A Framework for Coarse-Grain Optimizations in the On-Chip Memory Hierarchy

Jason Zebchuk, Elham Safi, and Andreas Moshovos
University of Toronto
{zebchuk, elham, moshovos}@eecg.toronto.edu

Abstract

Current on-chip block-centric memory hierarchies exploit access patterns at the fine-grain scale of small blocks. Several recently proposed techniques for coherence traffic reduction and prefetching suggest that further useful patterns emerge with a macroscopic, coarse-grain view. To exploit coarse-grain behavior, previous work extended conventional caches with additional coarse-grain tracking and management structures considerably increasing overall cost and complexity.

This paper demonstrates that as multi-megabyte caches have become commonplace, coarse-grain tracking and management no longer needs to be an afterthought. This functionality comes "for free" via RegionTracker. RegionTracker is a dual-grain cache design that maintains block-level communication while directly supporting coarse-grain tracking and management. Compared to a block-centric conventional cache of the same data capacity, RegionTracker requires less area to achieve a nearly identical miss rate (within 1%). RegionTracker can be used as the building block for coarse-grain optimizations, reducing their overall cost and easing their adoption. Using full-system simulation of a quad-core chip multiprocessor, commercial workloads, and area estimates based on full-custom layouts on a 130nm commercial technology, we demonstrate the performance and cost viability of the RegionTracker design. We also demonstrate the potential of RegionTracker as a framework for coarse-grain optimizations by showing that it boosts the benefits and reduces the cost of a previously proposed snoop reduction technique.

1. Introduction

Future on-chip caches will most likely grow to several tens of megabytes to compensate for limited off-chip bandwidth, large application footprints, and to meet the demands of multiprocessing and multithreading. This unprecedented on-chip storage offers unique opportunities for improvements beyond conventional cache designs. Our thesis is that *coarse-grain tracking and management*, i.e., tracking information about multiple blocks belonging to coarser memory *regions*[1] and managing the corresponding blocks, becomes increasingly appealing as higher-level (L2 or higher) caches grow larger. Our motivation is that a macroscopic view of access behavior reveals useful patterns that are hard to discern in existing fine-grain hierarchies. Several recently proposed techniques corroborate this observation. Region information and management have been shown to facilitate: 1) performance, bandwidth and power improvements for snoop-coherent shared memory multi-processors [2,4,12], and 2) prefetching for applications with demanding memory footprints [3,16]. These techniques rely on two types of information: 1) whether any block in a region is cached [2,12], and 2) which specific blocks of a region are cached [3,16]. They also require support for selectively fetching or invalidating the blocks of a region [2,16].

Since region tracking and management are not readily supported in existing caches, previous work relied on separate structures. These structures are imprecise [12], incomplete [16], or restrict data placement [2,3]. Data placement restrictions introduce special-case handling complexity in the cache design, while imprecise or incomplete information reduces the benefits of coarse-grain optimizations. Moreover, the relative cost of these structures can be as high as an additional area of 60% compared to a conventional tag array [3]. Commercial designs are more likely to incorporate these optimizations if they had lower area and

[1] A region is a continuous portion of memory comprising a power of two number of blocks.

complexity costs. On-chip area has always been a precious commodity, power is a limiting factor and, thus, designers have to think long and hard before adopting any area and power consuming techniques.

We observe that these optimizations all require very similar functionality. Accordingly, we revisit cache design with coarse-grain management and tracking as first class considerations. We present RegionTracker (RT), a framework for coarse-grain optimizations that reduces the overhead and eliminates the imprecision of these extraneous tracking structures. RT introduces region-level functionality without compromising performance, or area compared to a conventional cache. To avoid bandwidth explosion, communication still uses fine-grain blocks. However, a single lookup in RT is sufficient to determine *which, if any*, blocks of a region are cached and *where*. Moreover, region-level management actions such as region invalidation, migration, and replacement are naturally supported. Accordingly, RT is a *dual-grain* cache design.

The RT design methodology starts with a conventional cache and replaces the tag array with a structure that facilitates region-level lookups and management. The resulting cache requires less area while offering nearly identical performance. As Section 3 explains, RT combines elements from various sector cache designs and introduces additional mechanisms, resulting in improved performance and additional functionality. Compared to previous sector cache designs, RT offers simple block and region lookups and replacements, it does not require higher associativity, and it does not hurt performance, latency, complexity or area.

The key contributions of this work are twofold: first, it articulates why incorporating region information and management in the on-chip hierarchy should be a priority in modern designs; and, second, it presents a framework for incorporating region-level optimizations in the on-chip hierarchy "for free", i.e., without requiring more area or hurting performance. In more detail, the contributions of this work are:

- It revisits cache design with coarse-grain tracking and management functionality as first class considerations presenting RegionTracker. RegionTracker is a coarse-grain replacement of conventional tags that improves upon previous coarse-grain cache designs in the context of large on-chip caches and commercial workloads. It demonstrates that given a conventional cache design, it is possible to replace the tags with an RT design that requires the same or lower associativity and fewer resources, without sacrificing hit rate or cache latency. This work discusses two variants of the RegionTracker design. The first provides full support for region tracking and management. The second builds upon the decoupled sectored cache design [14] to achieve additional area savings at the expense of reduced functionality and increased complexity. For an 8MB L2 cache, using an RT design instead of a conventional design results in a miss-rate increase of only 0.4% while decreasing tag area by 3%. Previous coarse-grain designs that provide the same functionality result in a miss-rate increase of 26% or require a 56-way set-associative structure to achieve a 0.3% miss-rate increase.
- It shows that it is possible to implement a previously proposed broadcast elimination technique for snoop coherent hierarchies with RegionTracker [12] boosting benefits without *any* of the resource overhead. The resulting implementation eliminates 22.5% more broadcasts than the original design while requiring 22KB fewer resources. In fact the proposed design uses even less resources compared to the conventional tag array the RT replaces. We also show that by exploiting precise knowledge about the blocks that are cached per region we can eliminate an additional 13% of broadcasts while still requiring 18KB fewer resources than the original technique.

The rest of this paper is organized as follows: Section 2 first describes the requirements that guided the RegionTracker design, and then describes the design itself; Section 3 describes previous coarse-grain cache designs and presents the second variation of the RegionTracker design; Section 4 evaluates RegionTracker; and finally, Section 5 concludes the paper.

2. RegionTracker Requirements and Design

The starting point for our dual-grain cache design is a conventional cache whose performance and area have been tuned appropriately. The goal is to replace just the tag array of this cache with a structure that can inspect and manipulate regions

comprising several blocks without hurting performance, area, or complexity. Compared to the conventional cache, the resulting cache will have the same data capacity, and, ideally, it will meet the following performance, complexity and area requirements:

1. Communication still uses fine-grain blocks.
2. The miss rate does not increase.
3. The cache latency does not increase.
4. The cache area does not increase.
5. Lookups do not require higher associativity.
6. There is no need for additional cache accesses as a result of regular cache operation (e.g., for replacements).
7. Banking and interleaving are possible.

In addition, the new design will provide the following region-level functionality:

1. A single lookup can determine whether a region is cached.
2. A single lookup can determine which blocks of a region are cached and where.
3. The cache supports region invalidation, migration and replacement.

The first two are needed by previously proposed techniques for snoop-reduction and prefetching, respectively. The third functionality can be useful for on-chip streaming and coherence optimizations, e.g., [10], or for optimizing data placement, e.g., moving regions close to where they are accessed.

2.1. Structural Description

For clarity, in the rest of this section, we assume an 8MB, 16-way set-associative L2 cache with 64 byte blocks, 50-bit physical addresses and 1KB regions, Figure 1(c) shows the indexing scheme used by a conventional cache design with this geometry. As Figure 1(a) shows, RT comprises three components: a region vector array (RVA), an evicted region buffer (ERB), and a block status table (BST). Each of these structures can be banked and interleaved.

1) Each **Region Vector Array (RVA)** entry tracks fine-grain, per block location information for a memory region. Figure 1(b) shows that its organization is independent of the data array. Each entry contains a region tag and several block information fields (BLOFs), one per block in the region, that identify in which data way the corresponding block is cached, if any. For a 16-way

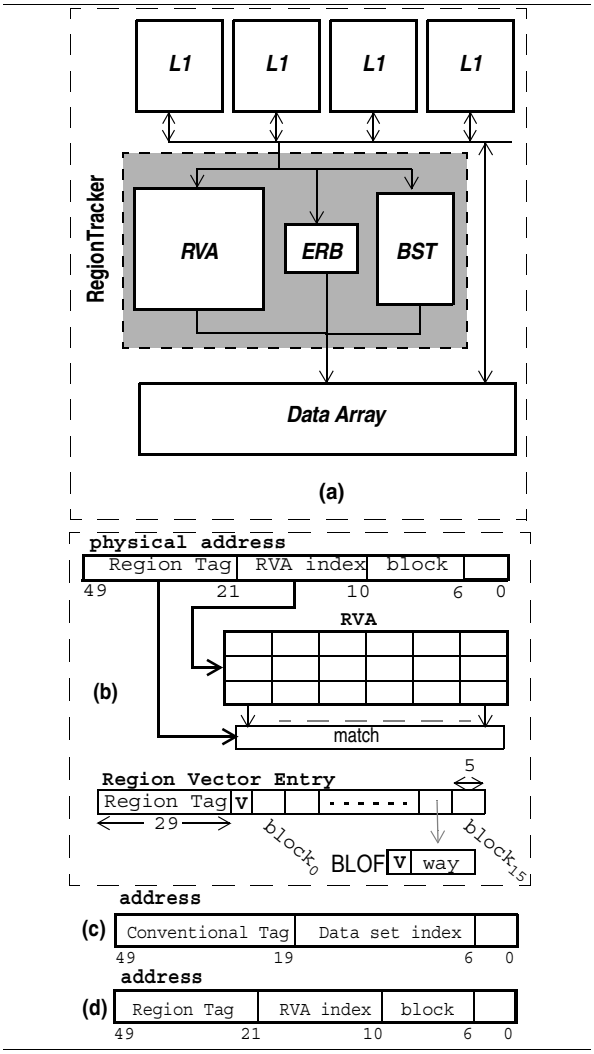

Figure 1. (a) Block diagram of RegionTracker L2 cache design. (b) Region Vector Array. (c) Data array indexing. (d) RVA indexing.

set-associative array, each BLOF contains a valid bit plus four bits for the way. For non-power of two associativities one of the unused way combinations can be used for invalid blocks. In addition to these RVA entries, each set in the RVA contains LRU information. An example of the RVA indexing scheme is shown in Figure 1(d).

2) Evicted RVA entries are copied into the **Evicted Region Buffer (ERB)** eliminating the need for multiple simultaneous block evictions. The ERB does not contain any data blocks, and, thus, it is not a victim buffer. A small ERB (e.g., 12 entries) is sufficient to avoid performance losses. The ERB evicts blocks eagerly from the oldest one third of its entries. In practice, these eager evictions ensure that an empty ERB entry is available anytime a region is

evicted from the RVA. When an empty entry is not available, the cache uses standard back-pressure mechanisms to stall the cache until one becomes available. The ERB concept has been proposed before but without the notion of eager evictions [8]. The ERB can be centralized or, as we assume in this work, banked to mirror the cache's organization.

3) The **Block Status Table (BST)** stores per-block status information. The RVA stores information for many more blocks than are actually present in the cache (typically two to four times as many) and since this status information is only required for blocks that are resident in the cache, storing this information in the BST reduces overall storage requirements. The BST stores LRU information and block status bits (e.g., coherence state). The BST geometry is identical to that of the data array, and it uses the same indexing scheme as shown in Figure 1(c). When comparing this to the RVA indexing scheme in Figure 1(d), note that addresses that map to a specific BST set can map to multiple different RVA sets. To avoid searching multiple RVA sets when performing block replacements, optional BST backpointers contain the RVA index bits that are not contained in the BST index (e.g., bits 21 and 20 in the example shown). All the designs considered in this work, use two-bit backpointers. The BST is an un-tagged structure, and relies on the results of an RVA or ERB lookup to perform the final selection of a single way from the BST.

In principle many different RT organizations are possible. In practice there are a lot fewer RVA entries than blocks. This inevitably restricts data placement and can decrease the hit rate. In practice the RVA is less associative (e.g., 12-way vs. 16-way) and has fewer sets (e.g., 2K vs. 8K) than the data array. Moreover, a single data set maps to two or four RVA sets. This is sufficient for avoiding a performance decrease. The RVA and BST require less area compared to a conventional tag array and since they are smaller, and each access reads and writes fewer bits, they will likely be faster and consume less energy. Finally, the RT is banked to mirror the organization of the conventional cache. Addresses are interleaved across banks at the region level. We found that the performance of conventional, block- and region-interleaved caches was within 0.7% of each other.

2.2. Functional Description

When servicing a cache request, there are multiple different scenarios, as described in Figure 2. The common scenario is a hit for the region and the block. As shown, the access proceeds in parallel to the RVA, ERB and BST. In the event of a block hit, the result of the RVA or ERB lookup determines which way in the BST is selected in the final stage of the BST access. On an RVA hit and block miss, we need to replace a block from the same data set. Since multiple RVA entries may map onto this data set, the BST backpointers determine the RVA set of each block and the BLOFs of that RVA set determine the corresponding RVA entry. This process determines the address of the victim block so it can be evicted, and the two RVA entries (for the requested and victim blocks) are updated to reflect their new states. When a region miss occurs, that is no RVA entry or ERB entry is found with the correct region tag, then a victim RVA entry is selected and copied to the ERB. As a result of this process, the backpointers in the BST for all the blocks in the victim region become inconsistent. These stale pointers do not impair correctness since they are easily detected by lack of a matching BLOF. Once the RVA entry has been copied to the ERB, a new entry can be allocated in its place or the requested region, at this point the access proceeds the same as if it had originally found the newly initialized RVA entry.

When a request hits on an entry in the ERB, the access proceeds the same as for a hit in the RVA. The ERB entry is updated to reflect any newly allocated or evicted blocks. ERB entries can be readily promoted back to the RVA since the entry would necessarily map onto the same RVA set it was evicted from. In this work we disallow these promotions in lieu of a policy to determine when this might be beneficial. Studying potential policies is left for future work. In this work, the ERB must evict all blocks of a region before it can be cached again in the RVA.

Region actions are directly supported by the RVA. A single lookup is sufficient for determining which blocks of a region are cached and where. This information can be used to invalidate (e.g., in preparation for I/O DMA) or to migrate these blocks (e.g., to another processor or to an I/O device). Additional information can be tracked either by introducing additional state per RVA entry or with the unused encodings of the BLOFs. As an additional optimization, it is not necessary to access

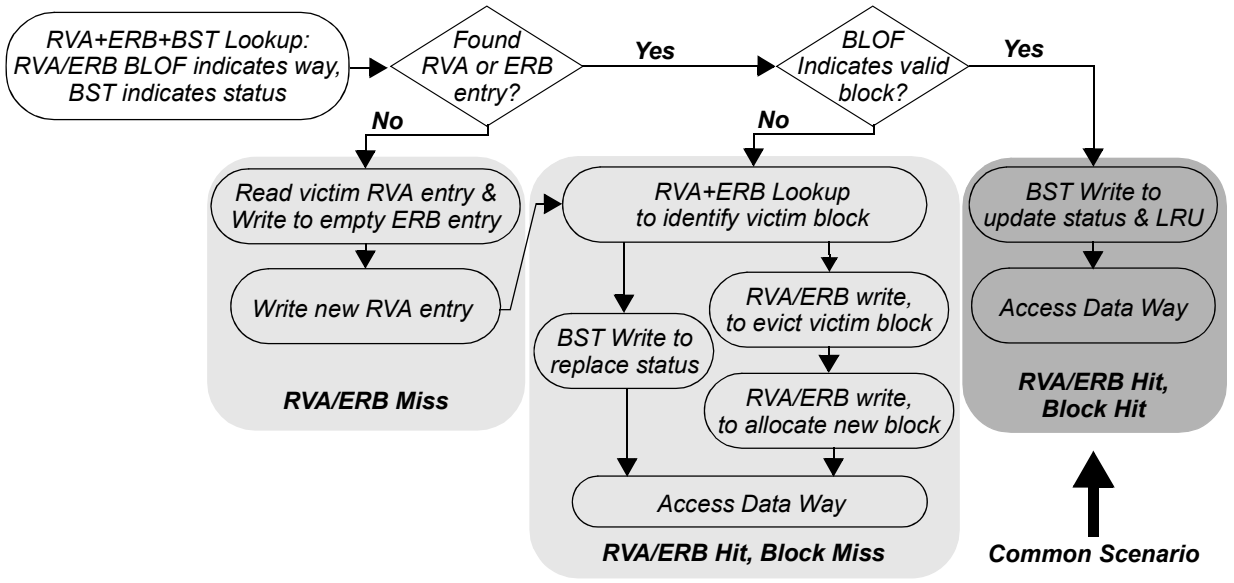

Figure 2. RegionTracker lookup procedure

the BST in parallel with the RVA for every lookup. Since the BLOFs contain a valid bit, the BST is only necessary for accesses requiring additional state information (e.g., dirty, or exclusive). For example, reads only require a valid copy of the data, and do not need to access the BST. If the access misses in the cache, another lookup will be performed when the missing data is ready to be allocated, and the BST lookup can be performed at that time to select a replacement block and to update the BST. By avoiding BST lookups for reads, RegionTracker can reduce average power consumption. This optimization may not always be possible, for example if prefetch bits need to be checked for all accesses.

2.3. Optimization Framework

The main contribution of this work is providing a practical framework for implementing memory hierarchy optimizations that exploit patterns in coarse-grain memory behavior. This section describes two different optimizations that can easily be implemented in a RegionTracker cache.

2.3.1. Snoop Elimination

Recent works [2,12] have presented coarse-grain optimizations which reduce power and bandwidth in snoop-based multi-processors. Both proposals exploit coarse-grain sharing patterns to eliminate unnecessary broadcasts and snoops for non-shared data. RT provides a framework for implementing these optimizations with reduced overhead, and with improved results. In general, these optimizations work as follows: The first block access into a region uses broadcast. All remote nodes report whether they have any blocks from that region cached, and the originating node marks the region as non-shared if there are no remote blocks. Subsequent requests to the region from the same node do not use broadcast and hence to not result in additional snoops. If another node attempts to access a block within the region, it will use broadcast and hence the non-shared state of the region is invalidated.

This optimization requires the following functionality: 1) determining whether any blocks are cached in a region, and 2) tracking which regions are in a non-shared state. The two previous works use additional structures to provide this functionality. Coarse-grain coherence tracking (CGCT) [2] uses a Region Coherence Array (RCA) structure to precisely track which regions have cached blocks and to track the state of each region. In addition to the simple *non-shared* state, the RCA tracks whether any shared copies are possibly in a modified state to further optimize accesses to shared read-only data. This RCA structure maintains precise information, and as a result evicting an entry from the RCA can result in evicting data from the cache to maintain this precision. The RegionScout (RS) [12] approach instead uses imprecise information, trading some accuracy for reduced area overhead. RS uses an

imprecise Cached Region Hash (CRH) structure to track whether a region has any blocks cached in a region, as well as a second Non-Shared Region Table (NSRT) structure to track which regions are not shared by any other processors.

RT can easily support the two functions required for this optimization via a single bit addition per RVA entry to mark non-shared regions. The RVA BLOFs are already capable of determining in a single access precisely which blocks in a region are cached. The additional region states implemented by CGCT can also be supported with more bits in each RVA entry. Similarly to the original RS, the first block access in a region determines whether there are any remotely cached blocks. Remote nodes perform an RVA lookup to determine whether they have any blocks in the region cached. The originating node marks the region as non-shared using the extra RVA bit. Subsequent requests for other blocks in the region from the same node need not use broadcast. If another node requests a block in the region, it will use broadcast invalidating the non-shared status of the region. Section 4.7 compares this approach directly with RS and shows that the RT approach reduces overhead significantly while doubling the effectiveness of RS.

Using RT, *BlockScout*, a new optimization, becomes simple to implement. A single sharing bit is added to each BLOF to indicate whether a specific block is shared or not. In this case, the originating nodes collect a sharing vector on the first block access in the region and avoid broadcasts for accesses to non-shared blocks in the region, even if some other region blocks are shared. This requires communicating sharing vectors in snoop replies, which is reasonable for single chip multiprocessors. Section 4.7 shows that this new BlockScout design eliminates an additional 13% of snoops while still requiring less area overhead than the previous RegionScout design.

2.3.2. Prefetching

Stealth Prefetching [3] tracks which blocks in a given region have been previously fetched and uses this information to later prefetch these blocks. With 1KB regions, this approach improves performance by an average of 20% across a variety of commercial, scientific and multi-programmed workloads, but at the cost of significant area overhead (e.g., 11.6% of total cache area). This technique extends the RCA structure of [2] with the addition of bit-vectors to track which blocks have been fetched and which are present in the cache. The information in these bit-vectors can easily be encoded in the existing RT BLOFs (using the invalid states) and the other information stored in the RCA of [2] only requires adding a few state bits to each RVA entry. Although the RVA and BST structures of RT may be larger than the RCA structure, by *replacing* the tag array RT reduces the total overhead from 11.6% to 7.4% of the overall cache area, assuming 1KB regions and using an RT with the same geometry as the RCA used in [3]. This design provides identical coverage and performance as the RCA used in [3], but it eliminates the complexities associated with evicting RCA entries, it streamlines lookups by using a single cache to track all information, and it reduces the area overhead by 4%. The evaluation of RT in Section 4.4 suggests that a smaller RT structure could be used without significantly decreasing the hit-rate of the cache. To maintain the effectiveness of the prefetching technique, a smaller RCA could be used in conjunction with the RT cache to track regions that are no longer cached. Such a design would again maintain the coverage and performance obtained in [3], it would still eliminate the complexities of evicting RCA entries, and it would further reduce the overhead to just 5.3% of total cache area.

3. Relation to Previous Coarse-Grain Cache Designs

RT builds upon previous dual-grain caches namely the sectored cache (SC) [9], sector pool cache (SPC) [13], and the decoupled-sectored cache (DSC) [14]. These designs focused solely on reducing tag requirements and they do not meet all of the different goals of this work.

3.1. Sector Caches and Sector Pool Caches

SC, SPC and RT all use a *region*[1] *vector array* (RVA) and hence provide precise region-level information. These designs inevitably restrict the mix of blocks that can co-exist in the cache because they use fewer RVA entries than blocks. Figure 3 illustrates the differences amongst SC, SPC and RT showing the relationship amongst RVA entries and

[1] The definition of sectors and regions is the same, however, we opt for the term region to signify that we focus on much larger regions of memory than those considered in previous works.

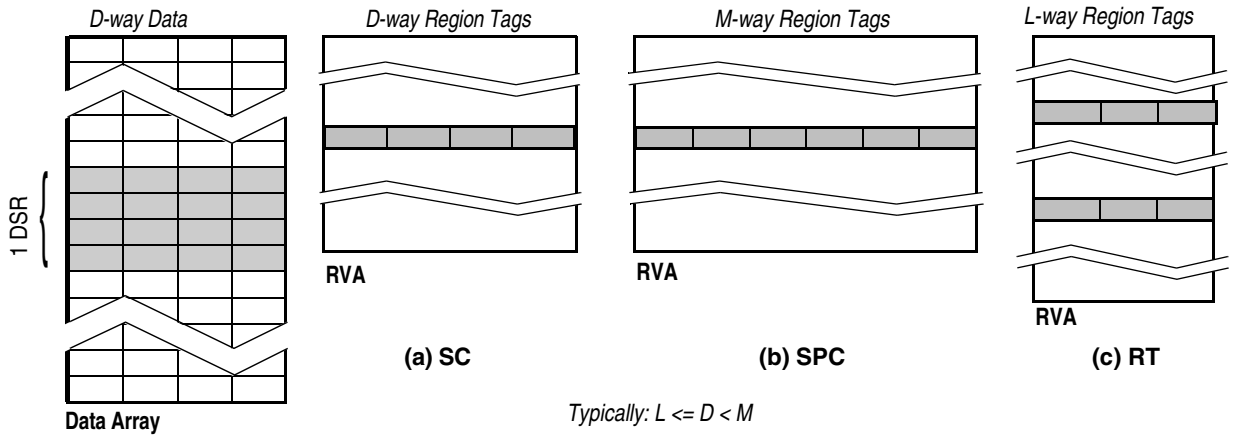

Figure 3. Dual-grain tracking cache designs: (a) sector cache, (b) sector pool cache, (c) RegionTracker.

data array blocks. Assuming a direct indexing scheme, the blocks belonging to a region will all map to a *data set region* (DSR) which is a continuous portion of data array sets. SC is the most restrictive as it allocates a single RVA entry for each data way in each DSR. SPC adds flexibility by having more RVA entries per DSR. In SPC, these additional entries form a single, highly associative set in the RVA. RT offers flexibility similar to SPC, but RT increases the number of RVA sets instead of just increasing associativity. As our experiments show, depending on cache size, SC increases miss rate by up to 142%, while a 32-way to 56-way RVA is necessary for SPC to approach within 1% the hit rate of a 16-way cache.

3.2. Decoupled-Sectored Cache

DSC combines a region tag array (RTA) with a status table (BST). Each BST entry contains a pointer which, when combined with the set index of that entry, uniquely identifies an RTA tag. These pointers are longer than the BST backpointers of RT (e.g., for the 8MB cache considered in Section 2.1, a DSC would require six bits per backpointer) DSC lookups proceed into two phases. The first accesses the RTA, comparing region tags against the address, and, in parallel, it accesses the BST. The second phase compares the BST pointers against the way of the matching region tag.

DSC overcomes the problems of poor miss-rates and high associativity suffered by SC and SPC respectively. However, the original DSC design cannot be used directly for our purposes for two reasons. First, when a region tag is replaced, all BST sets in a DSR must be scanned on-the-spot to evict the corresponding blocks. This must be done irrespective of whether there are any blocks left or not, consuming cache bandwidth and increasing cache latency for concurrent accesses in non-blocking caches. Second, DSC cannot identify with a single access whether a given region is cached or which blocks in a region are cached. The presence of a region tag in DSC indicates that some blocks from that region *may be* cached, but DSC must scan multiple sets to identify which, if any, blocks are cached. While this imprecise information may be sufficient for optimizations such as RS [12], it is not for optimizations such as CGCT [2], prefetching [3,16] and sector evictions.

This paper discusses two extensions to DSC. The first smooths out region evictions (please see the Acknowledgments section). The second extension introduces per-region block information.

3.2.1. Smoothing Out Evictions

The *optimized DSC* (oDSC) smooths out region evictions via the use of a modified ERB. In the original DSC design a region tag cannot be evicted without first invalidating all matching BST entries. By adding a *version tag* to each BST entry and to each region tag, it becomes possible to evict a region tag to the ERB while deferring the invalidation of all matching BST entries. The ERB can then scan and evict the region's blocks in the background.

When replacing a region tag, its original location and version tag are copied along with the tag itself into an ERB entry. A new region tag must use a different version tag value to prevent a match against the BST entries belonging to the evicted region. Eager evictions can now be performed by scanning the BST for blocks belonging to the tags

in the ERB. Version tags can be recycled once a region is evicted from the ERB. If no version tags are available, a new region must wait until one becomes available. oDSC still needs to scan for *all* blocks within an evicted region, cached or not, and hence still consumes the same bandwidth as the original DSC.

3.2.2. Precise Dual-Grain Tracking

The second extension provides some of the missing coarse-grain tracking functionality. The *RegionTracker-DSC,* or RT-DSC extends oDSC by adding single-bit BLOFs to each region tag. RT-DSC can now easily identify which, if any, blocks are cached for a given region. However, the RT design described in Section 2 still provides additional functionality. The RT BLOFs identify the precise location of each block in the data array. This allows RT to avoid BST accesses on reads, and to overlap some BST and data array accesses. Provided that the application has sufficient spatial locality, a single RT access can be used to service many subsequent cache requests as it reveals the location of all blocks within the region. Furthermore, RT accesses do not require the two phase lookup of DSC.

4. Experimental Analysis

This section is organized as follows: Section 4.1 describes the simulation methodology. Section 4.2 compares the miss rates of RegionTracker and traditional sector caches. Section 4.3 demonstrates that RegionTracker can be implemented in the same area required for a conventional tag array. Section 4.4 compares the area and miss rate trade-offs of various cache designs. Section 4.5 demonstrates that RegionTracker does not hurt performance while Section 4.6 shows that RegionTracker reduces energy compared to a conventional tag array. Finally, Section 4.7 demonstrates how RegionTracker performs when implementing a snoop elimination policy.

4.1. Methodology

We simulated a four-core CMP with a shared L2 cache based on the Piranha cache design [1] using the *Flexus* simulator [7]. Table 1 details the processor cores. Table 2 describes the simulated workloads. These include: (1) The TPC-C v.3.0 online transaction processing workload running on both *IBM DB2 v8 ESE* and *Oracle 10g Enterprise Database Server*. (2) Five queries from the TPC-H DSS workload running on *IBM DB2 v8 ESE*. (3) The SPECweb99 benchmark running over *Apache HTTP Server v2.0* and *Zeus Web Server v4.3*. The web servers were driven with separate simulated client systems, but client activity is not included.

Table 1. Base processor configuration

Branch Predictor	**Fetch Unit**
8k GShare +16K bi-modal + 16K selector 2k entry, 16-way BTB, 2 branches per cycle	Up to 16 instr. per cycle 64-entry Fetch Buffer
	Scheduler
	256-entry/64-entry LSQ
ISA & Pipeline	**Issue/Decode/Commit**
UltraSPARC III ISA, 4GHz, 8-stage pipeline, out-of-order execution	any 8 instr./cycle
	Main Memory
	3 GB, 300 cycles
L1D/L1I	**UL2**
32kB 4-way set-associative 64B blocks, LRU replacement 2 cycle latency	8-256MB, 16-way set-associative, 8 banks, 64B blocks, LRU replacement, 6/18 cycle tag/data latency

Table 2. Workload Descriptions

Online Transaction Processing (TPC-C) (OLTP)	
Oracle	100 warehouses (10GB), 16 clients, 1.4 GB SGA
DB2	100 warehouses (10GB), 64 clients, 450 MB buffer pool
Decision Support (TPC-H on DB2) (DSS)	
Qry 1	Scan-dominated, 450 MB buffer pool
Qry 2	Join-dominated, 450 MB buffer pool
Qry 6	Scan-dominated, 450 MB buffer pool
Qry 16	Join-dominated, 450 MB buffer pool
Qry 17	Balanced scan-join, 450 MB buffer pool
Web Server (WEB)	
Apache	16K connections, FastCGI, worker threading model
Zeus	16K connections, FastCGI

Performance simulations use the SMARTS sampling methodology [18]. Each sample measurement involves 100K cycles of detailed warming followed by 50K cycles of measurement collection. Results are reported with errors with a 95% confidence interval. We used matched-pair sampling to measure change in performance [5]. Performance is measured as the aggregate number of user instructions committed each cycle [17]. Miss rates are measured using functional simulation of two billion cycles, with each core executing one instruction per cycle. We take measurements only for the second billion. Two of the workloads complete in less than two billion cycles; therefore, for TPC-H queries 2 and 17, after warming up for one billion cycles, we take measurements for 209 million and 259 million cycles respectively. The base L2 uses block interleaving.

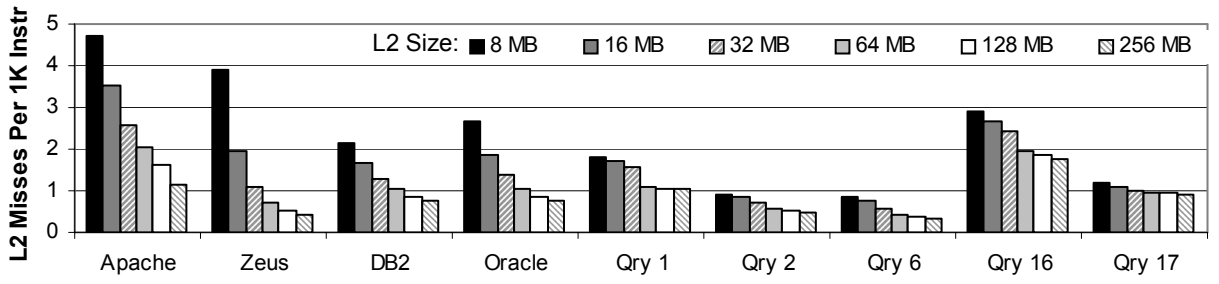

Figure 4. Misses Per 1K Instructions for conventional cache with size from 8MB to 256MB

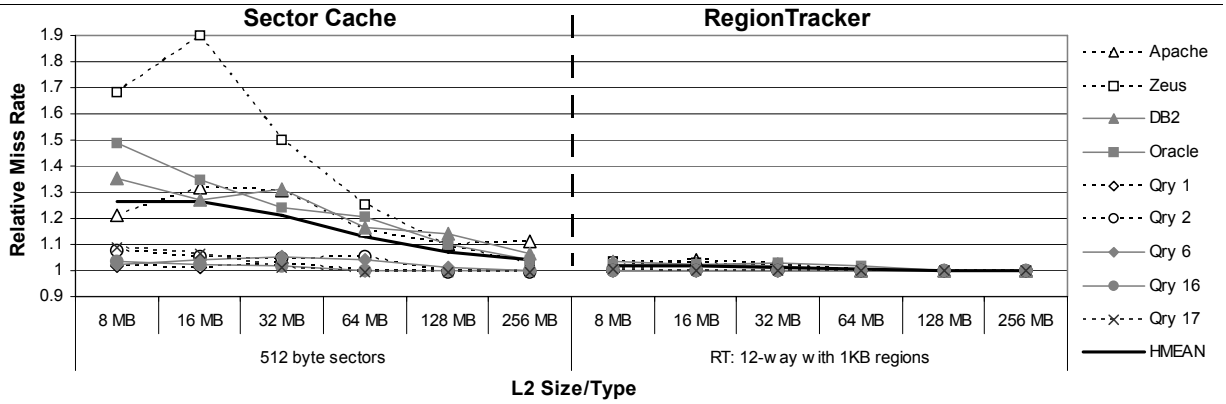

Figure 5. Relative Miss Rate for sector cache with 512-byte sectors (left) and a RegionTracker cache with 1kB regions (right), with capacities of 8MB to 256MB.

Figure 4 presents the misses per 1K instructions (MPKI) for all workloads with L2 cache sizes from 8MB to 256MB. Most of these workloads show continuing benefits from increasing cache sizes all the way to 256MB, but still exhibit considerable miss rates even for the larger caches. Although such large L2 caches may never be practical, L3 caches with these capacities will soon become practical. For clarity, the remaining results report miss rates relative to the results shown in Figure 4. When results are averaged across all workloads, we first calculate the harmonic mean of each workload class (e.g., web, oltp, dss), and then find the harmonic mean of these three means. This avoids over representation of the DSS workloads in the aggregate results. We restrict our attention to 16-way set-associative caches since we found that reducing associativity further impacts performance considerably. For example, reducing its associativity from 16-way to 15-way increases average miss rate by 2.6% for an 8MB cache.

4.2. RegionTracker vs. Sector Cache

First, we compare the miss rate of a traditional sector cache with the RegionTracker design. Figure 5 shows the relative miss rates of SC (left) and RT (right). While the RT design uses 1KB regions, we show results for an SC design with 512-byte sectors because SC performs much worse with 1KB sectors (e.g., as much as an additional 50% increase in miss-rate for some benchmarks). The RT design chosen for Figure 5 has 2K to 64K, 12-way set-associative RVA sets, depending on the cache size. The data array for both the SC and RT designs is 16-way set-associative. We chose this design to demonstrate that RT performs well even when its RVA associativity is lower than the original cache's associativity. The *y*-axis shows miss rate relative to a conventional cache with equal capacity. For each cache, the *x*-axis shows cache capacities from 8MB to 256MB. There is a separate curve for each workload, with a final, thick solid curve showing the harmonic mean of the three workload types.

As shown in Figure 5, sector caches can result in miss rate increases of up to 90% with 512-byte sectors. Note that increasing cache size improves

the relative performance of SC for most cases, but not all. Taking Zeus for example, the relative miss rate goes from 1.68 for an 8MB SC, to 1.90 for a 16MB SC. Figure 4 shows a steep decline in absolute miss rate for Zeus when going from an 8MB to a 16MB L2. Thus, the increase in relative miss rate for the SC can be explained by a particular working-set being able to fit into the 16MB conventional cache, but not fitting into a 16MB SC. This implies that a portion of the working set contains many sparsely populated sectors.

Looking at the different types of workloads, the WEB and OLTP workloads perform much worse than the DSS workloads. The harmonic mean increases for the WEB and OLTP workloads go from 41% and 42% respectively for an 8MB SC, to 7% and 5.4% respectively for the 256MB SC. The DSS workloads, on the other hand, only increase miss rate by 4.7% for an 8MB SC, and for the large 256MB SC, they perform identically using either a normal cache or SC. Thus, sector caches result in significant miss rate increases for WEB and OLT workloads.

When comparing SC to RT, RT demonstrates similar trends, but the relative miss rates are much lower. In fact, for the 128MB and 256MB caches, RT has the exact same miss-rate as a conventional cache. Across all cache sizes and all workloads, the RT design never increases the miss rate by more than 3.4%, which occurs for DB2 at 8MB.

4.3. Area Estimation

We estimated the relative size from full-custom implementations of the cache structures on a 130nm commercial technology. This is the best commercial technology that was available to us at the time of this writing. While commercial designs use highly-optimized special-purpose SRAM designs, these should primarily affect only absolute area. We believe that our methodology is sufficiently accurate for comparing the relative area of the various designs.

Table 3 shows the area occupied by various lookup structure designs. The columns show the type of cache used, the number of Kbits of storage required, the actual size of the tag array, the relative size compared to the conventional tag array, and the *relative bit-density* (where bit density is the number of bits stored in $1mm^2$) compared to the conventional tag array. The rows in Table 3 show, in order, a conventional tag array, an SC tag array with 512-byte sectors, a 52-way set-associative SPC tag array with 1kB sectors, an oDSC tag array and BST, an RT-DSC RVA and BST, and an RT RVA and BST. The last three designs all have 2K, 15-way associative sets with 1KB regions.

Table 3. Storage and area requirements for 8MB tag arrays.

Design	KBits	Size (mm^2)	Relative Size	Relative Density
Tag Array	4352	37.6	1.0	1.0
SC	880	9.4	0.25	0.81
SPC	3718	36.8	0.98	0.87
oDSC	2180	19.8	0.53	0.95
RT-DSC	2660	24.3	0.64	0.95
RT	3940	36.3	0.97	0.94

We show the largest RT design that is smaller than a conventional tag array, a 16-way set-associative RT results in a tag array area increase of 2%, which represents an increase of 0.1% in total cache area. Our performance results demonstrate that lower RVA associativities are needed in practice and hence the area of RT would compare even more favorably. All designs use an 8MB, 16-way set-associative data array. The number of bits required for each structure was calculated assuming 50-bit addresses, three state bits per block, and 64-byte blocks. The RT and RT-DSC designs could use only two state bits, but this would complicate the process of selecting invalid blocks for replacement. The Itanium 2 is an example of a modern processor that uses 50 bits for physical addresses [11]. The measurements do not include ERB area since this is a very small structure that is compatible with all caches shown except for the conventional one. Additional overheads, such as LRU information and ECC checksums where not included in these estimates but these apply to all structures equally and often represent a small fraction of overall area.

The results show that the SC and SPC designs are the least area efficient (i.e., have the lowest bit-density). This results mostly from having similar comparator and sense amp overhead combined with a much smaller number of bit cells. Although the RT design is smaller than the tag array, it is slightly less efficient. Our implementation uses more sense-amplifiers and more routing than a conventional cache to support two types of read operations: either read all the BLOFs for a particular block for all the regions in a given set, or read all the BLOFs for a single region. The sense-amps occupy roughly 5.2% of the total area in the RT design compared to 1.6% in the conventional tag array. While RT uses more

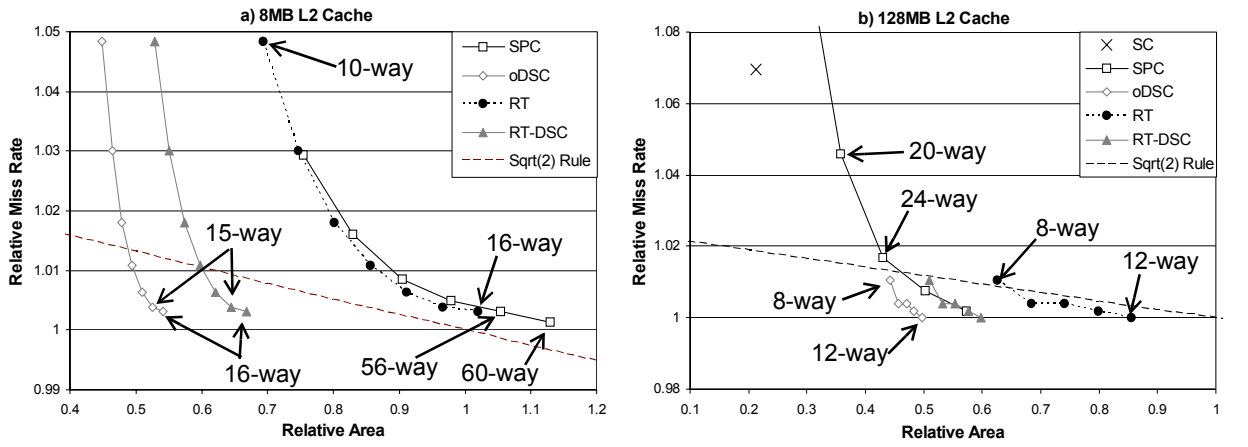

Figure 6. Relative miss rate vs. tag area. a) 8MB caches, and b) 128MB caches.

sense amplifiers, for any given access it activates fewer sense amplifiers compared to the conventional tag array.

4.4. Relative Size and Miss Rate

Figure 6 reports the *relative* miss rate for SC, SPC, oDSC, RT and RT-DSC designs as a function of the relative area of each design and the RVA associativity. The *x*-axis shows the relative area of the structure compared to the area of a conventional tag array. The *y*-axis shows the miss rate relative to the conventional cache. Figure 6a shows results for an 8MB cache, and Figure 6b shows results for a 128MB cache (we omit results for other sizes due to space limitations). The annotations indicate the actual associativities used for a few of the designs, the associativities of the remaining points can easily be inferred as the difference in associativity of consecutive points is constant for each curve. The oDSC, RT-DSC, and RT designs all use the same set of associativities, and these designs all have 2K RVA sets for the 8MB cache and 32K RVA sets for the 128MB cache. The SC design shown in Figure 6b uses 512-byte sectors. For clarity, Figure 6a omits all SC designs, but we note that these designs would be above and to the left of the limits shown. Finally, the two lines labeled "Sqrt(2) Rule" indicate the extrapolated trend of miss-rate vs. cache size. Specifically, the lines represent a cache whose miss-rate is $M = M_0 A^{-0.4}$, where A is cache area, and -0.4 and M_0 are empirically determined constants based on the miss rates of our workloads for cache sizes from 8MB to 128MB. Points beneath this line perform better than a conventional cache with the same total area.

As Figure 6a shows, the 14-way and 15-way RT designs increase the miss-rate less than 1% compared to conventional caches and require 3-9% less area. For the 128MB caches shown in Figure 6b, even the 8-way RT design increases the miss-rate by less than 1%, and the 12-way RT designs achieves miss-rates identical to the conventional cache. These RT designs require 14-37% less area than a conventional tag array. While SPC provides designs with comparable miss rate and area trade-offs, they require at least a 28-way set-associative RVA for the 128MB cache, and at least a 52-way set-associative RVA for the 8MB cache. The RT-DSC design provides an additional area savings of 12% to 35% compared to RT, but at the cost of reduced functionality and increased complexity. For completeness we show the oDSC design as well.

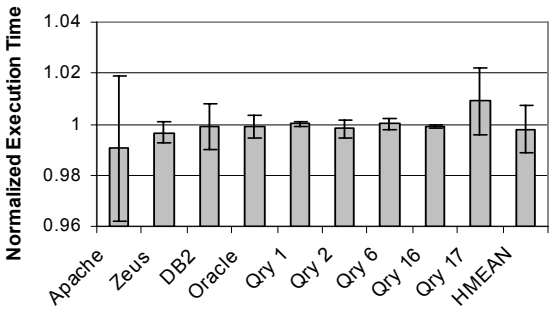

Figure 7. Slowdown for 8MB RT Cache with 2Kx12way RVA sets.

4.5. RegionTracker Performance

Figure 7 shows the percent slowdown of an 8MB RegionTracker cache for each workload, as

well as the harmonic mean across the three types of workloads. The error bars indicate the 95% confidence interval. The RT design used has 2K, 12-way set-associative RVA sets. RegionTracker has a negligible performance impact, with an overall slowdown of 0.2% ± 1.0%. Apache suffers the largest slowdown of 0.95% ± 2.9%, and TPC-H Query 17 actually sees a speedup of 0.9% ± 1.3%. When the error is considered, any performance differences between RT and a conventional cache are statistically insignificant.

4.6. RegionTracker Energy

Figure 8 shows the percent reduction in total energy consumed by tag array lookups for RT compared to a conventional 8MB cache. The error bars indicate a 95% confidence interval. The RegionTracker design uses 2K, 12-way set-associative RVA sets, with 1KB regions. All workloads see a reduction close to the average of 33% ±6 %.

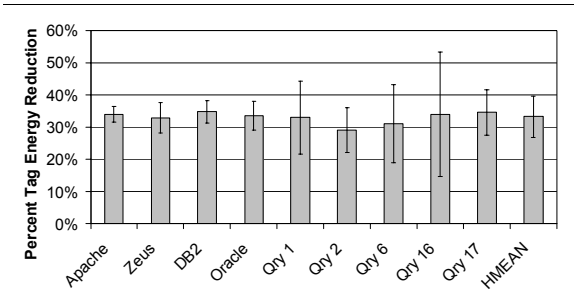

Figure 8. Percent reduction in tag energy for 8MB RT Cache with 2Kx12way RVA sets.

4.7. Snoop Broadcast Elimination

This section evaluates the implementation of the RegionScout technique [12] under the RT framework. Figure 9 shows the reduction in coherence traffic for a 4-way CMP with 512KB private L2 caches. Results are averaged across all the workloads described in Section 4.1. The first three bars show conventional RegionScout implementations with 4K, 8K and 16K CRH tables and 8KB regions [12] (using smaller regions reduces benefits). The last, black bar shows reduction with an RT implementation that uses 1KB regions. The RT implementation reduces traffic even further with less resources because it uses precise sharing information. Per node, the original RS requires approximately 178.3Kbits while the RT requires just 2K non-shared bits. Assuming the same relative bit-density as shown in Table 3, the RT implementation requires 5% *less* area than a conventional tag array.

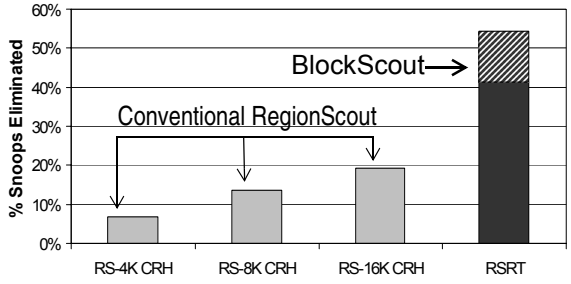

Figure 9. Coherence traffic reduction with original RegionScout (RS), RegionTracker-embedded RegionScout (RSRT), and RegionTracker-embedded BlockScout.

The top, stripped portion of the last bar in Figure 9 shows the additional improvement obtained with the BlockScout optimization described in Section 2.3.1. For 512KB L2 with 1KB regions, the BlockScout optimization requires an additional 32Kbits of overhead, which results in a total overhead that is still 144.3Kbits less than the size of the original RS. This BlockScout implementation only increases tag array area by 5.5%, which represents an increase of only 0.4% in total cache area. Using this optimization results in an additional 13% reduction in broadcasts compared to the RT implementation of RS.

5. Conclusion

Several recently proposed techniques rely on coarse-grain memory information for improving various aspects of memory hierarchy performance and power. As caches grow larger, we anticipate that such coarse-grain optimization techniques will play an increasingly important role. Accordingly, this work set a goal of developing a framework, in the form of a new cache tag array design, that readily exposes coarse-grain information while maintaining the flexibility and benefits of fine-grain block management and communication. It proposed RegionTracker, which offers virtually identical miss rates and performance compared to a conventional cache, while reducing resource requirements. Compared to previous dual-grain tracking cache designs RegionTracker offers the following advantages: 1) It avoids the significant miss rate increase suffered by sectored caches; and 2) it avoids the need for a highly associative RVA

lookups that are required by the sector pool design. Additionally, this paper described a variation of the RT design that extends the decoupled sector cache to provide some of the functionality of our base RT.

For an 8MB cache, RT reduces area by 3-9% and uses lower associativity for lookups, while increasing miss-rate by less than 1% with only a statistically insignificant difference in performance. In addition, using RT to implement an existing snoop broadcast elimination technique doubles the effectiveness of the technique while completely eliminating any area overhead, and in fact reducing tag array area by 5.5% compared to a conventional cache.

We believe that as caches grow larger, coarse-grain management and optimization techniques will become increasingly important. We believe that such optimizations will be key in optimizing coherence, communication, data placement and management for large on-chip memory hierarchies. RegionTracker has the potential of becoming the key building block for such techniques, alleviating the need for many auxiliary structures and facilitating their adoption while reducing costs and complexity.

Acknowledgements

We would like to thank the anonymous reviewers for their insightful comments. We thank André Seznec for suggesting the oDSC design and for his insightful comments on the final version of this paper. Ioana Burcea has patiently read and commented on earlier versions of this paper. Jason Zebchuk is partially supported by an NSERC Postgraduate Scholarship. This work is supported by an Intel Research Council grant, an NSERC Discovery Grant, an equipment donation from the Intel Corporation, and a Canada Foundation for Innovations New Opportunities equipment grant.

References

[1] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. *Piranha: a scalable architecture based on single-chip multiprocessing*. Proc. Int'l Symposium on Computer Architecture, June 2000.

[2] J. Cantin, M. Lipasti, and J. Smith. *Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking*. In Proc. Int'l Symposium on Computer Architecture, June 2005.

[3] J. Cantin, M. Lipasti, and J. Smith. *Stealth Prefetching*. In Proc. Int'l Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 2006.

[4] M. Ekman, F. Dahlgren, and P. Stenström. *TLB and Snoop Energy-Reduction using Virtual Caches in Low Power Chip-Multiprocessors.* Proc. ACM Int'l Symp. on Low Power Electronics and Design, August 2002.

[5] M. Ekman and P. Stenström. *Enhancing multiprocessor architecture simulation speed using matched-pair comparison.* Proc. Int'l Symposium on the Performance Analysis of Systems and Software, March 2005.

[6] K. Gharachorloo, A. Gupta, and J. Hennessy. *Two techniques to enhance the performance of memory consistency models*. In Proc. Int'l Conference on Parallel Processing, Aug. 1991.

[7] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. *SimFlex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture*. SIGMETRICS Performance Evaluation Review, Apr. 2004.

[8] C. Lai and S-L Lu. *Efficient Victim Mechanism on Sector Cache Organization*. Advances in Computer Systems Architecture, Lecture Notes in Computer Science 3189: 16-29, 2004.

[9] J. S. Liptay. *Structural Aspects of the System/360 Model 85 Part II: The Cache*. IBM Systems Journal, 7:15-21, 1968.

[10] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, D. A. Woods. *Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared Memory Multiprocessors.* In Proc. Int'l Symposium on Computer Architecture, June 2003.

[11] C. McNairy, D. Soltis. *Itanium 2 Processor Microarchitecture.* IEEE Micro, 23(2): 44-55, March 2003.

[12] A. Moshovos. *RegionScout: Exploiting Coarse-Grain Sharing in Snoop Coherence.* Proc. Int'l Symposium on Computer Architecture, June 2005.

[13] J. B. Rothman and A. J. Smith. *The Pool of Subsectors Cache Design*. In Proc. Int'l Conference on Supercomputing, June 1999.

[14] A. Seznec. *Decoupled sectored caches: Conciliating low tag implementation cost and low miss ratio*. In Proc. Intl' Symposium on Computer Architecture, June 1994.

[15] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickenmeyer, J. B. Joyner. *POWER5 System Microarchitecture*. IBM Journal of Research and Development, 49(4): 505-522, 2005.

[16] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, A. Moshovos. *Spatial Memory Streaming*. In Proc. Intl' Symposium on Computer Architecture, June 2006.

[17] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, B. Falsafi. *Temporal streaming of shared memory.* In Proc. Intl' Symposium on Computer Architecture, June 2005.

[18] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, J. C. Hoe. *SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling*. In Proc. Intl' Symposium on Computer Architecture, June 2003