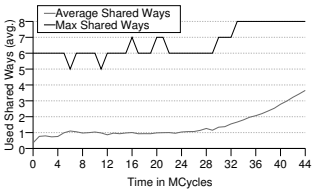
According to dynamic variability in the run-time demands of shared entries, there are some differences between applications, yet some general observations can be concluded. Figure 1 plots the dynamic evolution of the number of shared entries averaged across all the directory sets and banks, and the maximum number of shared entries in any set for each application assuming a 8-way directory cache.

It can be observed that, a static approach with $S = 2$ and $P = 6$, the best one in PS-Directory and Hybrid Representation, fails to adequate to specific directory sets at a given point in time, since typically there is always one (i.e. labelled as `Max` in the plots) or some sets that require more than two ways for shared entries. Yet, most of the applications have scarce set requirements, on average, to track shared blocks. Only `Radiosity` and `LU` require on average more associativity to track shared blocks than the deployed in the aforementioned proposals, but only during a small fraction of its execution time. This will inevitably lead to performance losses. Therefore, the solution to improve performance lies on adding extra shared entries. However, this way also would be at cost of area and energy expenses, thus, key challenge lies on investigating the number of entries an efficient directory should deploy in order to achieve the best area and energy savings while sustaining the performance of a conventional all shared-entry directory. On the other hand, notice that there are also many other applications which do not need more than one

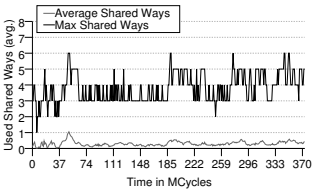


(a) FFT

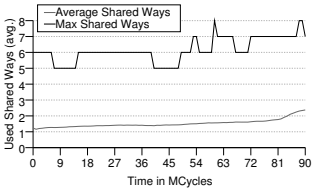
(b) FMM



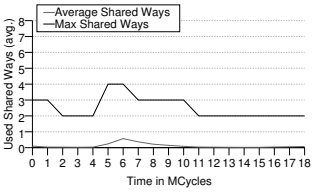
(c) LU



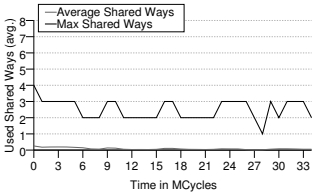
(d) Ocean



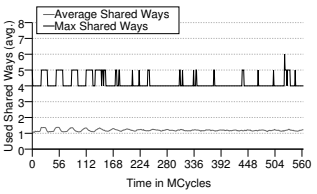
(e) Radiosity



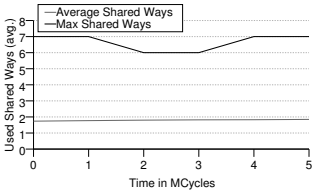
(f) Radix



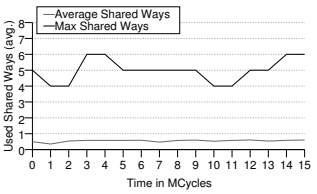
(g) Tomcatv



(h) Unstructured



(i) Volrend



(j) WaterNsq

Fig. 1: Average and maximum number of shared entries per set over the execution time across all the directory banks.

shared entry per set for most of its execution time (i.e. `FFT`, `Ocean`, `Radix`, `Tomcatv` and `Waternsq`). The additional shared associativity in the directory is not required in these cases, which in turn brings additional energy consumption and area that could be otherwise avoided.

To provide deeper insights in the most adequate number of ways, we quantified the fraction of time the directory is keeping track any given number of shared blocks. Figure 2 shows the results across the studied benchmarks.

It can be seen that, on average, two or less directory cache ways able to keep track of shared blocks are required during 93.8% of the execution time, while only during a 3% of it more

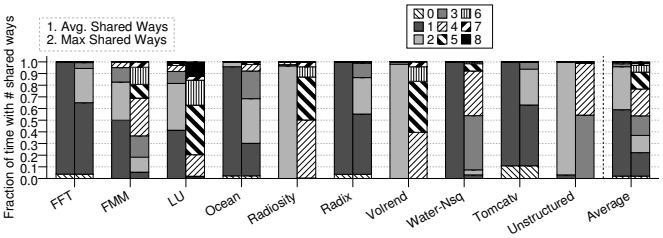


Fig. 2: Fraction of time with # shared entries in a set.

than four shared entries are in demand. Regarding maximum requirements in individual sets, it can be appreciated that, on average, during 76.8% of the execution time, there are no individual sets requiring more than four shared entries. This value makes sense since by definition, a shared block must be stored in at least two L1 caches, but since workloads are not ideally balanced, sometimes the accesses may concentrate on specific directory banks or sets. We experimentally found that these happens in some workloads like `Radiosity`.

The previous analysis, as well as experimental results will confirm, shows that a directory with a quarter or half of its ways providing storage for shared blocks are the most interesting design choices, and can provide the best tradeoff between performance, area and energy.

In accordance to these results, we analyze two approaches in which a quarter or half of the cache ways in a 8-way directory provide support for shared entries, while the remaining ways only support private entries. Since most of the time at most two shared ways are required, this only incurs performance losses during a negligible percentage of time. This results in important benefits in terms of area and energy, especially leakage, as discussed in the next section.

III. DWP-Directory

The design of DWP-Directory is mainly motivated by two observations discussed in Section II: i) there are applications that need more than 3 or 4 shared ways during some phases of their execution and ii) there are applications that require nearly all the ways to track private blocks. These observations are not supported for state-of-the-art directory approaches.

Keeping these observations in mind, the main goal of DWP-Directory is to provide support for both of them. Figure 3 depicts the structure of a generic DWP-Directory. Two types of entries are deployed: those having storage space to contain the sharer vector and those lacking the sharer vector. The directory deploys N shared entries and $M - N$ private entries per set, where M is the total associativity. Three areas can be appreciated: the most-left way is always shared, the $M - N$ most-right ways are always private and the rest of the ways in the middle may contain shared or private entries (i.e. repartitionable area, highlighted in gray). An entry in the repartitionable area include the `On/Off bit` that is set when the associated way is tracking shared blocks and reset when it tracks private blocks.

When the bit is reset, the voltage supply to the sharer vector is disabled since private blocks do not need it. Notice that this allows energy savings, mainly leakage, with no performance penalty. In other words, with this design i) the private blocks do not consume the energy dissipated to hold the sharer

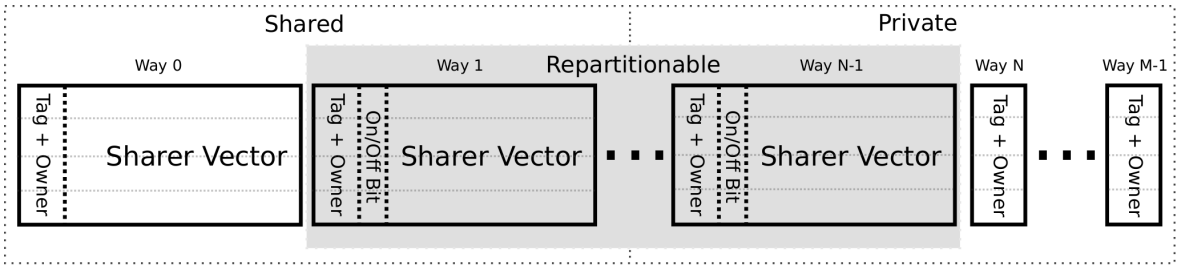


Fig. 3: The DWP-Directory architecture.

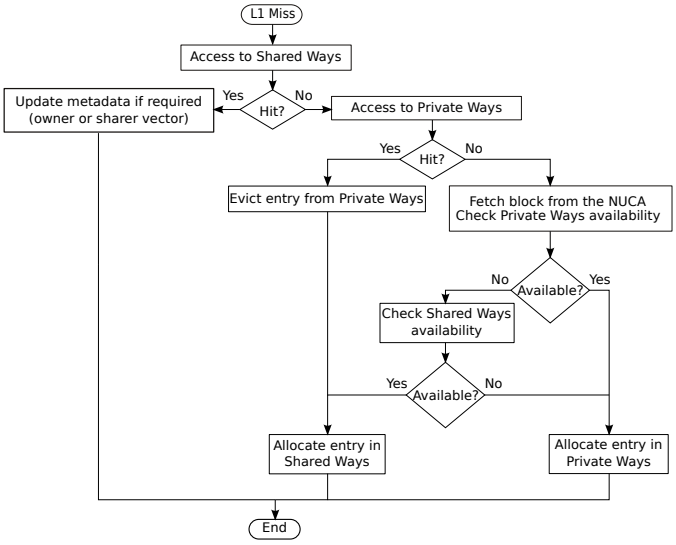


Fig. 4: DWP-Directory working flow chart.

vector, and ii) the directory size becomes smaller due to the removal of the sharer vector in part of its ways. An entry in a traditional directory for a MOESI protocol, apart from the tags, is comprised of an owner and a sharer vector field that require $(log_2(C)+C)$ bits, being C the number of cores in the CMP. The higher the number of cores the larger the number bits that can be saved with our proposal, i.e. $(M-N)\times C$ bits per set. To this amount, we should subtract a few N bits per set required for `On/Off bits`. The higher the number of cores the wider the sharer vector field since it requires one bit per core. Hence, DWP-Directory scales much better in terms of energy and area than sparse directories.

In summary, unlike existing proposals, which hardly limit the number of shared ways to 2 and private ways to 6, DWP-Directory implements a flexible sparse directory that can use all the ways to track private blocks, and is able to track as many shared blocks as deployed sharer vectors.

A. Basic Working Behavior

DWP-Directory includes two types of entries: private and shared. Private entries are short, do not include the sharer vector, and are only able to keep track of private blocks. Shared entries are wider, implement the sharer vector, and can keep track of either shared and private blocks. Figure 4 depicts a flow chart that summarizes how DWP-Directory handles private and shared entries. On a miss in the L1 cache of a given core, the directory is accessed in order to maintain coherence. In a traditional directory, all the cache ways in the directory are accessed in parallel which translates into highly consuming searches.

To reduce dynamic energy consumption, the first lookup in DWP-Directory only accesses the subset of ways tracking shared blocks. The reason to look up first these ways is that most of the accesses to the directory are to shared blocks [18]; thus, it is more likely to find the required block in the shared entries. Moreover, as discussed in Section II, the number of active shared entries in the directory is on average lower or much lower than the number of private ways, so important energy savings can be achieved.

Upon a miss in the first lookup, DWP-Directory searches the target block in the remaining entries, i.e. private entries. If there is a hit in any of these ways, this means that the requesting core differs from the owner of the block, thus the block should become shared and the entry moved to a shared way. In case no shared entry is available, an entry should be evicted and all the copies of the block in the processor caches should be invalidated. Even though DWP-Directory has potentially no limitation in the minimum number of shared ways, this work does not evaluate the option of supporting no shared ways since the complexity of the coherence protocol increases. Notice that if there is no active shared way (i.e. all sharer vectors are deactivated), the previous owner of the block is invalidated and the new owner updated accordingly. New transitions are required in the protocol to take this case into account, while DWP-Directory ensuring at least one shared way can work directly with the conventional coherence protocol. This case would be accounted as a shared entry eviction for the repartitioning algorithm as explained below.

If both directory lookups miss, a new entry is allocated. This entry is set as private since it only tracks a single copy. If the directory has an available entry, the new entry is allocated on it, prioritizing private entries over shared entries in case there are several available entries. If all the entries are busy, the directory controller has to evict one of them. In such a case, the least recently used way, independently of being private or shared, is selected for eviction.

B. Repartitioning Approach

DWP-Directory dynamically repartitions the number of shared entries enabled to keep track of shared and private blocks considering the run-time application needs. In other words, some of the shared entries by design are considered as private and its sharer vector field powered off for leakage savings. After a given number of accesses to the directory, DWP-Directory analyzes the eviction ratio between shared and

private blocks and the number of private ways is readjusted taking into account the physical constraints.

The repartitioning mechanism is implemented with negligible hardware with only three main parameters. These parameters help the algorithm in decision taking about when a repartitioning should be triggered as a consequence of an increase or decrease of the demand of shared ways: a *interval length (IL)*, a *shared threshold (ST)* and a *private threshold (PT)*. The selection of *IL* is quantified in number of accesses to the directory.

Algorithm 1 summarizes the pseudocode of the reconfiguration mechanism. This algorithm is *called* on every directory access. Two global counters are used: *directory_accesses* and *ctr*. The former accounts for the number of accesses to the directory. The latter is an up/down counter that saturates at an upper threshold *PT* and at a lower threshold *ST*. Small top/down counters have a low implementation complexity and have been widely applied in the past, hence this design choice has been selected.

```
//For every access to the directory
directory_accesses++;
if (ctr != PT && ctr != ST) { //Ctr not saturated
  if (private_eviction_required) {
    ctr++;
  } else if (shared_eviction_required) {
    ctr--;
  }
}

if (directory_accesses == IL) {
  if (ctr == PT && shared_ways > 1) {
    private_ways++;
    shared_ways--;
  } else if (ctr == ST && shared_ways < N) {
    private_ways--;
    shared_ways++;
  }
  reset(); // Resets all counters
}
```

Algorithm 1: Repartitioning algorithm

The algorithm works as follows. When the directory is accessed for *IL* times, the repartitioning logic checks the value of the *ctr* counter to decide if the number of shared ways should be increased, decreased or remain in its actual value.

- Each time a private entry is evicted from the directory, the *ctr* counter is increased and is decreased each time a shared entry is evicted.
- When the *directory_accesses* counter reaches IL:
  - If the counter saturates at its lower threshold *ST*, then additional shared entries are required. Thus, the most-left shared entry tracking a private block (Figure 3) is set as shared and its shared vector activated.
  - If the counter saturates at *PT*, then directory needs additional private ways in detriment of shared ones. In such a case, the most-right shared way in the *repartitionable area* (Figure 3) is set to private. Thus, its sharer vector is powered down and all sharers but the owner are sent an invalidation message.
  - If the counter is not saturated, then the system remains in its actual state for further *IL* accesses.
  - *ctr* and *directory_accesses* are reset to 0.

TABLE I: 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

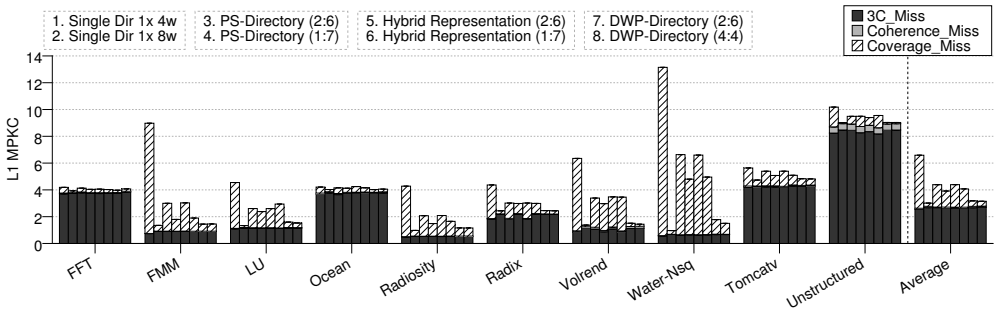
This algorithm allows the proposal to dynamically adapt to the application phases, providing leakage savings without affecting performance. The reconfiguration of a way is done in all sets of the directory simultaneously in order to minimize complexity and to guarantee a very simple first lookup in the directory. Notice that the cost of evicting shared entries is higher than the cost of evicting private entries, but that is taken into consideration when choosing the PT and ST values.

IV. Simulation Environment

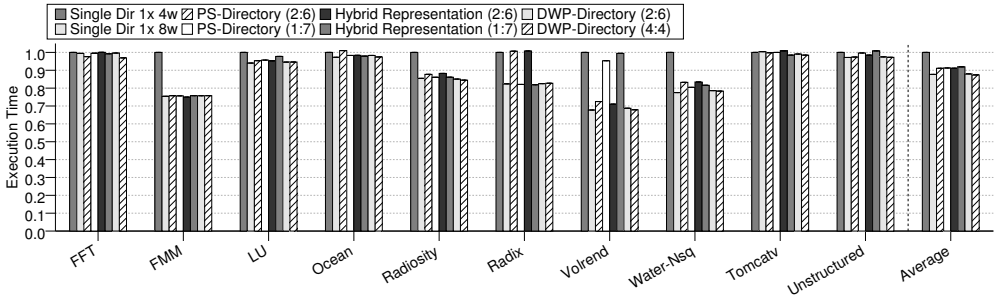
DWP-Directory is evaluated using full-system simulations with Simics [20] and GEMS [21], which enables detailed simulation of multiprocessor systems. The interconnection network is modeled using GARNET [22]. We evaluate both 16- and a 32-core CMPs comprised of a cache hierarchy with private L1 caches and a shared L2 NUCA distributed among all tiles. A MOESI directory-based cache coherence protocol keeps coherence for the data within the private caches. 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 they will have in the directory. Our base directory scheme is an on-chip distributed sparse directory with a bit-vector sharing code in each entry. Other baseline system parameters are shown in Table I. We use CACTI 6.5 [23] to estimate access time, area requirements and power consumption of the different cache structures for a 32nm technology node.

DWP-Directory is evaluated using a wide range of scientific applications. *FFT* (64K complex doubles), *FMM* (16K particles), *LU* (512×512 matrix), *Ocean* (514×514 ocean), *Radiosity* (room, -ae 5000.0 -en 0.050 -bf 0.10), *Radix* (512K keys, 1024 radix), *Volrend* (head), and *Water-Nsq* (512 molecules) are from the SPLASH-2 benchmark suite [24]. *Tomcatv* (256 points) and *Unstructured* (Mesh.2K) are two scientific benchmarks. The experimental results reported in this work correspond to the parallel phase of the evaluated benchmarks.

DWP-Directory is sensitive to both the directory configuration and the threshold parameters. We have tested many configurations, however, given the analysis shown in Section II-B only results for two most effective configurations are presented. One configuration implements half of its 8 ways without the sharer vector field, hereby noted as *DWP-Directory (4:4)*, while the other implements the sharer vector in two of



(a) L1 Misses per kilocycles and per core



(b) Execution Time

Fig. 5: Performance normalized with respect to a single-cache directory with 4 ways and 16 cores.

them, hereby noted as *DWP-Directory (2:6)*. Both configurations share an *interval length (IL)* of 500 directory accesses, a *shared threshold (ST)* of 10 and a *private threshold (PT)* of 100. These thresholds were tuned to the studied workloads, showing minor differences for thresholds relatively high, but due to space constraints no sensitivity analysis is presented.

V. Experimental Evaluation

This section evaluates DWP-Directory against a 4-way conventional or single-cache directory (which acts as the baseline), a 8-way conventional directory, and two state-of-the-art architectures: PS-Directory and Hybrid Representation.

Unlike our proposal, the directory space assigned to each type of block in the aforementioned approaches is fixed and cannot be changed at run-time according to the needs of each particular workload during its execution.

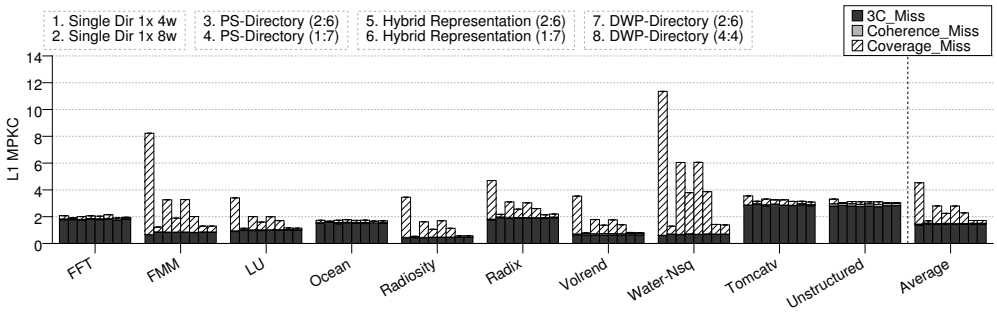
Notice that all evaluated schemes, with the only exception of the baseline, implement a 8-way directory associativity. Both state-of-the-art architectures dedicate two ways to track shared blocks and the remaining ones to track private blocks (2:6 configuration). Since some workloads require a single shared way most of its execution time, as shown in the next section, a 1:7 configuration is also implemented for comparison purposes.

A. Impact on Performance

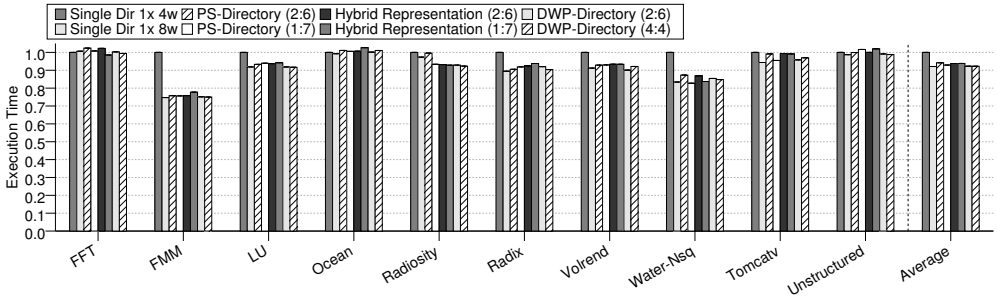
The impact of the proposal on performance has been evaluated by analyzing the L1 Misses per kilocycles (MPKC) and the execution time. Every time a directory entry is evicted, invalidation messages are sent to the corresponding processor caches keeping a copy of the block being tracked in order to be able to maintain cache coherence. These invalidations will cause coverage misses upon a subsequent memory request to those blocks, thus impacting on the final performance. Figure 5a shows the L1 MPKC, which matches the number of directory accesses per kilocycles, with respect to a 4-way single-cache directory in the studied 16-core CMP. The misses have been categorized in three types: 3C (capacity, compulsory and conflict), coherence and coverage.

The different evaluated schemes have negligible impact on 3C and coherence misses over the baseline. On the other hand, the aggregated associativity degree of the directory, as expected, has a big impact on the number of coverage misses. An increase from 4 to 8 ways in a single cache greatly decreases the number of coverage misses, approaching to the optimum performance that an ideal directory can achieve. The additional associativity allows more flexibility when keeping track of both shared and private entries in a set. Notice that even though most of the blocks are private and would hence require a higher number of entries, they are scarcely accessed, in comparison to shared ones, so they can be prematurely evicted due to an LRU replacement policy, when space constraints problems arise. Thus, additional associativity mitigates this problem.

Regarding the state-of-the-art schemes, the PS-Directory reduces the number of misses by 34.5% and 40.6% for the 2:6 and 1:7 configurations, respectively. Hybrid Representation reduces this number by 34.3% and 38.2%. These reductions are achieved due to the different treatment of private and shared blocks. Since the associativity degree is partitioned, entries do not have the same allocation flexibility as a single-cache directory with the same associativity. Notice that configuration 1:7 obtains the best results, since as discussed above, most of the applications present a low associativity requirement for



(a) L1 Misses per kilocycles and per core



(b) Execution Time

Fig. 6: Performance normalized with respect to a single-cache directory with 4 ways and 32 cores.

shared entries. Yet, there are some exceptions in which the 2:6 configuration works best, *e.g.* in *LU* and *Unstructured* for the Hybrid Representation. Hence it can be seen that there is no optimal static configuration that satisfies every workload.

DWP-Directory, which unlike the aforementioned schemes has the ability to adapt the private-shared partition size dynamically at run-time, obtains better results, reducing the number of misses by 49.8% and 50.4% in the 2:6 and 4:4 configurations, respectively. It performs similar as an 8-way single cache, with only 1% degradation. Notice that following the characterization presented in section II-B, those applications with a higher maximum number of shared ways benefit the most from our proposal, compared to the state-of-the-art schemes. On the other hand, applications with low shared requirements, do not benefit as much. The dynamic adaptability allows DWP-Directory a similar flexibility as the single-cache directory, while also keeping or improving most of the benefits that provide the differentiation between shared and private entries in terms of area and energy reduction, as will be discussed below.

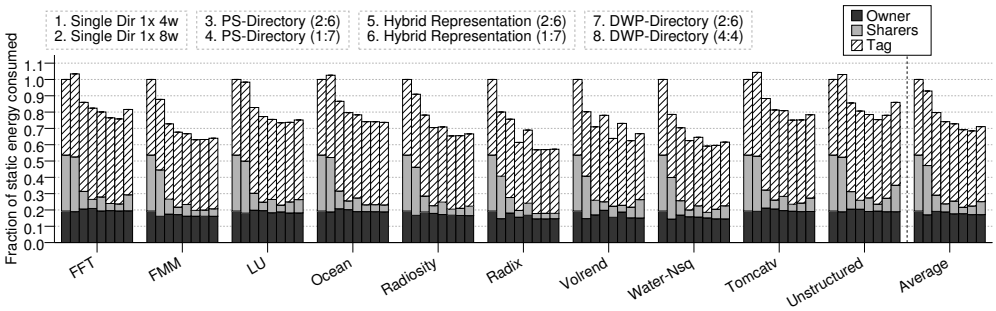
Reducing the number of L1 misses translates into a lower execution time of the applications, as shown in Figure 5b. The reduction of misses achieved by the 8-way single-cache directory improves the execution time by 12.3%. The PS-Directory and Hybrid Representation both reduce the applications average execution time by 8.9%. Meanwhile, DWP-Directory reduces the execution time by 12.1% and 12.7% in the 2:6 and 4:4 configurations, respectively. As expected, applications with low MPKC values are the ones that have a lesser improvement in their execution time. Power-up and power-down delays of the proposal are taken into account in these results.

To explore how the proposal behaves on a higher number of cores, we launched experiments for a CMP with 32 cores. Figure 6a and Figure 6b show the L1 MPKC and the execution time, respectively. Results are similar as those presented for 16 cores. While the 8-way single cache reduces misses by 51%, DWP-Directory 2:6 and 4:4 reduce them by 50.4% and 50.1%, respectively. The difference between our proposal and the 8-way single cache is smaller. In terms of execution time it translates into a reduction of 7.9%, 7.7% and 7.7%, respectively. The state-of-the-art architectures achieve lower reductions, but as with 16 cores, a 1:7 shared-to-private way ratio performs on average slightly better than a 2:6 one.

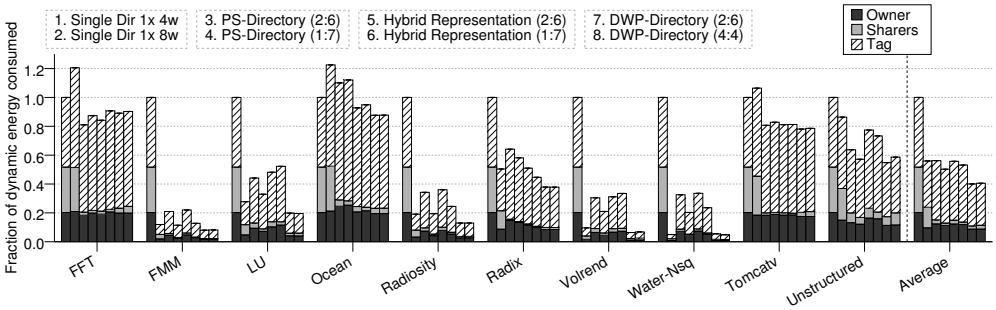
B. Impact on Energy Consumption

Typically, static or leakage energy dominates the total energy consumption of the directory structure. Figure 7a shows the normalized leakage energy consumed by the directory structure with respect to the 4-way single cache.

As can be seen, the 8-way single-cache directory reduces leakage by 7.1%, mainly due to the smaller execution time of the applications. The PS-Directory and the Hybrid Representation (2:6) achieve better energy savings by 20.3% and 27.2%, respectively, even though their execution time is slightly worse than the 8-way single-cache directory. These energy savings are the result of both schemes lacking the sharer vector field in some ways, namely those designated to keep track of private blocks, regardless if they are in a separate structure, like in the PS-Directory, or in the same set, as in Hybrid Representation. This allows the directories to consume less static energy, while the execution time of the application is not severely harmed as



(a) Static Energy



(b) Dynamic Energy

Fig. 7: Normalized energy consumed with respect to a single-cache directory with 4 ways and 16 cores

shown in the previous section. For this reason, configurations 1:7 consume even less energy, since the sharer vector is present in one way less. DWP-Directory reduces the static energy consumed by 31.5% and 28.9% for 2:6 and 4:4 configurations, respectively, which are the highest reductions of the evaluated directories. Notice that these leakage savings over state-of-the-art proposal come thanks to its repartitioning mechanism that allows DWP-Directory provisioning more shared ways when needed of even actually using none of them.

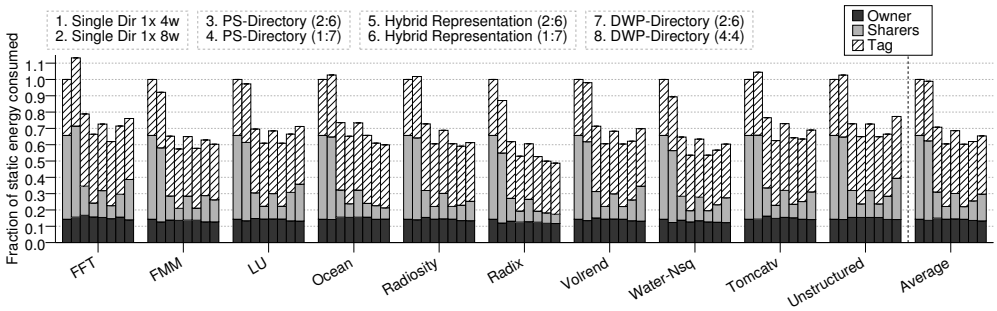
Results for the dynamic energy are shown in Figure 7b, also normalized with respect to a 4-way single cache. All the studied schemes, apart from DWP-Directory, achieve on average a similar energy savings falling in between 44% and 50% over the baseline. The best scheme regarding this parameter greatly fluctuates between the applications, so there is no definitive best approach. Meanwhile, with the only exception of FFT, DWP-Directory always achieves the better results. The consumption is reduced by 59.9% and 59.5% for the 2:6 and 4:4 configurations, respectively.

With 32 cores, in addition to maintaining similar performance ratio as in 16 cores, the proposal is able to achieve even better energy savings, offering a much scalable solution. Figure 8a and Figure 8b show the static and dynamic energy consumed in the 32 core CMP and normalized with respect to the 4-way single-cache. The leakage energy consumed by the 8-way single cache is only 1.1% better, despite the lower execution time. Meanwhile, the PS-Directory and Hybrid Representation 2:6 are able to reduce up to 29.3% and 31.3%, respectively, of this consumption. The energy savings are higher than those of the 16 core CMP mainly due to the larger amount of deployed sharer vectors. Since the mentioned schemes rely on the removal of the shared entry field, and this field increases its size with the number of cores, the overall number of bits that are eliminated is also higher. Lastly, DWP-Directory is able to reduce up to 38% and 34.6% of the leakage energy consumed by the directory structure for the 2:6 and 4:4 configurations, respectively.

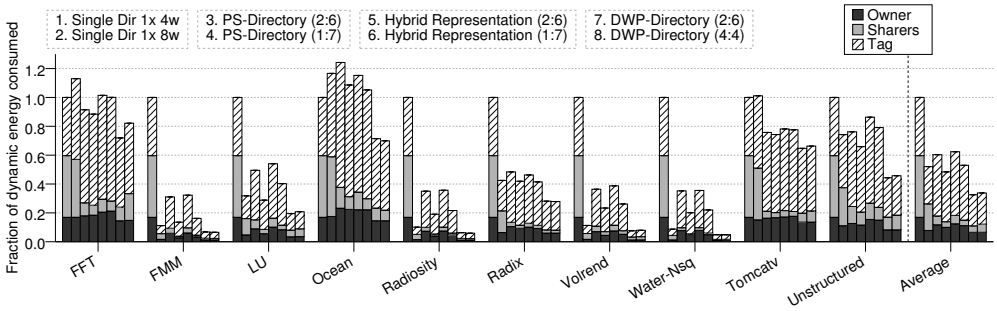
Regarding dynamic energy, DWP-Directory is able to reduce up to 67.4% and 66.2% for the 2:6 and 4:4 configurations, respectively, of the dissipated power, which is the highest across all the evaluated schemes.

C. Impact on Area Requirements

The on-chip area required to implement these directory structures is also analyzed in this section. Results obtained with CACTI are shown in Figure 9 for a conventional or single directory cache, the PS-Directory, Hybrid Representation and the proposal. With a higher number of cores, the area requirement difference between the single cache and the proposal grows more and more. DWP-Directory 4:4 requires only the 82.9%, 74.4% and 66.8% area that a single cache would need. The PS-Directory, Hybrid Representation and DWP-Directory 2:6 scale better and similar to each other, specially with 64 cores, than the 4:4 configuration, though. This is mainly because DWP-Directory 4:4 evaluated has a maximum of 4 shared ways, while the others only have 2. As results have shown, for a lower number of cores (i.e. 16 cores) 4 shared ways offer the best best performance albeit



(a) Static Energy



(b) Dynamic Energy

Fig. 8: Normalized energy consumed with respect to a single-cache directory with 4 ways and 32 cores

with a small energy and area penalty with respect to a DWP-Directory with just 2 shared ways. Overall, DWP-Directory with the 2:6 configuration offers the best tradeoff between performance, energy and area.

VI. Related Work

In shared memory systems where multiple cores are allowed to access the same memory blocks, cache coherence is a necessity. This work focuses on directory-based protocols, which are the commonly adopted solution for a medium to large core count.

Traditional directory schemes do not scale properly with the number of cores. One of today's major design concerns is the implementation of directories that scale to hundreds of cores in terms of power and area. Directory implementations, both in academia and industry, follow two main approaches: duplicate-tag directories and sparse directories.

Duplicate-tag directories keep a copy of the tags of all tracked blocks. Therefore, this approach does not raise any directory-induced invalidation nor coverage miss. Duplicate-tag directories have been implemented in modern small CMP systems [6], [8] and is the focus of recent research works [25], [13]. Although being area-efficient, obtaining the sharer vector requires multiple directory entry lookups, equal to the product of the number of core caches by the associativity of such caches [26]. That means that in a system with 64 8-way L1 caches, a directory access requires a 512-associative search. Hence, this approach becomes prohibitive for a larger number of cores.

Sparse directories [11] are organized as a set-associative cache like structure indexed by the block address. By reducing the directory associativity, this approach becomes more power-efficient than duplicate-tag directories. Sparse directories can reduce area by reducing the number of directory entries. This is done at the expense of performance, since each directory eviction due to lack of space, forces invalidations at the core caches of the blocks being tracked. Some works [27] employ block replication and migration to enhance performance.

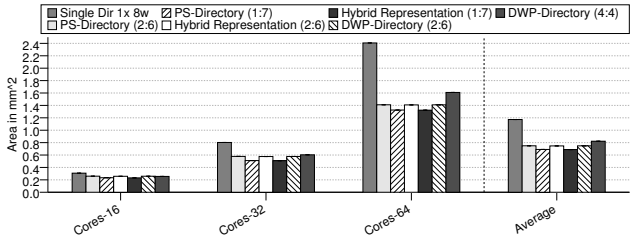


Fig. 9: Area required for the different directories with an increasing number of cores.

Previous research works have focused on reducing the directory area by focusing on the entry size. Some approaches have used compression [28], [29], [30], [31] to shorten the entry size. In [28], [32] a two-level cache directory is proposed. In the first level, the typical sharer vector is stored as usual, while the second level uses a compressed code instead. In these schemes, area is saved at expenses of using an inexact representation of the sharer vector when using compression. This induces potential performance losses.

Guo et al. [33] proposed a hierarchical representation of the sharer vector, also for entry size reduction purposes. Latency increases in these hierarchical organizations however, since they impose additional lookups on the critical path.

Others, like SCD [4], use different entry formats of the same length in order to solve the scalability problem. Unlike typical sparse directories where all lines share the same format, lines with one or few sharers use a single directory entry while shared lines employ several cache lines (multi-tag format) using hierarchical bit vectors. The proposed scheme entails extra complexity and directory accesses for managing the dynamic changes (expanding/contracting) in the entry format.

Multi-grain directories (MGD) [5] also use different entry formats of same length and track coherence at multiple different granularities in order to achieve scalability. Each entry in the MGD tracks either a single cache block with any number of sharers, as usual, or a temporarily private memory region.

Finally, Coherence Deactivation [3], [34], [35], [36], [37] improves the efficiency of the directory through OS-, TLB-, and compiler-based techniques, by removing the need of tracking private data at the directory. Differently, DWP-Directory focuses on shared entries and is transparent to these aspects.

VII. Conclusions

This work has identified that the current needs of multithreaded applications, regarding shared and private data access from the directory point of view, varies dynamically with execution time. Static private-shared structures are not able to properly adapt to this dynamic variation and, instead, dynamic strategies are in demand. Based on these observations, we have introduced the Dynamic Way Partitioning(DWP) Directory, a sparse directory that sacrifices the sharer vector field from part of its ways in order to gain in both area and energy scalability. Furthermore, the implemented sharer vectors can be powered off or on as required according to wether the need of more shared ways rises or drops at run time, respectively.

Experimental results for a 16-core CMP show that, compared to a conventional directory cache with the same number of entries, DWP-Directory reduces the static and dynamic energy consumed by 31.5% and 59.9%, respectively, while having an almost negligible performance penalty when compared to a more energy and area demanding 8-way conventional cache, and having a lower execution time than a more power-efficient 4-way directory.

Acknowledgments

This work has been jointly supported by MINECO and European Commission (FEDER funds) under the project TIN2015-66972-C5-1-R, TIN2015-66972-C5-3-R, and TIN2014-62246-EXP, and by *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. 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.

[2] A. Ros, M. E. Acacio, and J. M. García, *Parallel and Distributing Computing*. IN-TECH, Jan. 2010, ch. Cache Coherence Protocols for Many-Core CMPs, pp. 93–118.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] A. K. Nanda, A.-T. Nguyen, M. M. Michael, and D. J. Joseph, "High-throughput coherence control and hardware messaging in Everest," *IBM Journal of Research and Development*, vol. 45, no. 2, pp. 229–244, Mar. 2001.

[8] 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.

[9] 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.

[10] "Intel Xeon Phi Coprocessor," Apr. 2013. [Online]. Available: http://software.intel.com/en-us/mic-developer

[11] 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.

[12] A. Ros, B. Cuesta, R. Fernández-Pascual, M. E. Gómez, M. E. Acacio, A. Robles, J. M. García, and J. Duato, "EMC^2: Extending magny-cours coherence for large-scale servers," in *17th Int'l Conf. on High Performance Computing (HiPC)*, Dec. 2010, pp. 1–10.

[13] 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.

[14] 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.

[15] 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.

[16] L. Fang, P. Liu, Q. Hu, M. C. Huang, and G. Jiang, "Building expressive, area-efficient coherence directories," in *22nd Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2013, pp. 299–308.

[17] J. H. Kelm, M. R. Johnson, S. S. Lumettta, and S. J. Patel, "WAYPOINT: Scaling coherence to thousand-core architectures," in *19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2010, pp. 99–110.

[18] J. J. Valls, A. Ros, J. Sahuquillo, and M. E. Gómez, "PS directory: a scalable multilevel directory cache for cmps," *The Journal of Supercomputing*, vol. 71, no. 8, pp. 2847–2876, 2015.

[19] P. Liu, L. F. M. C. Huang, , Q. Hu, and G. Jiang, "Building expressive and area-efficient directories with hybrid representation and adaptive multi-granular tracking," *IEEE Transactions on Computers (TC)*, May 2015.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] A. Ros, M. E. Acacio, and J. M. García, "Scalable directory organization for tiled CMP architectures," in *Int'l Conference on Computer Design (CDES)*, Jul. 2008, pp. 112–118.

[26] ——, "A scalable organization for distributed directories," *Journal of Systems Architecture (JSA)*, vol. 56, no. 2-3, pp. 77–87, Feb. 2010.

[27] S. Bartolini, P. Foglia, C. A. Prete, and M. Solinas, "Coherence in the cmp era: Lesson learned in designing a llc architecture," *WSEAS Transactions on Computers*, vol. 13, pp. 195–206, 2014.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] B. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Increasing the effectiveness of directory caches by avoiding the tracking of non-coherent memory blocks," *IEEE Transactions on Computers (TC)*, vol. 62, no. 3, pp. 482–495, Mar. 2013.

[35] A. Esteve, A. Ros, M. E. Gómez, A. Robles, and J. Duato, "Efficient tlb-based detection of private pages in chip multiprocessors," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 3, pp. 748–761, Mar. 2016.

[36] A. Esteve, A. Ros, A. Robles, M. E. Gómez, and J. Duato, "TokenTLB: A token-based page classification approach," in *International Conference on Supercomputing (ICS)*, Jun. 2016, pp. 26:1–26:13.

[37] A. Ros and A. Jimborean, "A hybrid static-dynamic classification for dual-consistency cache coherence," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Feb. 2016.