Building Expressive, Area-Efficient Coherence Directories

Lei Fang, Peng Liu, and Qi Hu
Department of ISEE
Zhejiang University
Hangzhou 310027, China
{lei_fang, liupeng, huqi_isee}@zju.edu.cn

Michael C. Huang
Department of ECE
University of Rochester
Rochester, NY 14627
michael.huang@rochester.edu

Guofan Jiang
IBM China Systems and Technology Lab
Shanghai 201203, China
jianggf@cn.ibm.com

***Abstract*—Mainstream chip multiprocessors already include a significant number of cores that make straightforward snooping-based cache coherence less appropriate. Further increase in core count will almost certainly require more sophisticated tracking of data sharing to minimize unnecessary messages and cache snooping. Directory-based coherence has been the standard solution for large-scale shared-memory multiprocessors and is a clear candidate for on-chip coherence maintenance. A vanilla directory design, however, suffers from inefficient use of storage to keep coherence metadata. The result is a high storage overhead for larger scales. Reducing this overhead leads to saving of resources that can be redeployed for other purposes.**

In this paper, we exploit familiar characteristics of coherence metadata, but with novel angles and propose two practical techniques to increase the expressiveness of directory entries, particularly for chip-multiprocessors. First, it is well known that the vast majority of cache lines have a small number of sharers. We exploit a related fact with a subtle but important difference: that a significant portion of directory entries only need to track one node. We can thus use a hybrid representation of sharers list for the whole set. Second, contiguous memory regions often share the same coherence characteristics and can be tracked by a single entry. We propose a multi-granular mechanism that does not rely on any profiling, compiler, or OS support to identify such regions. Moreover, it allows co-existence of line and region entries in the same locations, thus making regions more applicable. We show that both techniques improve the expressiveness of directory entries, and, when combined, can reduce directory storage by more than an order of magnitude with negligible loss of precision.

Keywords: Directory, Coherence, Scalability

I. Introduction

Technology scaling has steadily increased the number of cores in a mainstream chip-multiprocessor. Special-purpose large-scale chip-multiprocessors (CMPs) are also appearing in the marketplace [13], [21], [30]. Shared-memory programming interface is still a crucial element in productively exploiting the performance potential of these chip-multiprocessors. Consequently cache coherence will continue to be a key requirement of chip-multiprocessors. The increasing core count makes pure snooping protocols less appropriate. A directory-based approach will be increasingly seen as a serious candidate for on-chip coherence solution.

This work is supported in part by NSFC under grants 61028004 and 60873112 and by NSF under grants 1217662 and 0747324.

While directory-based coherence design has been studied extensively in the context of conventional multiprocessor design, chip-multiprocessors present subtle and yet important differences that call for new solutions or new twists. For example, conventional multiprocessors are built from commercial, off-the-shelf processor components that are fabricated with a focus towards personal systems. The directory logic is implemented outside the processor chip, whereas in a chip multiprocessor, the directory can closely interact with other on-chip logic. Also, in conventional multiprocessors, the cost of directories is only incurred when building a multiprocessor. In contrast, directory cost is incurred for every chip multiprocessor. Cost saving is thus more important.

In the most basic incarnation, a directory entry is allocated for every memory block [8], and each entry uses a full bit vector to track the list of sharers of that block. The overhead of a full bit vector is clearly significant in a system with many cores. For instance, a 64-core system with 32-byte cache lines, the overhead is 25%. Directory storage can be viewed as a 2D array, with the height being the number of entries and the width being the number of sharers to track. Reducing storage requires reducing one or both dimensions. And there are plenty of characteristics of access and sharing patterns that allow us to reduce the two dimensions: only a small portion of all memory blocks are cached at any time; most cache lines have a small number of sharers, etc. [9], [11], [12], [16], [28], [31]. Additionally, we can adjust parameters to reduce overhead such as using larger cache lines or coarser-grain sharing vectors. Regardless of the exact mechanism, reducing the storage size comes at a cost of loss of precision of coherence tracking, which leads to extra messages, invalidations, misses, and ultimately performance loss. However, the resources saved can be redeployed elsewhere to make up for the performance loss. In particular, other meta information such as access pattern, frequency, and affinity can help processor optimize data placement, storage allocation and so on [17], [18]. Chip multiprocessor presents brand new opportunities to provide holistic on-chip data management solutions. Providing expressive, area-efficient directory systems is a starting step.

One issue about the conventional directory mechanism is that it is a mechanical, access-pattern-agnostic approach in tracking coherence. Sharing patterns are tracked cache line by cache line even though there may be much more expressive means. For instance, private data have no other sharer than the owner. The coherence information of a whole region can be

described by a single metadata entry. Similarly, code segments and other (mostly) read-only data need not be tracked line by line either. Note that in practice, exploiting private data and code segments is not trivial, as these access patterns are not architected. Relying on help from external agents (e.g., compiler, programmer) brings its own set of issues. In this paper, we do not attempt to identify these patterns directly, but exploit the consequence of these patterns. First, (partly) because of prevalence of private data, many sharers lists can be represented by *single* pointers. Therefore we use hybrid representation of sharers list within a cache set: many single-pointers plus a few full-blown vectors. Second, because of regions of private or read-only data, we can use a single entry to capture the state of many cache lines simultaneously. Additionally, we allow exceptions within the region. This flexibility makes such regions more common than otherwise. Both types of savings can be achieved with simple, practical architectural support.

The rest of the paper is organized as follows. Section II gives the background and related work on directory optimizations. Section III presents our design in details. Section IV gives the evaluation result of our schemes and Section V concludes.

II. Background and Related work

Typically, all the cached copies of a memory block are invalidated on eviction of the associated directory entry. Increasing the size of a directory cache can reduce the frequency of evictions and cache miss rate but aggravates memory overhead and power consumption. As the number of processors increases, more directory entries are required to track a growing number of cache blocks. And the size of full bit vector grows linearly with the number of processors. As a result, the aggregate area of directory cache grows as the square of the number of processors, making them very expensive in large-scale systems [24], [32].

Compressing a full bit vector into a compact structure can reduce the size of a directory entry. Some information is lost during the compression and this imprecision would cause performance degradation. Limited pointers scheme [2] associates a few pointers to each directory entry to track a small set of sharers. Pointer overflow occurs when the number of sharers exceeds the number of pointers. LimitLESS directory [9] extends the directory pointer array into local memory by software on pointer overflow. Pointers are either linked to increase their number [23] or converted into a coarse vector [16]. Scalable coherence directory (SCD) [29] combines limited pointers and hierarchical directory [15], [33], [36]. Dynamic pointer allocation scheme [31] associates pointers dynamically based on the number of sharers. SLiD [11] combines limited pointer scheme and chained directory scheme [19]. Other than pointers, Tagless directory [38] uses bloom filter to encode the tags in each private cache. SPACE [39] encodes the sharing patterns and stores the code in each directory entry. SPATL [40] combines Tagless and SPACE. Trees can used to encode sharers [27] and a hybrid of limited pointers and tree scheme has also been proposed [10]. Segment directory scheme [12] chops the full bit vector into smaller pieces and stores only the non-zero pieces each with an identifying pointer so that the full bit vector can be reconstructed. Multilayer clustering [1], tristate [2], gray-tristate [26] and home [26] stores processor numbers in the sharing state.

In a shared memory system, memory blocks can be thought of as those that need coherence tracking (read-write shared by multiple nodes) and those that do not (private to one node or read-only). The latter is sometimes referred to as non-coherent [14]. Space can be saved when these "non-coherent" blocks are recognized in advanced and sharers tracking can be avoided, thus saving directory entries. This can be achieved with the help of page-level information with the aid of TLB and the operating system [14]. Another example is Spatiotemporal Coherence Tracking [3], which sets the first processor accessing a region as the owner. When a region is accessed by other processors, the data are tracked with the granularity of a block. Coarse-Grain Coherence Tracking [7] and RegionScout [25] remove unnecessary broadcast by tracking large regions at the expense of extra storage. RegionTracker [37] supports region tracking and management, which can be used as the building block for coarse-grain optimization. These region-based schemes are most effective when memory layout is such that data with the same access behavior (e.g., private, or read-only) are grouped together, separate from other types, and placed in aligned regions. In reality, without conscious layout optimization, different types of data may well be commingled and render these region-based schemes less effective. Indeed, it is found that "non-coherent blocks" make up about half of the space inside coherent pages [14]. In our analysis (with a broader set of applications), more than 60% of the lines in coherent pages are non-coherent. In our opinion, allowing exceptions is an important factor in making region-based tracking effective.

III. Hybrid Representation and Multi-Granular Tracking

Our goal is to reduce the memory overhead of directory cache for effectiveness and scalability. The total size of directory cache is the product of directory entry size and the number of entries. We target both in this paper. We first discuss the underlying baseline directory cache. We then discuss hybrid representation and multi-granular tracking that target reduction in entry size and entry number, respectively.

A. Baseline directory cache

We assume a tiled chip-multiprocessor baseline where each tile contains a core, private caches, and a slice of the globally shared cache as Fig. 1 shows. Data are mapped to their home L2 slice in a round-robin fashion based on their (physical) address. In the most straightforward implementation, the directory is directly attached to the data cache, where the sharers list is essentially part of the cache lines metadata. Alternatively, the directory is decoupled from the data cache. In this case, the number of directory entries need not match the number of cache lines. Indeed, a directory cache [16], [28] is one such example where the number of directory entries is significantly smaller than the number of cache lines. Even considering that the directory cache has to include a separate tag field, the result is often a net savings in total storage. A natural side effect of such decoupling is that inclusivity is only necessary between the directory entry and L1 cache lines. In other words, when an L2 cache line is evicted, there is no need to invalidate the

corresponding L1 cache lines. Only when the directory entry is evicted, the corresponding L1 cache lines are invalidated.

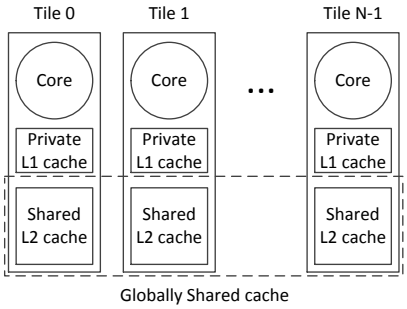

Fig. 1. System overview of a tiled chip-multiprocessor.

B. Hybrid representation

In terms of directory entry size, a full bit vector provides complete freedom to track all possible sharing patterns. But the size is large and grows linearly with the number of processors. Earlier studies have shown that a large number of lines are shared by only a few processors at any given time [34]. A handful of compact structures are proposed to replace the full bit vector based on this knowledge. Compact structures cannot track as many sharers as vectors and may cause *pointer* overflow [22]. Schemes are proposed to address the issue of pointer overflow at a cost of extra traffic and performance degradation [2], [16].

We exploit sharing pattern from a different angle: a significant fraction of the cache lines have only one sharer/owner at a time. If we consider the storage of directory entries *in a set as a whole*, we can use hybrid representation with a few entries capable of tracking multiple sharers and the rest tracking just single sharers. By pooling resources together, we need not attempt to make a single entry versatile and self-sufficient. The cost is that when a single-sharer-tracking entry proves to be insufficient, we need to perform intra-set swapping. Fortunately, such swapping is infrequent in the steady state and only involves the directory cache entries and not data cache lines.

To quantify the potential, we track the number of multi-sharer entries in any set of an 8-way associative directory cache. As shown in Fig. 2, at any given time, in a vast majority (on average 90%) of the cases, all 8 entries are tracking cache lines with a single sharer. In more than 99% of the cases, there are no more than 2 entries tracking multiple sharers.

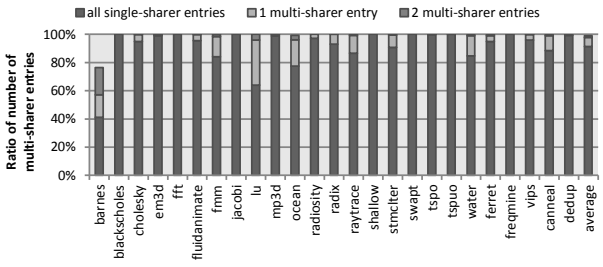

Fig. 2. The ratio of number of multi-sharer entries for an 8-way associative directory cache in a 16-way CMP.

In terms of implementation, we use a pointer for entries of single sharer (which we call *pointer entries*) and full vectors for multi-sharers (*vector entries*) as illustrated in Fig. 3. In this figure, we show the directory cache set of an 8-way associative directory cache in a 16-way CMP. Each set uses 2 vector entries and 6 pointer entries. The average size of a directory entry depends on the fraction of vector entries. Assuming a P-way CMP, with an A-way associative directory cache that has V vector entries per set, the average size of a directory entry is thus $\frac{V*P+(A-V)*log_2 P}{A} + TagSize$.

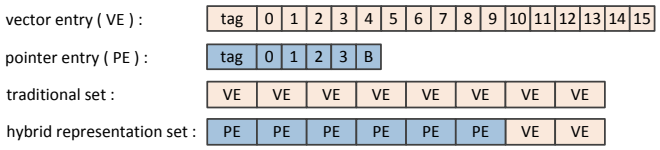

Fig. 3. Storage for the proposed hybrid representation compared to traditional representation in a 16-way CMP with 8-way associative directory sets.

Since a pointer entry can only track one sharer, when another processor requests to read the line, the pointer entry can no longer track both sharers. We handle the overflow by swapping its content with a vector entry (in the same set). If there is an unused vector entry, or a vector entry tracking just one sharer, such a swap is straightforward as illustrated in Fig. 4. If all vector entries are multi-sharer entries, the least recently used (LRU) vector entry is converted into a single-sharer vector entry to allow the swap.

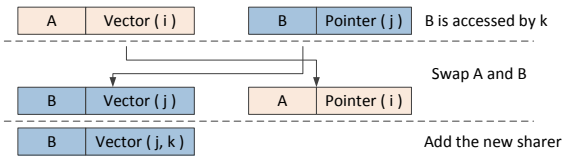

Fig. 4. Swap between pointer entry and vector entry.

To minimize imprecision of this conversion, we use a threshold (Fig. 5) to "round" the sharers list either down to a randomly selected current sharer (which we call down conversion), or up to all nodes (up conversion). In the former case, other sharers are invalidated. In the latter case, a single *broadcast* bit ("B" in Fig. 3) is set to indicate that all nodes potentially share the cache line.

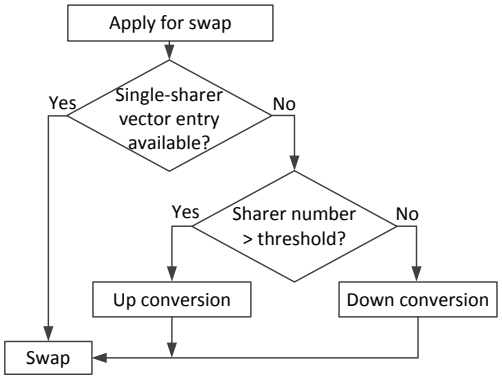

Fig. 5. Adaptive conversion.

TABLE I. OPERATIONS OF HYBRID REPRESENTATION.

entry type	request	broadcast bit	operation
vector	—	—	Conventional.
pointer	write	0	Invalidation message is sent to the exclusive owner.
pointer	write	1	Invalidation message is sent to all processors except the writer.
pointer	read	0	Swap occurs. Then the coherence operation is conventional.
pointer	read	1	The copy is obtained from the shared data cache, private cache of the sharer or main memory.
pointer	eviction	0	Invalidation message is sent to the exclusive owner.
pointer	eviction	1	Invalidation message is sent to all processors.

Note that the swap is between two (metadata) entries in the directory cache, not the data cache lines. When a swap occurs, the directory cache needs to be read and written to in consecutive cycles, increasing its occupancy. Such swaps are very rare: only about 1 in every 1000 transactions involves a swap (experimental setup will be discussed later in Sec. IV, and the occupancy is faithfully modeled).

Finally, the operations of hybrid representation are summed up in Table I.

C. Multi-granular tracking

In addition to reducing the size of each entry, reducing the number of entries is another factor of reducing the overall footprint of directory cache. One approach is to exclude regions that contain only private data. These regions can be considered to be non-coherent regions [14]. The challenge of such an approach is the determination of private data as it is not semantically guaranteed by the architecture. Thus if one line in the region is detected to be coherent and thus require coherence tracking, the entire region ceases to be treated as non-coherent [14].

We take a different approach in reducing the number of entries. First, we employ region entries to capture a region of data with similar access patterns such as private and read-only. Second, a region entry does not preclude regular line-tracking entries to describe individual special cases inside an otherwise homogeneous region. In other words, the region entries are more of an optimization that describe common patterns than a special element to exclude the special case of non-coherent data.

1) Identifying regions: Perfectly identifying the right boundaries of regions is challenging as these regions come with different sizes and access patterns and their behavior changes dynamically. For simplicity of implementation, we use fixed-size, aligned regions and a simple online algorithm as follows to allocate region entries. This approach requires no pre-characterization from the compiler, off-line profiling, or OS support to identify or demarcate regions.

- Upon the first access to a line without an existing directory entry, we always start with a region entry. The implicit, optimistic assumption is that the whole region has similar sharing patterns and hence requires no fine-grain, line-by-line tracking.
- From that point on, any additional nodes reading from any cache line in that region will be added to the region's sharers list. In other words, a region entry is simply one where the coherence tracking unit is a region.
- When a write request arrives at the directory and the requester is the only sharer so far, we treat the region as private and change the region entry's state to "modified". If on the other hand the write requester is not the only sharer, we treat this line as a special case and start a line entry to track it. All sharers of the region will be sent an invalidation message just for the line.
- Similarly, if a region in modified state receives a read request from a node other than the owner, a line entry will be allocated to track the line under request.

2) Sizing of regions: Since we use fixed-size regions for simplicity, the design decision becomes their size. Clearly, a larger size creates a more compact tracking when the region is homogeneous, but can lead to more space waste when the actual size of a region with homogeneous sharing pattern is smaller. This can be seen in the example shown in Fig. 6. In this example, the chosen region size (eight lines) is bigger than the actual size of the two regions of homogeneous sharing patterns. As a result, the number of entries uses is more than when the region size is four.

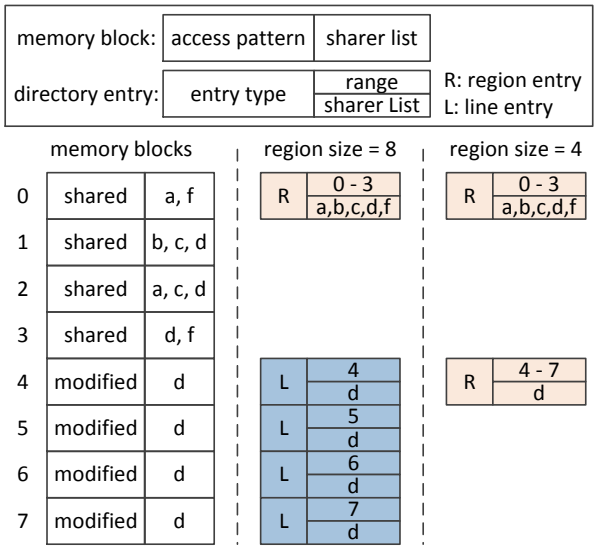

Fig. 6. An example of mismatch between the actual and chosen region size.

3) Implementation issues: The operations of multi-granular tracking scheme are summarized in Table II. Here are a few implementation issues worth noting.

To support multi-granular tracking, a grain size bit is used to distinguish between a region entry and a line entry. The

TABLE II. OPERATIONS OF MULTI-GRANULAR TRACKING SCHEME.

entry type	request	sharer list	operation
line	—	—	Conventional.
region	write	equal writer	The region is accessed by one processor. No line entry is allocated.
region	write	other sharers	The line is recognized to be coherent. A line entry is allocated for it.
region	read	—	The line is obtained from the shared data cache or memory. the requester is added to the sharer list.
region	write back	—	The requester may cache other lines of the region and cannot be removed from the sharer list.
region	eviction	—	Except exceptional lines, copies of the lines in the region are invalidated in the sharers' private cache.

natural indexes for the two types of entries are different, calling for two separate accesses to the directory. This need not be the case. We can shift the index for line entries to align with that of their corresponding region entry as shown in Fig. 7. This way, both line and region entries of any line will reside in the same set. When both are found, the line entry takes priority (Fig. 8). In that case, the region entry is not counted as a hit in the (LRU) replacement circuitry. Consequently, a region that has a diverse set of sharing patterns among its lines will have many individual line entries and the region entry will quickly fall into disuse and get recycled. In a sense, the system is thus automatically choosing the right type of entries to use depending on the circumstance. Mapping consecutive lines into the same set may cause increased conflict. Intuitively, given reasonable associativity for the directory cache, the magnitude of the problem should be small. Besides, region entries naturally eliminate many individual line entries that would otherwise exist, further reducing conflict pressure. Indeed, in our simulations we found that mapping both types of entries to the same set is slightly better than using their natural indexes.

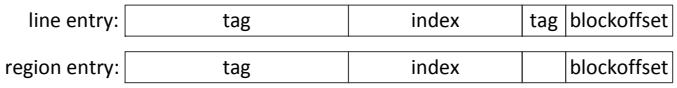

Fig. 7. Address fields of region and line entries.

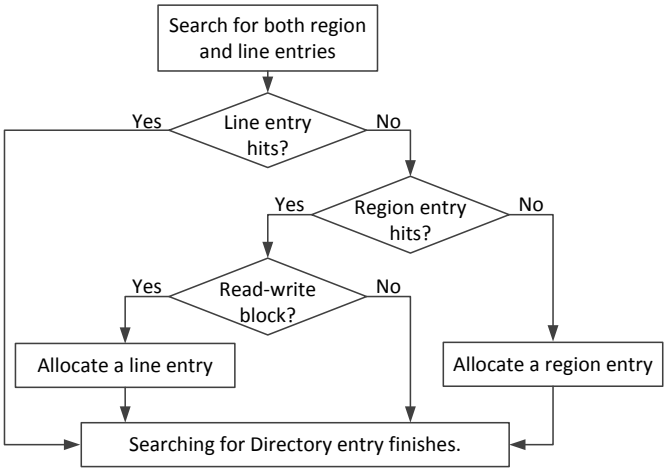

Fig. 8. Search of directory entry with multi-granular scheme.

When an L1 cache miss is serviced, we may find a region entry to the line and at the same time no data in the L2 cache since we do not maintain L2-L1 inclusivity. In this case, if the region entry is in shared state, we do not have a high confidence any node on the sharers list actually has the line: Intuitively, the L2 cache has a much larger capacity and is thus unlikely to evict a line still in some L1 cache. (We found through simulations that in those cases there is only a small chance – 2.4% on average – that *some* L1 cache actually has the data on-chip.) For simplicity, we go off-chip for the data. Of course, if the region entry is in modified state, we will forward the request to the owner node first. We also check the sharers first if the line is tracked by a line entry.

Upon the eviction of a region entry from the directory cache, we need to inform all the sharers to invalidate those lines in the region that are not tracked by line entries. In this paper, for simplicity, we infer from the coherence state at the L1 level which lines must have line entries already and therefore need not be invalidated. We compare the coherence state of the line in L1 cache to that of the region to be invalidated. If they are incompatible, then the line must have been tracked by a line entry. For example, if the region is in modified state and the line is in shared state. Clearly, these lines are only a subset of those that can be exempted from invalidation. We found that on average 85% of lines that can be exempted from region invalidation can be inferred this way.

D. Combinations

The two techniques described can be combined together or with other space-saving techniques in a rather straightforward manner. For instance, in multi-granular tracking the sharers list can be implemented in either pointer or vector format as in hybrid representation. As another example, the vector entries in hybrid representation can also be replaced by a few pointers as in [9]. In some cases, applying multiple techniques working on the same source of storage inefficiency quickly reaches diminishing returns. The techniques can be contrasted based on cost benefit ratio. Section IV contains some quantitative analysis.

IV. EXPERIMENTAL ANALYSIS

We analyze in detail hybrid representation (Sec. IV-B) and multi-granular tracking (Sec. IV-C) in isolation as well as in conjunction (Sec. IV-D).

A. Experimental Setup

Our quantitative analyses are performed using an extensively modified version of SimpleScalar [6]. We use wattch [5] and orion 2.0 [20] for power analysis. The execution-driven simulator models in great detail the cache coherence substrate using a MESI protocol, the processor microarchitecture, the communication substrate and the energy consumption of 16-way and 64-way CMP. The system parameters are summarized in Table III.

TABLE III. SYSTEM PARAMETERS.

Processor core	
Fetch/Decode/Commit	4 / 4 / 4
ROB	64
Issue Q/Reg. (int,fp)	(32, 32) / (64, 64)
LSQ(LQ,SQ)	32 (16,16) 2 search ports
Branch predictor	Bimodal + Gshare
- Gshare	8K entries, 13 bit history
- Bimodal/Meta/BTB	4K/8K/4K (4-way) entries
Br. mispred. penalty	at least 7 cycles
Memory hierarchy	
L1 D cache (private)	16KB, 2-way, 64B line, 2 cycles, 2 ports
L1 I cache (private)	32KB, 2-way, 64B line, 2 cycles
L2 cache (shared)	256KB slice, 8-way, 64B line, 15 cycles, 2 ports
Directory cache(shared)	128 sets slice, 8-way, 15 cycles, 2ports
Intra-node fabric delay	3 cycles
Main memory	at least 250 cycles, 8 memory controllers
Network packets	Flit size: 72-bits data packet: 5 flits, meta packet: 1 flit
NoC interconnect	4 VCs; 2-cycle router; buffer: 5x12 flits wire delay: 1 cycle per hop

TABLE IV. BENCHMARKS.

Splash-2	barnes, fft, fmm, lu, ocean, radiosity, radix, raytrace, water-spatial
Parsec	blackscholes, canneal, dedup, ferret,freqmine, fluidanimate,streamcluster, swaptions, vips
Others	em3d, jacobi, mp3d, shallow, tsp

We perform the evaluation with a suite of parallel applications including SPLASH2 [35] benchmark suite, PARSEC [4], a program to solve electromagnetic problem in 3 dimensions (em3d), a program to iteratively solve partial differential equations (jacobi), a 3-dimensional particle simulator (mp3d), a shallow water benchmark from the National Center for Atmospheric Research to solve differential equations on a two-dimensional grid for weather prediction (shallow), and a branch-and-bound based implementation of the non-polynomial traveling salesman problem (tsp). The applications are summarized in Table IV. The cache sizes are smaller than typical values to compensate for the reduced data sets used by many applications. With these sizes, the average L1 miss rate is 5.6%.

The directory cache of baseline is configured to be 128-set and 8-way associative per slice. This configuration is chosen since it performs close to a system with full directory. Further reducing the size of the directory cache will cause serious performance degradation. At this configuration, assuming a 40-bit physical address and a 16-way CMP, the directory storage overhead (including the extra tag) comes to about 11 bits per L2 cache line, or about 2.0%, compared to 3.6% for an in-L2 directory. In a 64-way CMP, those overheads would rise to 4.2% for directory cache and 12.6% for the in-L2 directory.

B. Hybrid Representation

In the following, we use a 16-way CMP as baseline for analysis (Sec. IV-B1); then show that the design's impact on execution is not sensitive to configuration parameters (Sec. IV-B2); and finally show that the technique saves more space with far less performance impact than related designs (Sec. IV-B3). The baseline conventional directory cache is noted as $DC(set, associativity)$. In particular, the baseline configuration is $DC(2048, 8)$. The proposed hybrid representation will be noted as $DC(set, v/associativity)$ where v means the number of vector entries in a set.

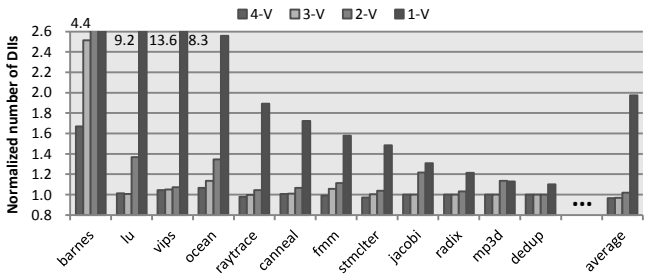

Fig. 9. The number of DIIs in $DC(2048, v/8)$ normalized to that in $DC(2048, 8)$.

1) Effects on the baseline system: As discussed earlier (Fig. 2), in a typical set, only a few entries need vectors to track sharers. When the number of vector entries is limited, there is an increased chance of victimizing a vector entry. The result can be a down conversion that invalidates all but one current sharer. Such invalidations have nothing to do with program's true communication and are purely due to directory imprecision. We call these directory-induced invalidations (DIIs). Note that without hybrid representation, a directory cache already creates DIIs.

Fig. 9 shows the relative number of DIIs for different applications under hybrid representation with one to four vector entries. For clarity, we sort the applications based on decreasing values and only show the several applications where these values are large. The rest of the 25 applications have results very close to 1.

As we can see, having a single vector entry is a bit too extreme and for a few applications can dramatically increase DIIs. It is worth noting that normalizing the number of DIIs highlights the imprecision introduced by the hybrid representation. Overall, DIIs represent a small portion of the total number of invalidations (15.1% in the baseline). So even a several-fold increase may not cause significant increase in overall invalidation number. This can be seen from Fig. 10, which shows the L1 cache miss rate normalized to that in baseline.

Take application *lu* for example. Having just one vector entry increases DIIs by 13.6X. But this dramatic increase in DIIs only results in 46% increase in miss rate. On average, even having just one vector entry only increases miss rate by 5%. With two vector entries per set, the increase in the number of misses (and off-chip traffic) is 0.4% on average, and 2.3% in the worst case, which is almost negligible.

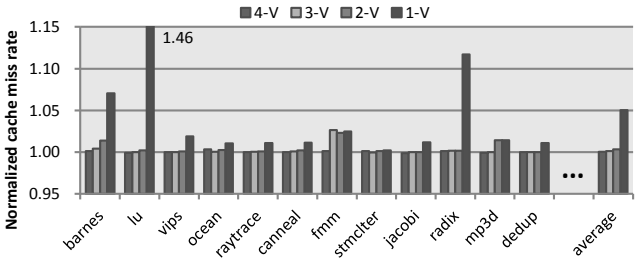

Fig. 10. L1 cache miss rate of $DC(2048, v/8)$ normalized to $DC(2048, 8)$.

Recall that victimizing a vector entry that tracks more than one sharer results in either a down or an up conversion. While down conversions increase the number of DIIs, up conversions cause imprecision that increases the number of unnecessary invalidation messages later on. In $DC(2048, 2/8)$, the increase in all invalidation messages averages about 0.7%. Of course, both type of conversions lead to performance loss and energy overhead. Finally, the swapping between a vector and a pointer entry increases the occupancy of directory cache, potentially delaying other requests. According to the experimental result, the port usage increases by 1.9% on average for $DC(2048, 2/8)$. All these factors impact the overall execution time and energy. These statistics for $DC(2048, 2/8)$ are shown in Fig. 11.

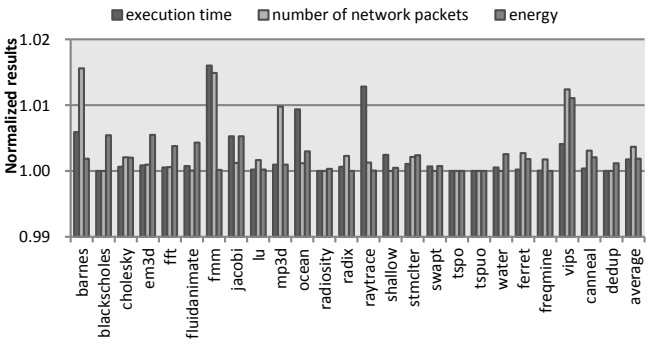

Fig. 11. Execution time, number of network packets and energy of $DC(2048, 2/8)$ normalized to $DC(2048, 8)$.

As the figure shows, when using a quarter of the entries as vector entries, the execution time, number of network packets, and energy consumption of $DC(2048, 2/8)$ increase by less than a half of a percentage point (specifically 0.2%, 0.4% and 0.2%, respectively). Since we use a quarter of the entries as vector entries, the asymptote of area savings is 4x. The actual savings in storage depends on a number of parameters. For the 16-way CMP, $DC(2048, 2/8)$ reduces the total directory storage by about 1.31x (24% reduction). For 64-way CMP the savings becomes 2x with similarly negligible impact on execution (details later). In contrast, if the directory cache is shrunk by a quarter (in the number of sets), the performance degradation is a much more pronounced 10%.

2) Sensitivity analysis: Hybrid representation exploits the fact that most lines in a set have a single sharer at any moment. In roughly 99% of the cases, less than a quarter of the lines in a set have more than one sharer. This statistic is largely unchanged with increase of directory cache associativity (as shown in Table V) or the number of nodes in a CMP.

TABLE V. THE RATIO OF NUMBER OF MULTI-SHARER ENTRIES.

Number of multi-sharer entries	Cumulative ratio	
	$DC(2048, 8)$	$DC(1024, 16)$
0	91.4%	87.0%
≤ 1	97.7%	96.0%
≤ 2	**99.0%**	97.8%
≤ 3	99.7%	98.3%
≤ 4	99.9%	**99.2%**

In a 64-way CMP, keeping each tile the same, the execution impact is essentially the same as in a 16-way CMP. The increase of execution time, number of packets, and energy consumption are 0.6% or less on average and 2.5% for the worst case.

3) Comparison with related schemes: We compare hybrid representation (HR) with other compacting schemes, including limited pointer (LP) [2], coarse vector (CV) [16] and scalable coherence directory (SCD) [29]. Table VI shows the relative area savings and performance degradation of these schemes compared to the full bit vector scheme in a 64-way CMP.

TABLE VI. STORAGE OF DIRECTORY CACHE AND PERFORMANCE OF DIFFERENT COMPACTING SCHEMES.

	area reduction	increment of network packets(%)	increment of execution time(%)
HR	2X	0.4	0.6
LP	1.8X	8.0	8.5
LP+HR	2.5X	8.1	8.8
CV	1.8X	2.7	2.4
CV+HR	2.5X	2.8	2.5
SCD	2.1X	9.3	10.2
SCD+HR	2.6X	9.6	10.7

As the table shows, hybrid representation outperforms other schemes and causes negligible degradation in both network traffic and execution time. Hybrid representation is able to track the sharers more precisely with the same storage for its dynamic allocation and adaptive conversion schemes. We also evaluate the combination of hybrid representation and other schemes. As the table shows, hybrid representation is quite orthogonal to other schemes and is able to reduce the storage further with little performance degradation.

C. Multi-Granular Tracking

In the following, we first analyze the appropriate size for region in multi-granular tracking in Sec. IV-C1, then show the overall effect in Sec. IV-C2, and finally compare to an alternative design [14] in Sec. IV-C3. We use the notation $DC(set, associativity, regionsize)$ to represent a specific configuration. Again, we start from the baseline of $DC(2048, 8)$.

1) Region size: Region size is an important parameter in our design. Increasing the size of a region reduces the cost to track a large, homogeneous region but increases the chance a region is no longer homogeneous. We start with a directory cache 8x smaller than baseline and compare a number of region sizes from 8 to 64 cache lines and the average results of major statistics from all applications are summarized in Fig. 12.

When the region size increases from 8 to 16, the overall effective tracking capability of the directory is on average better than the baseline directory 8 times larger and sometimes generates significantly less DIIs. As a result, fewer invalidations and acknowledgement packets are sent, which leads to a noticeable (10%) average reduction in overall network packets. Note that some DIIs merely time-shift a future eviction as a present invalidation. Thus the impact on cache miss rate is smaller. Note that there is unevenness in application behavior

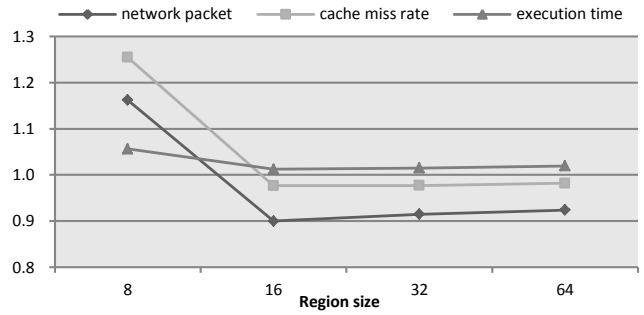

Fig. 12. The impact of region size on the number of L1 data cache misses, total network packets, and the execution time. For clarity, all results are first normalized to baseline $DC(2048, 8)$, then averaged over all applications. In all cases, lower is better.

and the performance of $DC(256, 8, 16)$ is slightly worse (by 1.2%) than $DC(2048, 8)$ on average despite having a slightly lower average cache miss rate. When the region size further increases, the intra-region disparity starts to increase and negates the benefit of larger regions. In our setup, the best performing region size is 16, although both 32 and 64 produce rather similar results in terms of performance.

2) Effects of multi-granular tracking: Region entries magnify the descriptive power of a directory entry, thereby requiring less storage. Depending on the application, the average magnification factor ranges from 4 to 11, with a suite-wide average of about 7. In other words, a region entry replaces an average of 7 line entries, albeit with a slight loss of precision. Specifically, compared to baseline $DC(2048, 8)$, the number of DIIs of $DC(256, 8, 16)$ is 10% lower on average for an 8x reduction in directory size, though with significant variation among applications. In contrast, a mere 2x reduction in directory size (to $DC(1024, 8)$) results in an average of 10x increase in DIIs and consequently a 41% increase in L1 misses. An 8x reduction in directory size (to $DC(256, 8)$) would make those increases balloon to 25x and 89%, respectively.

We evaluate the performance impact of directory cache size with and without multi-granular tracking. Fig. 13 shows the relative performance of different configurations normalized to an ideal directory cache system. The associativity of directory cache is fixed to be 8 and the number of sets is changed from 4096 to 128.

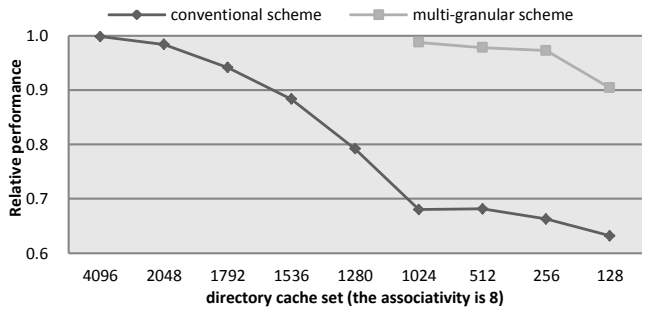

Fig. 13. Performance vs directory cache size with or without multi-granular tracking. Note that between 2048 and 1024, we added a few configurations not at power-of-2 sizes to better capture the marginal benefit of entries around 2048.

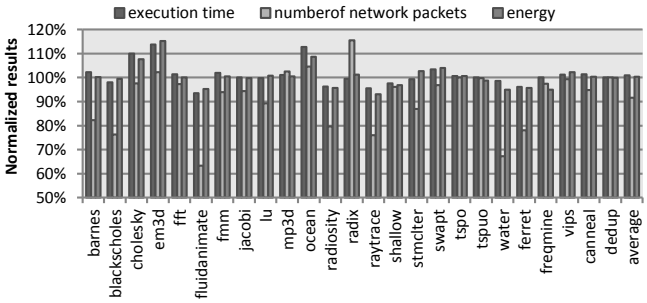

Fig. 14. Execution statistics of $DC(256, 8, 16)$ normalized to $DC(2048, 8)$.

As the figure shows, without multi-granular tracking, the degradation in performance is negligible when the size is reduced to 2048 (sets), but noticeable even with a small reduction from 2048 (which is why we set the baseline to this configuration). Indeed, halving the entries to 1024 would reduce performance by more than 30%. In stark contrast, when employing multi-granular tracking, with only 256 sets, the performance is only 1.2% lower than $DC(2048, 8)$ and 3% lower than $DC(4096, 8)$. Note that there is non-trivial variation from application to application. However, even in the worst case, the performance impact is 13% as shown in Fig. 14. We have also evaluated multi-granular tracking in a 64-way CMP. With an average increase of execution time, number of packets, and energy consumption of less than 1.0%, the observations are essentially the same as in the 16-way CMP.

3) Comparison to coarse grain coherence tracking: An example of related coarse grain coherence tracking schemes is the use of page translation information in bypassing private pages as non-coherent regions [14]. There are a number of differences between the two techniques. First, our system is completely transparent to other subsystems and does not require modifications to TLB control. Second, regions can be in any coherence state and are not limited to just private pages. Third, our regions allow exceptions, making regions more applicable. The net effect of these differences is that given the same storage capacity for entries, our directory is able to capture the coherence states with less DIIs and cache misses. As Fig. 15 shows, using a 256-set directory cache with the page-based technique [14], the cache miss rate increases by 15% on average over the baseline $DC(2048, 8)$. In contrast, with our multi-granular scheme, the increment is less than 0.5%.

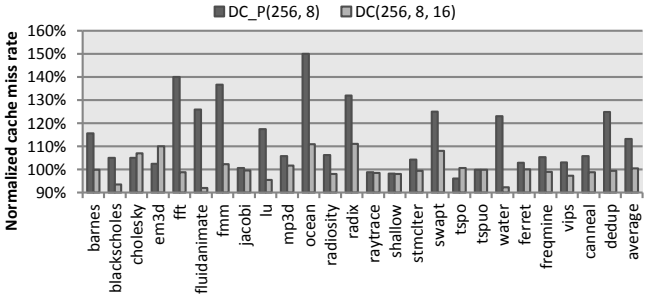

Fig. 15. Cache miss rate normalized to $DC(2048, 8)$. DC_P indicates directory cache with page-based coarse-grain technique [14].

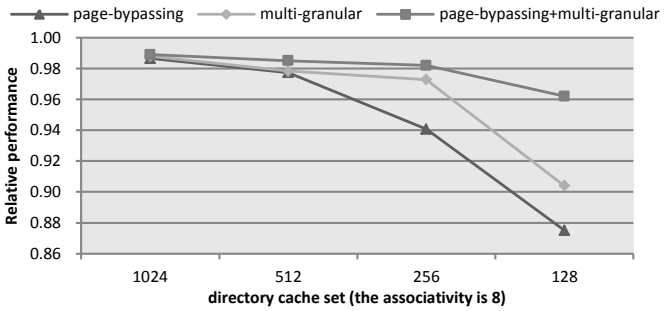

Fig. 16. Relative performance of multi-granular tracking, page-based bypassing, and their combination as a function of directory cache size.

Even though multi-granular tracking is on average more effective, page-based technique has an advantage that bypassed pages occupy no entries in the directory at all. It is conceivable that page-based bypassing can be implemented on top of multi-granular tracking to allow further reduction of directory expenditure. Fig. 16 shows the relative performance of these two approaches acting independently and when combined. Indeed, when the two are combined, the resulting scheme shows smaller performance loss than either technique alone, and extends the range of acceptable performance to even smaller configurations of directory cache.

D. Multi-granular tracking and hybrid representation

To see if the two schemes would affect each other, we implement multi-granular tracking and hybrid representation together in the 16-way CMP. As cache lines are consolidated into region entries, the composition of the entries in a set changes. Intuitively, many single-sharer entries are private blocks. They are particularly amenable to region-based tracking and thus single-sharer entries see more reduction than multi-sharer entries. Furthermore, the sharers list of the region entry is the union of the sharers of the constituent cache lines, increasing the number of multi-sharer entries.

Fig. 17 shows the breakdown of entries in a set based on the number of multi-sharer entries for $DC(256, 8, 16)$. As we can see, only 86% of the sets have no more than 2 multi-sharer entries. This compares to 99% of sets in the baseline (see Table V). Apparently, more vector entries are needed to accommodate the increased ratio of multi-sharer entries. According to Fig. 17, having 4 multi-sharers entries

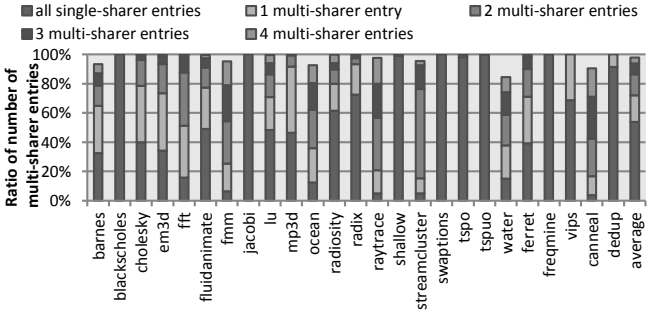

Fig. 17. The ratio of numbers of multi-sharer entries in a set for $DC(256, 8, 16)$.

in a set will cover 98% of the cases, similar to what 2 such entries cover without region-based scheme. The real performance impact of increased multi-sharer entries, however, is barely measurable. When enabling hybrid representation on top of $DC(256, 8, 16)$ and using 2 vector entries per set, the performance is only around 0.1% lower.

To sum up the overall effect, when the directory configuration changes from the knee-of-the-curve baseline directory cache $DC(2048, 8)$ to $DC(256, 2/8, 16)$ which has 10x smaller storage requirement in 16-way CMP (or 16x smaller in 64-way CMP), the average performance degradation is 1.2%.

V. Conclusion

We have proposed an expressive, area-efficient directory cache in this paper. Two schemes of hybrid representation and multi-granular tracking are used to reduce directory entry size and directory entry number respectively. In a shared memory system, a significant fraction of cache lines are owned by one processor. Hybrid representation uses a combination of vector and pointer entries in each directory cache set. The pointer or vector entry is allocated based on the number of sharers. The result is a reduction of directory entry size with little performance degradation. Multi-granular tracking allows a region of cache lines with similar coherence characteristics to be tracked by a single directory entry, thereby reducing the total number of entries needed. By allowing the co-existence of line and region entries in the same locations, we make the region entries more applicable and allow the system to automatically adapt to the right type of entry simply via normal eviction in the directory cache. Moreover, the use of region does not rely on any profiling, compiler, or OS support.

We evaluate 16-way and 64-way CMP systems with a detailed execution-driven simulator across a wide range of parallel applications. According to the experimental results, only a quarter of the sharers lists need to use a vector, making the asymptote of area savings of hybrid representation 4x. In 16-way and 64-way CMP, the actual area saving is 1.31x and 2x, respectively. The resulting degradation in runtime, network traffic, and energy consumption are essentially negligible. Multi-granular tracking is able to reduce the number of directory entries needed by 8x with a 1.2% performance degradation. When the two simple schemes are combined (reaping multiplicative savings benefits), the directory becomes much more expressive and area-efficient: Space expenditure is reduced by more than an order of magnitude (to about 0.2% of the L2 cache) while the loss of precision costs a small performance penalty of 1.2%. Such space efficiency can allow better scalability of directory without resorting to more drastic design overhauls. Also, some of the space saved can be redeployed to track access patterns and offer a more intelligent support of on-chip communication.

References

[1] M. Acacio, J. Gonzalez, J. Garcia, and J. Duato, "A New Scalable Directory Architecture for Large-Scale Multiprocessors," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, Jan. 2001, pp. 97–106.

[2] A. Agarwal, R. Simoni, and J. Hennessy, "An Evaluation of Directory Schemes for Cache Coherence," in *Proceedings of the International Symposium on Computer Architecture*, May–Jun. 1988, pp. 280–289.

[3] M. Alisafaee, "Spatiotemporal Coherence Tracking," in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2012, pp. 341–350.

[4] C. Bienia, S. Kumar, J. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, Sep. 2008, pp. 72–81.

[5] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 2000, pp. 83–94.

[6] D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2.0," Computer Sciences Department, University of Wisconsin-Madison, Technical Report 1342, Jun. 1997.

[7] J. Cantin, M. Lipasti, and J. Smith, "Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 2005, pp. 246–257.

[8] L. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, vol. C-27, no. 12, pp. 1112–1118, Dec. 1978.

[9] D. Chaiken, J. Kubiatowicz, and A. Agarwal, "LimitLESS Directories: A Scalable Cache Coherence Scheme," in *Proceedings of the International Conference on Arch. Support for Prog. Lang. and Operating Systems*, Apr. 1991, pp. 224–234.

[10] Y. Chang and L. Bhuyan, "An Efficient Hybrid Cache Coherence Protocol for Shared Memory Multiprocessors," in *Proceedings of the International Conference on Parallel Processing*, Aug. 1996, pp. 172–179.

[11] G. Chen, "SLiD–A Cost-Effective and Scalable Limited-Directory Scheme for Cache Coherence," in *Proceedings of Parallel Arch. and Lang. Europe*, Jun. 1993, pp. 341–352.

[12] J. Choi and K. Park, "Segment Directory Enhancing the Limited Directory Cache Coherence Schemes," in *Proceedings of the International Parallel and Distributed Processing Symposium*, Apr. 1999, pp. 258–267.

[13] 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, 2010.

[14] B. Cuesta, A. Ros, M. Gomez, A. Robles, and J. Duato, "Increasing the Effectiveness of Directory Caches by Deactivating Coherence for Private Memory Blocks," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 2011, pp. 93–104.

[15] S. Guo, H. Wang, Y. Xue, D. Li, and D. Wang, "Hierarchical Cache Directory for CMP," *Journal of Computer Science and Technology*, vol. 25, no. 2, pp. 246–256, 2010.

[16] A. Gupta, W. Weber, and T. Mowry, "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," in *Proceedings of the International Conference on Parallel Processing*, Aug. 1990, pp. 312–321.

[17] H. Hossain, S. Dwarkadas, and M. Huang, "Improving Support for Locality and Fine-Grain Sharing in Chip Multiprocessors," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, Sep. 2008, pp. 155–165.

[18] H. Hossain, S. Dwarkadas, and M. Huang, "DDCache: Decoupled and Delegable Cache Data and Metadata," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, Sep. 2009, pp. 227–236.

[19] D. James, A. Laundrie, S. Gjessing, and G. Sohi, "Distributed-Directory Scheme: Scalable Coherent Interface," *IEEE Computer*, vol. 23, no. 6, pp. 74–77, 1990.

[20] A. Kahng, B. Li, L. Peh, and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," in *Proceedings of the Design, Automation and Test in Europe*, Mar. 2009, pp. 423–428.

[21] R. Kalla, B. Sinharoy, W. Starke, and M. Floyd, "Power7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, no. 2, pp. 7–15, 2010.

[22] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 1997, pp. 241–251.

[23] Y. Maa, D. Pradhan, and D. Thiebaut, "Two Economical Directory Schemes for Large-Scale Cache Coherent Multiprocessors," *ACM SIGARCH Computer Architecture News*, vol. 19, no. 5, pp. 10–18, 1991.

[24] M. Marty and M. Hill, "Virtual Hierarchies to Support Server Consolidation," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 2007, pp. 46–56.

[25] A. Moshovos, "RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 2005, pp. 234–245.

[26] S. Mukherjee and M. Hill, "An Evaluation of Directory Protocols for Medium-Scale Shared-Memory Multiprocessors," in *Proceedings of the International Conference on Supercomputing*, Jul. 1994, pp. 64–74.

[27] H. Nilsson and P. Stenstrom, "The Scalable Tree Protocol-A Cache Coherence Approach for Large-Scale Multiprocessors," in *Proceedings of the International Parallel and Distributed Processing Symposium*, Apr. 1992, pp. 498–506.

[28] B. O'Krafka and A. Newton, "An Empirical Evaluation of Two Memory-Efficient Directory Methods," in *Proceedings of the International Symposium on Computer Architecture*, May 1990, pp. 138–147.

[29] D. Sanchez and C. Kozyrakis, "SCD: A Scalable Coherence Directory with Flexible Sharer Set Encoding," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, Feb. 2012, pp. 1–12.

[30] M. Shah, J. Barren, J. Brooks, R. Golla, G. Grohoski, N. Gura, R. Hetherington, P. Jordan, M. Luttrell, and C. Olson, "UltraSPARC T2: A Highly-Treaded, Power-Efficient, SPARC SOC," in *Proceedings of the IEEE Asian Solid-State Circuits Conference*, Nov. 2007, pp. 22–25.

[31] R. Simoni, "Cache Coherence Directories for Scalable Multiprocessors," Ph.D. dissertation, Department of Electrical Engineering, University of Stanford, 1992.

[32] R. Singhal, "Inside Intel Next Generation Nehalem Microarchitecture," in *Hot Chips*, Aug. 2008.

[33] D. Wallach, "PHD: A Hierarchical Cache Coherent Protocol," Ph.D. dissertation, Department of Electrical Engineering and Computer Sciences, MIT, Sep. 1992.

[34] W. Weber and A. Gupta, "Analysis of Cache Invalidation Patterns in Multiprocessors," in *Proceedings of the International Conference on Arch. Support for Prog. Lang. and Operating Systems*, Apr. 1989, pp. 243–256.

[35] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of the International Symposium on Computer Architecture*, Jun. 1995, pp. 24–36.

[36] Q. Yang, G. Thangadurai, and L. Bhuyan, "Design of an Adaptive Cache Coherence Protocol for Large Scale Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 3, pp. 281–293, May 1992.

[37] J. Zebchuk, E. Safi, and A. Moshovos, "A Framework for Coarse-Grain Optimizations in the On-Chip Memory Hierarchy," in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2007, pp. 314–327.

[38] J. Zebchuk, V. Srinivasan, M. Qureshi, and A. Moshovos, "A Tagless Coherence Directory," in *Proceedings of the International Symposium on Microarchitecture*, Dec. 2009, pp. 423–434.

[39] H. Zhao, A. Shriraman, and S. Dwarkadas, "SPACE: Sharing Pattern-based Directory Coherence for Multicore Scalability," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, Sep. 2010, pp. 135–146.

[40] H. Zhao, A. Shriraman, S. Dwarkadas, and V. Srinivasan, "SPATL: Honey, I Shrunk the Coherence Directory," in *Proceedings of the International Conference on Parallel Architecture and Compilation Techniques*, Oct. 2011, pp. 33–44.