CMP Directory Coherence: One Granularity Does Not Fit All

Arkaprava Basu[#] Bradford M. Beckmann[*] Mark D. Hill[#] Steven K. Reinhardt[*]

[#] University of Wisconsin-Madison [*] AMD Research

Abstract

To support legacy software, large CMPs often provide cache coherence via an on-chip directory rather than snooping. In those designs, a key challenge is maximizing the effectiveness of precious on-chip directory state. Most current directory protocols miss an opportunity by organizing all state in per-block records.

To increase the "reach" of on-chip directory state, we apply ideas from snooping region coherence to develop a dual-grain CMP directory protocol. First, we trade enable a tradeoff between unnecessary probes (e.g., invalidations) and on-chip directory storage size by organizing a directory entry with both per-1KB-region state and per-64B-block state. Second, to optimize for sparsely accessed regions, we evaluate an asymmetric dual-granularity directory, wherein some entries are smaller and can hold only one block per region rather than as many as to 16.

Results with commercial and PARSEC workloads on a 16-node CMP show that the new dual-grain CMP directory design uses less space, or usually does fewer unnecessary probes, than conventional designs and eliminates directory accesses for many private blocks.

1. Introduction

Moore's Law still enables architects to double chip multi-processor (CMP) core counts, albeit under power constraints. As mainstream parallel applications and operating systems [4]

remain mostly reliant on the shared-memory programming model, cache-coherence protocols, which keep caches functionally transparent, critically determine CMP performance and efficiency.

Because industry-standard snooping protocols do not scale well, future CMPs will likely need to use directory protocols [17, 18]. However, CMPs are unlikely to use a classic directory protocol with directory state at off-chip memory for two reasons. First, CMP designers seek to avoid the latency and power overhead of off-chip accesses for data cached on-chip. Second, they want to leverage commodity DRAM and avoid the large specialized off-chip directory storage.

For these reasons, a likely CMP directory design distributes directory information among banks of a logically shared last-level cache (the L3 in our case), but does not store any information off-chip. When finite directory state must store information for a new block and no space exists, the directory victimizes another block and invalidates cached copies of the victim. Maintaining the invariant that "a block not present in the on-chip directory is not cached on-chip" allows such a directory protocol to operate without off-chip state.

Arguably, the key design challenge for this CMP directory design is to maximize the effectiveness of precious on-chip directory state. Most existing proposals organize CMP directory state at the granularity of a block (e.g., 64 bytes) and differ only in the structure of the per-block entry (e.g., full bit vector or coarse-grain bit vector). In contrast, our position is that a more effective CMP directory can be achieved by organizing the directory at a dual granularity in which some state pertains to many cache blocks (e.g., 1K bytes). Such a design can multiply the "reach" of precious on-chip state.

To build our more efficient dual-grain CMP directory, we leverage the foundation of region coherence, which was previously proposed for conventional snooping [2, 10] and unconventional

virtual-tree multi-cast coherence systems [14, 15]. A region is an aligned multi-block range of physical memory (e.g., 1K bytes of sixteen 64-byte blocks). For the conventional snooping applications of region coherence, on an initial miss to a region, all other nodes extend their response to include what other blocks they cache in the region. The requestor then stores that additional region information so it can determine whether subsequent blocks in a region are not cached elsewhere (i.e., private) and, if so, satisfy the miss with a direct, non-broadcast request to memory. While the initial region coherence proposals RegionScout [22] and coarse-grain coherence [5] used supplemental structures to store the region information, the subsequent RegionTracker proposal [35] improved the storage overhead of region coherence by combining the cache tags and region information primarily into a single set-associative cache-like structure called the region vector array (RVA).

Due to the storage efficiency of the RVA design, we use the same basic structure to extend the reach of our dual-grain CMP directory. With the RVA, we develop a directory protocol that addresses engineering issues, including how nodes inform the directory about what they cache and how the directory revokes a node's permission to cache. Moreover, our design facilitates direct access to memory for private regions, transferring a benefit of snooping region coherence to a CMP directory.

At a higher level, a dual-grain CMP directory with region coherence presents design challenges and opportunities. A first challenge is deciding what information should be provided at the region or block granularity. Tracking sharers, for example, is more compact if done once per region but more precise if done per-block. A second challenge is designing the RVA to make effective use of precious on-chip storage in the presence of workload behaviors like many blocks being private or the presence/absence of spatial locality. A key opportunity is investigating how

spatial locality, when present, helps with these new design questions. To this end, we find two key results.

Dual-grain directory coherence. First, we find that the RVA-based directory should *neither* track sharers at just the block or region level. Rather, each entry should be a hybrid with two parts. One part has a field for each block that tracks one sharer or owner for that block, which we call the single-ID method [8]. The second part is a single field for the whole region that tracks the cumulative set of sharers of all cache blocks in the region beyond the individual block-level single-ID. Compared with tracking each block's full sharing list, the proposed design uses less state (e.g., 2,158 rather than 3,264 kilobits for 16 nodes), while sending extra probes for invalidations only in the uncommon case that (i) there is write to shared block, (ii) there are multiple shared blocks in a region, and (iii) those blocks have been shared by a different set of nodes. In sum, this design exploits a form of spatial locality wherein the multiple contiguous blocks of a region tend to be cached by the same or similar node(s).

Asymmetric RVA ways. Second, while the RVA is cache of region entries, we find that it should use asymmetric ways. To this end, we take four entries out of sixteen entries in each set (i.e., four ways) and make them singleton RVA entries that allow only one block per region to be cached. The result is a smaller RVA; reducing on-chip state and the change has little effect on performance, because empirically many regions have only one cached block during their on-chip lifetime. Furthermore, we show how the design can cope with promoting blocks from singleton to full RVA entries (an "intra-RVA" miss) when a second block in a region gets cached. In summary, singleton entries mitigate the potential negative impact that sparely accessed regions would otherwise have on directory capacity.

We simulate the new dual-grain coherence against several alternative per-block protocols (full bit vector, coarse-grain bit vector, and single-ID) with commercial and PARSEC workloads on a 16-node CMP. We find that the new dual-grain CMP directory design uses less space and/or usually suffers fewer unnecessary coherence probes (e.g., invalidations) than per-block designs, and can eliminate directory accesses for many private blocks.

2. Motivation

This section motivates dual-granularity region coherence by discussing why extending prior directory coherence solutions may not be efficient for future many-core systems. While numerous directory cache-coherence protocols [1, 8, 17, 18] have been productized, this section concentrates on two prior solutions -- the SGI Origin [17] and AMD's Probe Filter [8] protocols -- that lie on the opposite sides of the bandwidth-storage spectrum on which our dual-granularity directory coherence builds.

The section also introduces our targeted many-core system and discusses the inefficiencies of extending the two prior solutions to our targeted system.

2.1. Prior Directory Coherence Solutions

Bit vector of sharers (SGI Origin-like). The SGI Origin directory protocol [17] maintains coherence for a large system by storing detailed sharing information in a distributed directory. Specifically, the distributed directory stores sharing information in a bit vector at the memory-block granularity. When only a few nodes share a cache block, each bit represents whether a particular node is currently caching the block (full bit vector); for larger systems, when many nodes share a block, each bit represents a particular group of nodes (coarse-grain bit vector). In this way, the directory knows with good precision which nodes should be probed for both read and writes requests. For example, for read requests to owned blocks, the directory probes only

the owner, thus ensuring up-to-date data is supplied to the requestor. Meanwhile, for write requests to shared blocks, the directory sends probes (invalidation requests) only to those nodes with their respective bits set in the sharing bit vector. As a result, the SGI Origin protocol minimizes the total messages required to maintain coherence at the cost of storing considerable state for every memory block in the system. For SMP configurations with a larger number of nodes, SGI Origin mitigates storage overhead by combining multiple nodes into a coarse-grain bit vector at the cost of generating unnecessary probes due to less precise sharing information.

Single-ID (AMD Probe Filter-like). At the other extreme, the AMD Probe Filter protocol [8] maintains coherence for small combinations of CMPs using the single-ID methods that point to one cache. In particular, the single-ID identifies the node storing the block in the traditional M, O, or E states [32], as well as the node storing an S copy when only one S block copy exists. For a read request, the directory simply forwards the request to the identified owner. The directory then understands that multiple nodes now share the block, assuming that exclusive permission does not migrate to the requestor [9, 31]. Similarly for a write request in which the directory identifies an exclusive owner, the directory simply forwards the request to the current owner. However, for a write request in which the directory identifies that multiple nodes may be caching the block, the request is broadcast to probe all nodes to ensure all existing shared copies are invalidated. The AMD Probe Filter protocol necessitates broadcasts for certain write requests as well as requiring broadcasts to maintain inclusion between the on-chip directory and all blocks currently cached.

For blocks potentially shared by multiple nodes, directory entry replacements -- like write requests -- must be broadcast to ensure inclusion. Overall, in comparison to the previously described SGI Origin protocol, the AMD Probe Filter protocol uses significantly less storage per

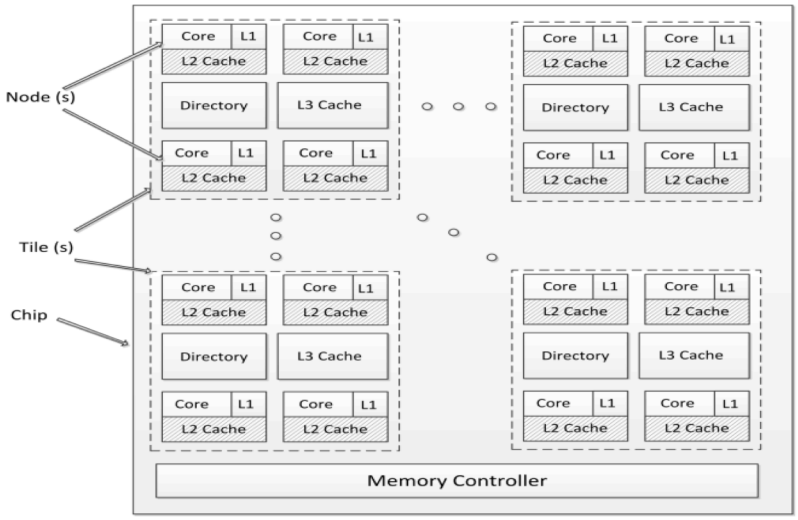

Figure 1. Target System Architecture

directory entry than a bit vector (log2(n) bits versus n bits for n nodes), but at the cost of increasing unnecessary probes.

2.2. Target Many-core System

While the previously discussed directory protocols maintain coherence for either large SMPs or small combinations of CMPs, our targeted system is a future many-core CMP. Figure 1 illustrates this targeted system with three levels of cache hierarchy. To reduce access latency for the most frequently requested data, the L1 and L2 caches are private to each core. To increase the effective L3 cache capacity, the L3 cache is logically shared among all the on-chip cores and its data content is non-inclusive with respect to the data contents of the private L1-L2 cache hierarchy. A coherence directory at the L3 hierarchy is used to maintain the coherence among the private L1-L2 cache hierarchies. Similar to the L3 cache, the coherence directory is logically shared based on address but banked and distributed across the chip. For each cache-block address, there is a unique bank (home) responsible for maintaining coherence for that block.

2.3. Inefficiencies with Extending Previous Directory Protocols to Many-core CMPs

This section discusses the inefficiencies incurred when extending the previous directory protocols to our targeted many-core CMP.

Storage overhead of bit vectors. While the SGI Origin off-chip directory stored the bit vector for all memory blocks, one can easily imagine extending the bit vector concept to an on-chip directory that tracks only those blocks currently cached on the chip. As pointed out in Section 2.1, storing sharing information in a bit vector minimizes probes. However, researchers have observed that for multi-threaded commercial workloads, the majority of the cache blocks tend to have a limited number of sharers [20]. Thus, representing every node with a separate bit wastes significant on-chip storage. To minimize this waste, a coarse-grain bit vector can be used. However, a coarse-grain bit vector either lacks the ability to unset the sharing information when a block is replaced or requires complicated localized broadcasting support to coordinate block replacements. Moreover, due to lack of accurate sharing information, a coarse-grain bit vector generates redundant coherence probes.

Unnecessary probing of single-ID. Due to the lack of accurate sharing information, single-ID protocols suffer from unnecessary probing, which is exacerbated as more cores are integrated on the chip. The result is that these protocols waste power (due to unnecessary tag look-ups and network utilization) and achieve limited scalability (from higher protocol occupancy due to unnecessary messages). Furthermore, replacing directory entries is wasteful; if the corresponding cache block is in shared state with more than one sharer, then invalidations must be broadcast to maintain the invariant that a directory entry always exists for any on-chip cache block.

3. Efficient Directory Coherence for Many-core CMPs

Prior solutions suffer from excess storage overhead or unnecessary probing. This section describes how we achieve more efficient coherence through our region-based directory design. Section 3.1 describes that by combining the bit vector and single-ID solutions into dual-granularity directory design, the directory reaps the benefits of both solutions while minimizing the costs of each. Section 3.2 explains how our asymmetric RVA design reduces directory storage by leveraging the observation that many regions contain only one valid cache block.

3.1. Dual-granularity Directory Coherence

Maintaining coherence at both the block and region levels better balances the directory's storage overhead and demand for bandwidth while allowing it to leverage spatial behavior opportunistically at the region granularity. The result is that the dual-granularity design more efficiently utilizes directory storage and reduces the unnecessary messaging required to maintain coherence. In particular, maintaining detailed sharing information at the region granularity amortizes the overhead across fewer entries, while incorporating limited sharing information at the block granularity can still facilitate efficient per-cache-block transfers. Furthermore, by maintaining coherence permissions at multiple granularities, the dual-granularity directory design can remove unnecessary directory look-ups similar to prior region-based snooping protocols removing unnecessary snoops. The remainder of this sub-section focuses on these two benefits in detail.

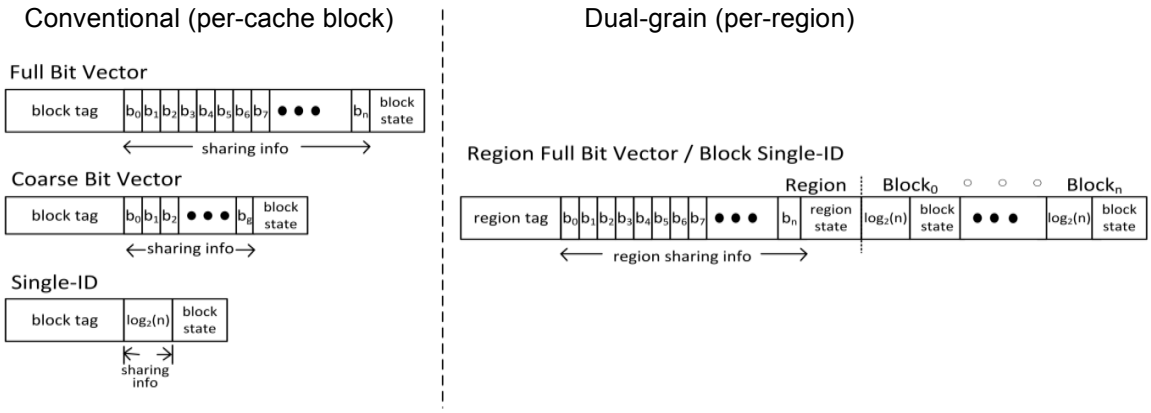

Figure 2. Directory Entry Comparison. The full bit vector tracks at node (n) granularity, while the coarse-grain bit vector tracks sharers at group (g) granularity.

Dual-granularity sharing information. The left side of Figure 2 illustrates the entries of the bit vector (both full bit and coarse-grained bit implementations) and the single-ID conventional directories. Figure 2 highlights that though these different types of entries may hold the same coherence state and tag data, the difference in sharing information substantially increases the total size of the bit vector entries compared to the single-ID entries.

Meanwhile, the right side of Figure 2 illustrates how bit vector and single-ID information can be unified into a single entry for the dual-granularity directory design. Specifically, the entry's region portion stores the region tag, a bit vector of sharers, and region coherence state, while the entry's block portion tracks each block's single-ID sharing information and block coherence state. In comparison to conventional block-level directories, the dual-granularity directory requires more storage per entry because both sharing information and coherence state (a.k.a, coherence permission) is tracked at the block and region levels. However, while the dual-granularity entry is larger, the dual-granularity directory requires far fewer entries than the conventional directory to cover the equivalent number of cache blocks.

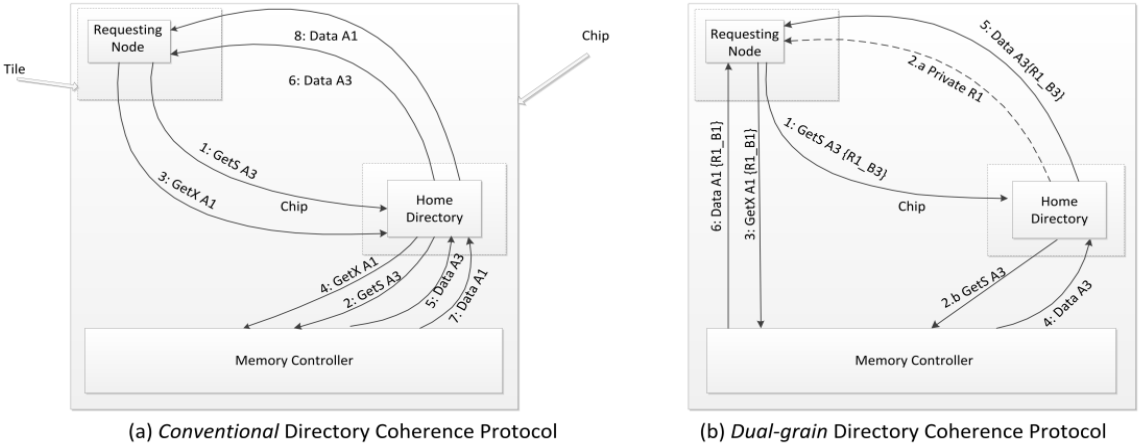

Figure 3**. Region Privatization Example.** Rx_y stands for blocks address that maps to region address "x" and the "y" block within region.

In fact, for reasonably sized regions (i.e., 1 KB), the dual-granularity directory can track several times the number of cache blocks and still require less storage than a conventional full bit vector directory, albeit at the cost of a potential reduction of efficiency due to internal fragmentation within a region and increased conflict misses.

Tracking the sharers at the region level while tracking the owner at the block level matches the behavior of much multi-threaded software. In multi-threaded applications, large regions of code and data tend to be shared by several coordinating threads. Therefore, the region-level bit vector tracking the current sharers ensures that only those processors operating on specific shared regions are probed. Meanwhile, frequently written data, like synchronization variables, tends to be passed around from processor to processor and thus necessitates fine-grain control. Thus, storing single-ID sharing information at the block level can efficiently coordinate these block-level migratory sharing patterns [9, 31] with minimal information.

Dual-granularity permissions. In addition to tracking sharing information at both the region and block levels, our dual-granularity directory also tracks coherence permissions at both levels, enabling region privatization optimizations similar to those performed by the previous snooping-

based region proposals [5, 22]. In particular, the directory may identify that a given region is accessed only by a single node and thus assign private (exclusive) region permissions to the node. Thereafter, later misses within that private region can access the off-chip memory directly.

Figure 3 illustrates an example in which region privatization avoids home (selected statically based on address) directory indirection compared to a conventional block-based directory protocol. For the conventional protocol, Figure 3(a) shows that both GetS requests, to addresses A1 and A3, must consult the home directory before travelling to off-chip memory. However, the directory look-ups would be unnecessary if the requesting node knew the requested blocks were not available on-chip. In contrast, by tracking coherence permissions at the region level, the dual-granularity directory can assign private permissions of the entire region, R1, when responding to the first GetS request for A3 (i.e., the third block, B3, of region 1, R1). Thus the subsequent requests from the node, such as the illustrated GetS request for A1, are sent directly to the memory controller. While the preceding example demonstrates the benefit of opportunistically using region-level coherence, one can easily envision the need to fall back dynamically on block-level coherence later if the region is accessed by other cores. Section 4 explains how the dual-granularity directory efficiently handles dynamic switching between the coherence granularities.

Overall, though the latency reduction is relatively small when compared to an off-chip memory access, the privatization of regions eliminates many unnecessary directory look-ups, saving power and reducing resource demand and protocol occupancy at the directory.

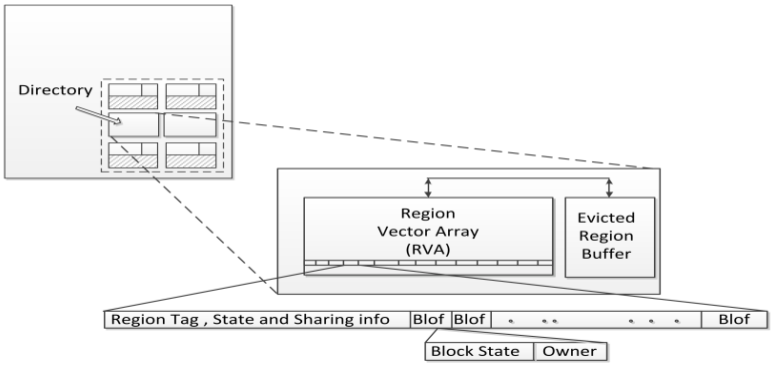

Figure 4. Basic Data Structures for Maintaining Region Grain Coherence.

3.2. Asymmetric RVA Ways

As mentioned in the previous sub-section, the greater peak tag reach of the region-based directory comes at the expense of potential reduced storage efficiency due to internal fragmentation within a region. Thus, for applications with sparse working sets, the region-level directory structure can suffer from more conflict misses than a conventional design, in spite of having greater tag reach. We address this inefficiency by proposing a novel asymmetric RVA design. We empirically observed that many regions have only one distinct cache block accessed during its on-chip cache lifetime. This observation is in line with similar observations made in the context of spatial prefetching [29]. This led us to propose an asymmetric RVA design in which *not* all ways of a set (row) are equally capable.

Our novel asymmetric RVA design builds on the data structures proposed in RegionTracker [35]. In particular, Figure 4 depicts L3 directory implemented with two of RegionTracker's main components: the region vector array (RVA) and evicted region buffer (ERB). The RVA is a set-associative structure in which each entry contains both region-level and block-level information.

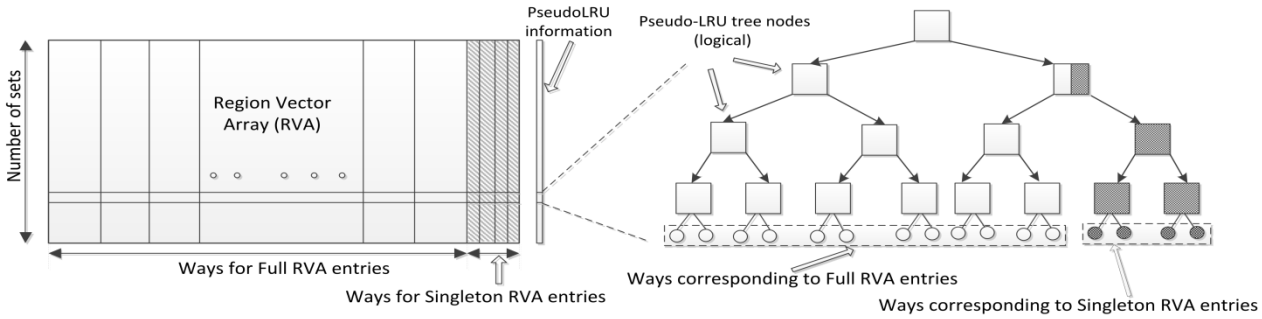

Figure 5. Asymmetric Region Vector Array Design. The ways holding small entries are shaded in dark gray.

To be consistent with the prior work, Figure 4 labels each block's information as a "BLOF" (cache BLOck inFormation), and each BLOF contains the block's sharing information and coherence state (previously shown in Figure 2). The ERB is a much smaller CAM structure that temporarily holds evicted RVA entries. Because multiple individual cache blocks may require invalidation on a region eviction, the ERB helps quickly unblock an RVA entry while performing the actual replacement actions off the critical path. Similar to Zebchuk et al., we found a small ERB (16-entry) avoided back-pressuring RVA allocations.

Our design improves the original RegionTracker RVA design through asymmetric ways. The left side of Figure 5 shows the asymmetric design in which each RVA set contains a few ways with singleton entries (not to be confused with single-ID sharing information). Singleton region entries can track only one cache block within the region (i.e., they have only one BLOF within each entry). The remaining ways contain full-region entries (i.e., region entries with individual BLOFs for each cache block in the region).

Though the ways are asymmetric, our RVA design uniformly manages pseudo-LRU information and replacements across all ways. The right side of Figure 5 illustrates how our binary tree-based pseudo-LRU policy [13, 28] represents each way within a 16-way asymmetric

set. On an initial region miss, the entire binary tree is searched to identify the entry to replace. Because the initial miss requests a single cache block, any entry -- either full or singleton -- can hold the requested region.

However, this asymmetric RVA design must cope with an intra-RVA miss when a node seeks to cache a second block in a region heretofore stored in a singleton RVA entry. There are three basic cases depending on how L1/L2 cache replacements have affected full RVA entries in the set. First, if there is a vacant full RVA entry, we move the former singleton entry there. Second, if there is a full RVA entry that is singleton-eligible (currently holds only one cache block in the region), we swap it with the former singleton entry. Third, if all full RVA entries point to multiple nodes (cores), we choose a victim following pseudo-LRU binary tree walk constrained to only ways corresponding to full RVA entries (lightly shaded in Figure 5) and move the former singleton entry there.

4. Implementing Dual-granularity Directory Coherence

This section describes the implementation details of dual-granularity directory coherence by answering a series of questions pertaining to the design. With these answers, we highlight several interesting mechanisms that improve the design's efficiency.

How do the L2 caches manage their region permission? Section 3.1 discussed how the directory can assign exclusive region permission to an L2 cache, enabling future L2 cache requests to bypass the directory and directly request data from the memory controller.

However, to perform the optimization, the L2 cache must store the private (exclusive) region permission granted by the directory. By constructing the L2 cache tag structure exactly like the original RegionTracker structure [35], the L2 cache can simply use a bit in the region entry that identifies private permissions to the entire region. In comparison to our RegionTracker-inspired

novel directory design, the L2 does not store single-ID sharing information or cache-block coherence state in the BLOF. Rather, just like original RegionTracker proposal, each BLOF stores a pointer to an entry in a structure called block status table (BST). BST entries maintain one-to-one correspondence to L2 data blocks and each entry holds cache-block state information. Also similar to the original RegionTracker design but dissimilar to our dual-granularity directory design, each BST entry contains a back-pointer to the corresponding RVA entry that is used to update the RVA on a data eviction.

What happens to evicted L2 cache blocks that pertain to an exclusive region? Because the L2 cache stores region permissions in its local RVA, which is decoupled from its data array, L2 data blocks can be evicted while an L2 cache maintains exclusive region permission. To retain those evicted L2 blocks on-chip, L2 forwards the evicted block to the home L3 and sets its local BLOF information to identify that the L3 potentially holds the data (one bit per BLOF). Thus, further L2 requests for this particular block will query the home directory.

How to reduce duplication between the region-level bit vector and block-level single-ID?

To maximize the filtering of unnecessary probes enabled by our dual-grain directory design, we minimize duplication of sharing information between the region-level bit vector and block-level single-ID. Specifically, when a single owner/sharer exists for a block, the block-level single-ID field can track the sharer, so setting the corresponding bit in the region-level bit vector is unnecessary. Therefore, only non-owner nodes sharing a block must have their corresponding bits within the bit vector set. By reducing the number of scenarios when bits are set in the region-level bit vector, more probes can be filtered by the bit vector when shared cache blocks must be invalidated.

How to reset bits within the region-level bit vector? One challenge that arises with the region-level bit vector that does not exist with a block-level bit vector is that a cache-block write does not allow one to clear the region-level bit vector. The reason is that the protocol invalidates all copies of a cache block in the L1-L2 cache hierarchies while processing the write (like a conventional protocol), but multiple copies of other cache blocks in the same region may still exist in shared state. Thus, it will be incorrect to clear the sharing bit vector for the whole region for a write to one of the blocks in the region. This issue, if not dealt with, can lead to accumulation of stale sharing information at the region level. This in turn can contribute to unnecessary probes to the L1-L2 cache hierarchies from the directory.

We address this issue by making the following enhancement to the protocol. Because the L2 cache tags use a region-based structure, the L2 can respond to invalidation requests mentioning whether they store any other valid blocks within a region by using a single tag look-up. The write requestor then records this information from all previously potential sharers of the region. Finally, when all invalidation acknowledgement messages have been received, the requestor appends the coalesced information to its unblock (request completion) message sent to the directory. Therefore, the directory can potentially reset a sub-set of bits in the sharing vector without having to send extra messages.

How to revoke private (exclusive) region permission dynamically? It may be necessary to revoke exclusive region coherence permission dynamically and fall back to conventional cache block-level coherence. This requirement can arise due to the replacement of a region entry from the directory or, in case of region privatization, if a private region observes an access from a core other than its private owner. The challenge is that once the region-level exclusive permission is assigned to an L1-L2 cache hierarchy, the directory does not remain responsible for maintaining

coherence for the cache blocks accessed in the region. Thus, when the directory requests revocation of region-level private (exclusive) permission from the owner of the region, the owner responds with information about which cache block(s) in the given region is present in its local L1-L2 cache hierarchy. The directory then emulates exclusive (M) coherence state for those cache blocks in the region with the recent region owner as the exclusive owner for those cache blocks. This allows the dual-grain directory coherence protocol to seamlessly fall back on cache block-level coherence because any later request to those cache blocks will be routed to the correct owner and proper coherence action can be taken thereafter.

What other benefits are achieved by region-level coherence beyond privatization? Besides removing unnecessary directory indirection by tracking coherence state at dual granularities and filtering unnecessary probes by tracking sharing information at dual granularities, our region-based directory design removes other unnecessary messages. The directory needs to maintain inclusion with the contents of the cache blocks in the L1-L2 cache hierarchies. Thus, on replacement from the directory, corresponding cache blocks in the L1-L2 cache hierarchies need to be invalidated first. By maintaining coherence at the region level, only a single invalidation to an L1-L2 cache hierarchy is enough to invalidate all the cache blocks in that region. This can potentially save a number of invalidations to the L1-L2 cache hierarchies.

5. Evaluation

In this section we evaluate our proposed dual-grain directory coherence scheme by describing storage costs of alternative directory designs, our simulation methodology, workloads, and simulation results of our proposed system and alternative conventional block-level designs.

5.1. CMP Directory Design Alternatives Evaluated

In this sub-section, we articulate the four alternative CMP directory formats (three conventional, one dual-grain) for 16-core CMP. They are:

- *FullBit*: 16-bit sharing vector per cache block; each bit represents a core.
- *CoarseGrained*: 4 bits sharing vector per cache block; each bit represents a group of four cores.
- *SingleID*: Only single sharer/owner pointer (4 bits) per cache block.
- *DualGrainedFS*: A dual-grain entry with:
  - 16 bits of sharing vector per 1KB region ("F") that point to any sub-set of 16 cores.
  - For each of the region's 16 blocks, 4-bit single-ID per block ("S") that points to one of 16 cores.

5.2. Directory Storage

In this section, we present the storage cost for the region-grain structure in the proposed system. The primary structure that is new in the proposed dual-granularity directory coherence scheme is the asymmetric RVA implementation of the directory at the L3 cache hierarchy. In Table 1, we list storage requirements for the four different implementations of the directory for 16-core and 32-core CMPs.

Table 1. Directory Entry Storage (in kilobytes and normalized percent)

	Per-block Full Bit Vector (*FullBit*)	Per-block Coarse-grain Bit Vector (*CoarseGrained*)	Per-block Single-ID (*SingleID*)	Dual-grain Coherence: Per-region Full Bit Vector & Per-block Single-ID (*DualGrainedFS*)
16 cores	3264 kb (100%)	2496 kb (76%)	2496 kb (76%)	2158 kb (66%)
32 cores	4288 kb (100%)	2752 kb (64%)	2560 kb (59%)	2610 kb (61%)

We observe:

- Less storage than full bit vector: Dual-grain coherence (with full bit vector at region and single-ID at block level) uses less storage than the per-block full bit vector (approximately a 40% reduction).
- Comparable storage to others: Dual-grain coherence uses comparable storage to other per-block protocols (coarse-grain bit vector and single-ID); Section 5.4 shows it performs better.
- Scales well: Dual-grain coherence scales well to more nodes because it stores a bit vector only once per region (e.g., 32-bit vector for a 1KB region is 0.4% storage overhead).

The storage advantage of *DualGrainedFS* comes from two primary sources: better amortization of the tag overhead over larger region granularity compared to block granularity and better amortization of storage overhead of detailed full bit vector over larger region granularity.

Moreover, as mentioned in Section 4, we implemented caches at L2 as region-grain structures similar to the original RegionTracker's [35] data structures to keep region-level permissions at L2, thus enabling optimizations like region privatization. A fringe benefit of region-grain implementation of L2 cache is tag storage savings. When we implemented our baseline 256 KB, 8-way set-associative private block grain L2 cache's (Table 2) tag structures at region (1 KB) granularity with four times the tag reach of the conventional block-level cache (i.e., 1 MB), the storage cost of the tag structure of each private L2 cache decreased by about 11% (136 kBits for region-grain L2 cache compared to 152 kBits of conventional block-level L2 cache).

5.3. Simulation Methodology

Simulation infrastructure: We use a detailed full-system simulator that combines the CPU and device simulation of Michigan's M5 [2] with the memory-system model of Wisconsin's GEMS [21]. We use x86 full-system simulation with the simulator running unmodified Linux 2.6 operating system. We run workloads from the PARSEC benchmark suite [24]. We also use memory reference traces from various server-class commercial workloads.

Table 2. Baseline System Configuration

Core	16, in-order, 2 GHz
L1 Caches	Private, 32 kb, 4-way, split I&D, write back
L2 Caches	Private, 256 kB, 8-way, inclusive to L1
L3 Cache	Shared, 4 MB, 16-way, non-inclusive
Coherence Directory	At L3 level, 4 MB tag reach, 16-way, MESI-like coherence
Memory	4 GB, ~ 300 cycle round trip

Baseline: In Table 2, we present the basic parameters for our targeted baseline system. We use 16-core CMP with three levels of cache hierarchy. L1-L2 is private to each core and inclusive, while the last-level cache (L3) is logically shared and non-inclusive with respect to the contents of private L1-L2 cache hierarchies. A coherence directory at the L3 level has the onus of maintaining on-chip coherence.

Workloads: Table 3 describes workloads used in the results section to evaluate our proposed scheme compared to conventional directory schemes.

5.4. Simulation Results

Our goal is to make CMP directory coherence protocols more efficient -- both in terms of precious on-chip storage cost and in terms of the number of messages required to maintain the coherence

Web servers	SPECweb2005 with different configurations
Database servers	OLTP with three different back ends and one DSS query workload
Cloud services	Two workloads from popular cloud services
Misc. server	SAP, Java Virtual Machine, Terminal Services, and Virtualization workload
PARSEC	10 programs from PARSEC benchmark suite

Table 3. Workloads Description

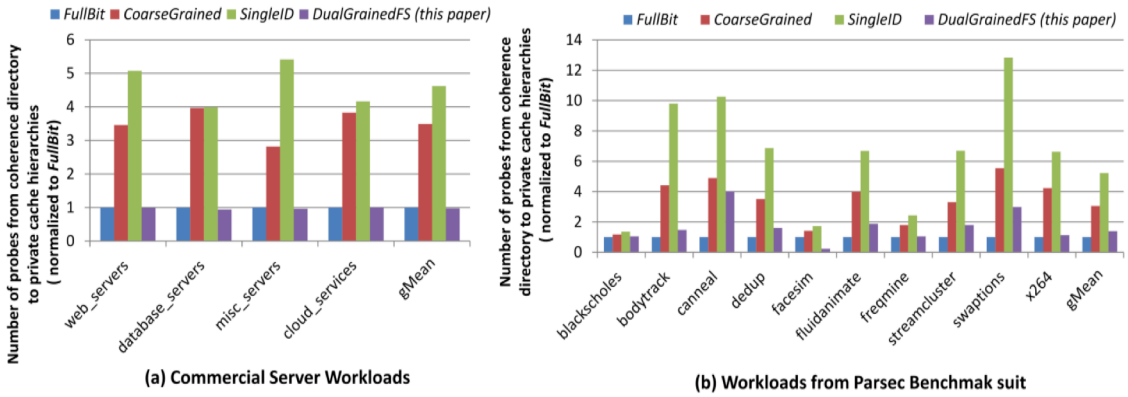

Figure 6. Comparison of Number of Coherence Probes Generated by Different Directory Schemes.

of on-chip data. To evaluate how we fared against these objectives, we ask and answer several questions

Can *DualGrainedFS* use the same number of or fewer probes than conventional protocols? Figure 6 compares the probes sent by *FullBit, CoarseGrained,* and *DualGrainedFS*, normalized to *FullBit* (shorter column is better). Figure 6(a) provides results for commercial server workloads, while Figure 6(b) depicts results for workloads from the PARSEC benchmark suite. Results first show that *DualGrainedFS* always has fewer probes than *CoarseGrained*, reducing probes by a factor of about 3.6x while reducing storage requirements by 12% as well. Thus, *DualGrainedFS* is preferred to *CoarseGrained*. For *SingleID*, *DualGrainedFS* provides as much as about 4.8x reduction in the number of probes for commercial workloads while requiring a little less storage.

Second, *DualGrainedFS* often has a comparable number of probes as *FullBit* (even through the latter uses about 50% more storage) for commercial workloads. A closer analysis reveals that *DualGrainedFS* performs well for workloads whose memory footprint puts pressure on the CMP directory (e.g., commercial workloads, Facesim), but the larger state of *FullBit* prevails for smaller-memory-footprint workloads (many of PARSEC). *DualGrainedFS* more gracefully

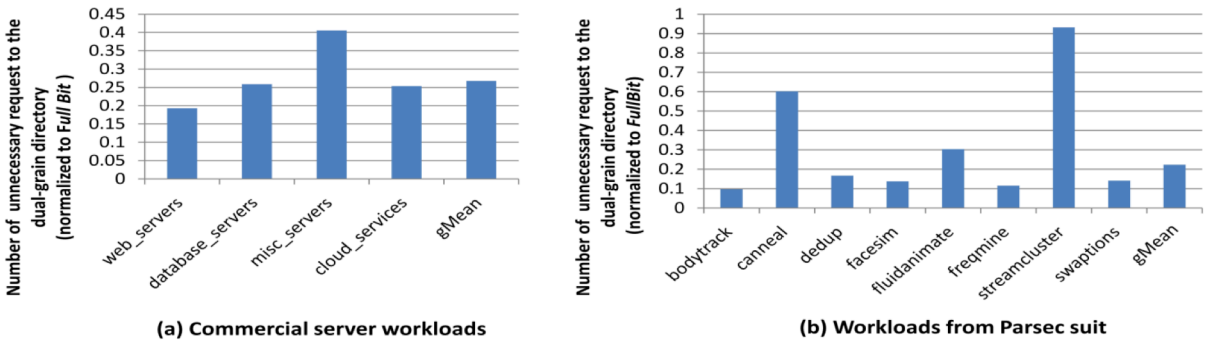

Figure 7. Normalized Number of "Unnecessary" Look-ups to the Coherence Directory.

handles large-memory-footprint workloads because its region entries "reach" more blocks, provided adequate spatial locality, and replacing a region requires only one invalidation for each sharing core instead of invalidations for each cache block in the region (Section 4). Thus, *DualGrainedFS* is preferred to *FullBit* for workloads with large memory footprints and/or whenever on-chip storage is precious. Overall, dual-grain coherence provides better value for the on-chip storage invested for the directory.

How often can *DualGrainedFS* eliminate unnecessary directory indirections? While the primary benefit of *DualGrainedFS* is its more effective use of CMP directory state, a secondary benefit is that misses to blocks cached by at most one node can go directly to memory without accessing directory entry at the home node (Section 3.1), thereby avoiding a waste of latency, interconnect energy, and bandwidth as well as directory energy and protocol occupancy. But how often are unnecessary directory indirections eliminated?

To this end, Figure 7 displays unnecessary directory requests, normalized to *FullBit*, for commercial workloads (left) and PARSEC (right), where shorter is better. Results show that *DualGrainedFS* reduces unnecessary directory indirections 10-90%, providing an additional benefit.

What about performance? The primary goal of the work is to provide better storage and coherence message bandwidth trade-off for large CMPs and reduce unnecessary coherence directory look-ups. We also help performance somewhat by shortening average off-chip memory access latency by avoiding unnecessary indirection through the home node directory. We observed that the average memory latency for memory-intensive commercial workloads decreased 7% on average while the performance improved by around 5% on average, compared to conventional full bit vector directory protocol implementation (not presented here). For less memory-intensive workloads from the PARSEC benchmark suite, we found that our techniques improved performance by an average of 2.5% and -- more importantly -- never significantly hurt performance.

Power savings? With reduction of cache probes to the L1-L2 cache hierarchy and elimination of many unnecessary look-ups at the coherence directory, we save power as well. We used CACTI 5.3 [27] to estimate the dynamic power of coherence data structures at the private L2 cache and the L3 cache hierarchy. Following the methodology used by Lotfi-Kamran et al. [19], we modeled the tag array as SRAM storage with ITRS-LSTP (low standby power) transistors. Our study shows that with reduction of probes from the coherence directory that looks up L1-L2 cache hierarchies, on average we reduced 66% and 47% of dynamic power for tag look-up of private L2 caches for coherence probes with coarse-grain bit vector directory for commercial and PARSEC workloads, respectively.

A recent work [19] showed that in large CMPs, significant power is spent looking up the coherence directory. We find that our scheme of reducing unnecessary look-ups saves around 20% and 16% of dynamic power for request look-up at the directory for commercial and PARSEC workloads, respectively.

Sensitivity analysis. To understand how our proposed system scales to a system with more than 16 cores on a chip, we ran experiments with many of the described workloads in a 32-core CMP configuration. This verified that for 32 cores, our coherence scheme of dual-grain directory coherence protocol was able to provide much better trade-off between storage and number of coherence probes, similar to a 16-core CMP.

6. Related Work

We draw inspiration for this work from the works for snoop-based coherent systems that exploit region granularity information to eliminate many unnecessary broadcasts [5, 22]. Researchers also looked into reducing snoop energy for a virtual first-level cache by keeping sharing information at virtual-page granularity [10]. Our work is different from this work because we focus on scalable directory coherence protocols for CMPs instead of snoop-based protocols for SMPs.

Spatiotemporal coherence tracking (SCT) [38] proposed by Alisafaee relates most closely to our work. Similar to our proposal, SCT makes use of dual-grain coherence tracking at both page and block granularities. However, while SCT aims to reduce area of a coherence directory, we focus on reducing coherence probe messages to private caches and unnecessary directory look-up that helps us save power.

The next most closely related work is possibly virtual tree coherence (VTC) [15]. Unlike our work here, VTC relies on the non-conventional on-chip network support of virtual circuit tree multi-casting [16] to create a virtual tree of sharers of a region dynamically. That work used the virtual tree of sharers to build a local snoop-like coherence protocol without requiring invalidation acknowledgements within the sharers of a region. This leads to challenges in

implementing standard memory consistency models like TSO [26] because it requires a non-local tree fence to ensure proper write-to-read ordering [14 (Section 6.4)]. Our work does not suffer from any of these issues because we extend conventional directory coherence protocols. We also exploit sharing information at dual granularity -- at both cache-block and region levels to offer better storage and bandwidth trade-offs.

In the context of prefetching for off-chip memory accesses, there has been work [6, 11, 29, 30, 33] that exploits the knowledge of which cache blocks in a given region are generally accessed in a co-related fashion. They try to prefetch other co-related cache blocks in a region when an access to a cache block in the region is observed. These works differs among themselves depending on the prediction scheme used to record co-related cache blocks in the region and/or support they require from compiler and/or hardware.

Our work relates to other recent research to scalable on-chip directory coherence. Zebchuk et al. [34] proposed the tagless coherence directory (TL), which replaces the coherence directory with a set of Bloom filters [3] that track the blocks cached in each core. Our approach provides similar storage reduction, but unlike TL, our region-level tracking also provides semantic sharing information that can be exploited for performance improvement, such as the region privatization optimization described in Section 3.1, counteracting the performance loss incurred due to reducing the accuracy of sharing information.

Qian et al. proposed ScalableBulk [25] that extends BulkSC [7] to allow fast commit of non-conflicting chunks of instructions in an all-time-transaction (or chunk) environment. Kelm et al. proposed WayPoint [36] to use a hardware-managed chained hash table for designing scalable coherence directory. More recently, Ferdman et al. proposed Cuckoo directory [37] that uses Cuckoo hash table organization for on-chip directory to improve its area and energy efficiency.

All of these works are based on single-granularity (cache-block) coherence, unlike our proposal, and ScalableBulk requires non-conventional chunk/transaction commit capability to function.

Finally, there have been decades of work on scalable directory coherence [17, 18, 23], most of which is primarily focused on SMPs instead of recent CMPs. Piranha's [1] design of directory coherence for on-chip coherence using duplicate tags and the work on sparse directory [12] are worth noting. All these protocols are based on a single-grain coherence granularity (cache block), which distinguishes our work on dual-grain coherence.

7. Conclusion

To increase the "reach" of on-chip directory state, we proposed a dual-grain CMP directory protocol that is both more efficient than conventional directory designs and extends the reach of region-based structures with asymmetric ways. We demonstrated that by combining region-level and block-level coherence information into directory entries, the directory can utilize directory storage more efficiently and reduce the unnecessary messaging required to maintain coherence. Our asymmetric directory design also leverages that many regions contain only one valid block and reduces the storage overhead for tracking those regions.

8. References

[1] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. In 27th Annual Intl. Symp. on Computer Architecture, 2000.

[2] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The M5 Simulator: Modeling Networked Systems. IEEE Micro, 26, 2006.

[3] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, 13(7): July 1970.

[4] S. Boyd-Wickizer, A. T. Clements, Y. Mao, A. Pesterev, M. F. Kaashoek, R. Morris, and N. Zeldovich. An Analysis of Linux Scalability to Many Cores. In 11th USENIX Symp. on Operating Systems Design and Implementation, November 2010.

[5] J. F. Cantin, M. H. Lipasti, and J. E. Smith. Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking. In 32nd Annual Intl. Symp. on Computer Architecture, June 2005.

[6] J. F. Cantin, M. H. Lipasti, and J. E. Smith. Stealth prefetching. In Intl. Conf. on Architectural Support for Programming Languages and Operating Systems, October 2006.

[7] L. Ceze, J. Tuck, P. Montesinos, and J. Torrellas. BulkSC: Bulk Enforcement of Sequential Consistency. In 34th Intl. Symp. on Computer Architecture, June 2007.

[8] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes. Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor. IEEE Micro, 30:16-29, 2010.

[9] A. L. Cox and R. J. Fowler. Adaptive Cache Coherency for Detecting Migratory Shared Data. In Proc. of the 20th Annual Intl. Symp. on Computer Architecture, May 1993.

[10] M. Ekman, F. Dahlgren, and P. Stenstrom. TLB and Snoop Energy-Reduction using Virtual Caches in Low-Power Chip-Multiprocessors. In Proceedings of International Symposium on Low Power Electronics and Design, 2002.

[11] Wi fen Lin, Steven K. Reinhardt, and Doug Burger. Reducing DRAM Latencies with an Integrated Memory Hierarchy Design. In Proc. of the 7th IEEE Symp. on High-Performance Computer Architecture, pages 301-312, January 2001.

[12] A. Gupta, W D. Weber, and T. Mowry. Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes. In International Conference on Parallel Processing (ICPP), volume I, pages 312-321, 1990.

[13] J. L. Hennessy and D. A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, third edition, 2003.

[14] N. D. Enright Jerger. Chip Multiprocessor Coherence and Interconnect System Design. PhD thesis, Department of Electrical Engineering, University of Wisconsin-Madison, 2008.

[15] N. D. Enright Jerger, Li-Shiuan Peh, and M. H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In Proc. of the 41st Annual IEEE/ACM International Symp. on Microarchitecture, November 2008.

[16] N. D. Enright Jerger, Li-Shiuan Peh, and Mikko Lipasti. Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support. In Proc. of the 35th Annual Intl. Symp. on Computer Architecture, pages 229-240, June 2008.

[17] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In Proc. of the 24th Annual Intl. Symp. on Computer Architecture, pages 241-251, June 1997.

[18] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In Proc. of the 17th Annual Intl. Symp. on Computer Architecture, pages 148-159, May 1990.

[19] P. Lotfi-Kamran, M. Ferdman, D. Crisan, and B. Falsafi. TurboTag: lookup filtering to reduce coherence directory power. In Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design, 2010.

[20] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared Memory Multiprocessors. In 30th Annual Intl. Symp. on Computer Architecture, June 2003.

[21] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, Min Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. Computer Architecture News, Sept. 2005.

[22] A. Moshovos. RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence. In Proc. of the 32nd Annual Intl. Symp. on Computer Architecture, June 2005.

[23] B. W. O'Krafka and A. R. Newton. An Empirical Evaluation of Two Memory-Efficient Directory Methods. In Proc. of the 17th Annual Intl. Symp. on Computer Architecture, pages 138-147, May 1990.

[24] Princeton. Princeton Application Repository for Shared-Memory Computers (PARSEC). http://parsec.cs.princeton.edu/.

[25] X. Qian, W. Ahn, and J. Torrellas. ScalableBulk: Scalable Cache Coherence for Atomic Blocks in a Lazy Environment. In International Symposium on Microarchitecture (MICRO), December 2010.

[26] P. Sewell, S Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen. x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors. Commun. ACM, 53(7):89-97, 2010.

[27] T. Shyamkumar, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi. CACTI 5.1. Technical Report HPL-2008-20, HP Labs, 2008.

[28] K. So and R. N. Rechtschaffen. Cache Operations by MRU Change. IEEE Transactions on Computers, 37(6):700-709, June 1988.

[29] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial Memory Streaming. In 33nd Annual Intl. Symp. on Computer Architecture, June 2006.

[30] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatio-temporal memory streaming. In Proc. of the 36th Annual Intl. Symp. on Computer Architecture, pages 69-80, June 2009.

[31] P. Stenström, M. Brorsson, and L. Sandberg. Adaptive Cache Coherence Protocol Optimized for Migratory Sharing. In Proc. of the 20th Annual Intl. Symp. on Computer Architecture, pages 109-118, May 1993.

[32] P. Sweazey and A. Jay Smith. A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus. In Proc. of the 13th Annual Intl. Symp. on Computer Architecture, June 1986.

[33] Z. Wang, D. Burger, K. McKinley, S. K. Reinhardt, and Charles C. Weems. Guided region prefetching: a cooperative hardware/software approach. In Proc. of the 30th Annual Intl. Symp. on Computer Architecture, pages 388-398, June 2003.

[34] J. Zebchuk, M. K. Qureshi, V. Srinivasan, and A. Moshovos. A Tagless Coherence Directory. In Proc. of the 42nd Annual IEEE/ACM International Symp. on Microarchitecture, pages 423-434, December 2009.

[35] J. Zebchuk, E. Safi, and A. Moshovos. A Framework for Coarse-Grain Optimizations in the On-Chip Memory Hierarchy. In Proc. of the 40th Annual IEEE/ACM International Symp. on Microarchitecture, December 2007

[36] J. H. Kelm, M. R. Johnson, S. S. Lumetta, S. J. Patel. WAYPOINT: Scaling Coherence to 1000-core architectures. In Proc. of 19th International conference on Parallel Architectures and Compilation Techniques, September, 2010.

[37] M. Ferdman, P Lotfi-Kamran, K. Balet, and B. Falsafi. "Cuckoo directory: A Scalable Directory for many-core systems." In Proc. of 17th Intl. Symposium on High Performance Computer Architecture, February 2011.

[38] M. Alisafaee. "Spatiotemporal Coherence Tracking." In Proc. Of 45th Intl. Symposium on Microarchitecture, 2012.