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

Jason Zebchuk and Andreas Moshovos
Electrical and Computer Engineering
University of Toronto

Abstract

Current on-chip block-centric memory hierarchies exploit access patterns at the fine-grain scale of small blocks. Several recently proposed memory hierarchy enhancements for coherence traffic reduction and prefetching suggest that additional useful patterns emerge with a macroscopic, coarse-grain view. This paper presents RegionTracker, a dual-grain, on-chip cache design that exposes coarse-grain behavior while maintaining block-level communication. RegionTracker eliminates the extraneous, often imprecise coarse-grain tracking structures of previous proposals. It 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 and commercial workloads, we demonstrate that RegionTracker overcomes the inefficiencies of previous coarse-grain cache designs. We also demonstrate how RegionTracker boosts the benefits and reduces the cost of a previously proposed snoop reduction technique.

I. INTRODUCTION

Future on-chip caches will most likely grow to several tens of megabytes to compensate for limited off-chip bandwidth and large application footprints, and to meet the demands of multiprocessing and multithreading. This unprecedented on-chip storage offers unique opportunities for new 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* and managing the corresponding blocks, becomes increasingly appealing as 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 cache hierarchies. Several works corroborate this observation. Region information has been shown to facilitate: (1) performance, bandwidth and power improvements for snoop-coherent shared memory multi-processors [1,9], and (2) prefetching for applications with demanding memory footprints [2,13]. These techniques rely on two types of information: 1) whether any block in a region is cached [1,9], and 2) which specific blocks of a region are cached [2,13].

Since region information is not readily available in existing caches, previous work relied on separate structures to track and manage it. These structures are imprecise [9], incomplete [13], or restrict data placement [1,2]. Moreover, their relative cost can be as high as an additional area of 60% compared to a conventional tag array [2]. Commercial designs are more likely to incorporate these optimizations if they have lower complexity and area cost. Wo observe that these optimizations require much of the same, or similar, functionality. Accordingly, we present RegionTracker (RT), a building block for such optimizations that reduces the overhead and eliminates the imprecision of these extraneous tracking structures. As an example of RT's potential we domesticate that it improves the performance and reduces the cost of a snoop-reduction method.

RT introduces region-level functionality without compromising performance, power or area compared to a conventional cache. A single lookup in RT is sufficient to determine *which, if any* blocks of a region are cached and *where*. Moreover, region-level management such as region invalidation, migration, and replacement are naturally supported; although RT still uses fine-grain block communication to avoid a bandwidth explosion. Accordingly, RT is a *dual-grain* cache design. The RT design methodology starts with a conventional cache and replaces the tag array with a slightly smaller structure that facilitates region-level lookups and management without changing overall performance. RT builds upon previous sector cache designs, improving their performance and adding functionality. Compared to previous designs, RT offers simple block and region lookups and replacements, without requiring higher associativity or hurting performance, latency, complexity or area. Sections II and III explain the key challenges in building a dual-grain cache, how RT meets these challenges, and how it compares to previous designs.

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. Second, it presents a framework for incorporating region-level optimizations in the on-chip hierarchy "for free", that is without requiring more area or hurting performance or power.

II. REQUIREMENTS

Our starting point is a conventional cache whose performance and area have been tuned appropriately. Our goal is to replace only the tag array so that we can inspect and manipulate regions comprising several blocks without impacting performance. Ideally, compared to a

Manuscript submitted: 08-May-2007. Manuscript accepted: 31-May-2007. Final manuscript received: 07-June-2007.

conventional cache, the new architecture should meet the following requirements: 1) Communication uses fine-grain blocks. 2) Miss rate does not increase. 3) Cache latency does not increase. 4) Cache area does not increase. 5) Lookup associativity does not increase. 6) There is no need for additional cache accesses as a result of regular cache operation (e.g., for replacements). 7) Banking and interleaving is possible.

In addition, the new architecture should provide the following region-level functionality: 1) A single lookup can determine if a region is cached. 2) A single lookup can determine which blocks of a region are cached. 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., [8].

III. RegionTracker Design

For clarity, in this section, we assume a 64MB, 16-way set-associative L2 cache with 64 byte blocks, 50-bit physical addresses, and 1KB regions. As Figure 1(a) shows, RT comprises three components. 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 a number of block information fields (BLOFs) that identify in which data way each block is cached. if any (four bits for a 16-way set-associative array plus an additional valid bit). 2) Evicted RVA entries are copied into the **Evicted Region Buffer (ERB)**, eliminating the need for multiple simultaneous block evictions. ERB entries contain the same information as RVA entries. The small ERB (e.g., 12 entries) does not contain any data blocks, and, thus, it is not a victim buffer. This concept has been proposed before [6]. In addition, the ERB evicts blocks eagerly from the oldest one third of its entries. 3) To reduce overall storage requirements, per-block status information (e.g., coherence state) is stored in the **Block Status Table (BST)** organized with the same geometry as the data array. The BST stores LRU information for each set, and for each block the BST contains three bits for coherence state and *optionally* backpointers to RVA sets. Without backpointers, multiple RVA sets (in practice four) are searched when a block is evicted.

In principle many different organizations are possible. In practice there are a lot fewer RVA entries than blocks. This inevitably restricts data placement and can hurt the hit rate. In practice, the RVA is less associative (e.g., 12-way vs. 16-way) and has fewer sets (e.g., 16K vs. 64K) than the data array, but without significantly decreasing hit rate. Figure 1(c) and Figure 1(d) show the indexing schemes for the data array and the RVA, respectively. The number of RVA sets that a data set maps to depends

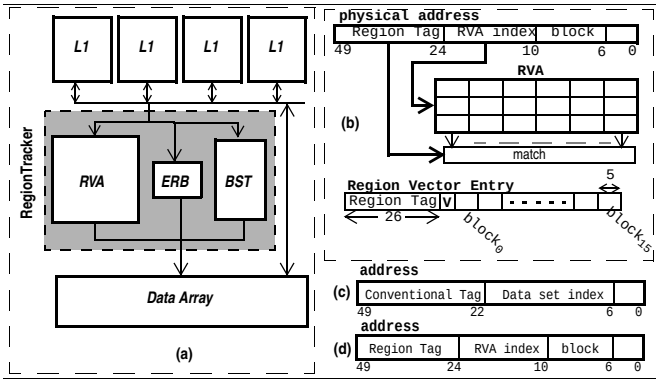

Figure 1: *(a) Block diagram of a RegionTracker L2 cache design. (b) Region Vector Array. (c) Data array indexing. (d) RVA indexing.*

on the number of non-overlapping bits in the RVA and data set indices as shown. In this example, the two non-overlapping bits (i.e., 22 and 23) indicate that each data set maps to four RVA sets. These bits are stored as the backpointer in the BST.

RT requires less area compared to a conventional tag array. Since the structures are smaller, and each access reads and writes fewer bits, they will likely be faster and consume less energy. As a result, cache latency does not increase, and might decrease. 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.

Table 1. RegionTracker Operation

Region	*Block*	*Actions*
Hit	*Hit*	*1. Check RVA + BST in parallel, BLOF indicates hit and which data way, BST provides the status* *2. Access the corresponding data way*
Hit	*Miss*	*1. Check RVA + BST in parallel, matching RVA entry R contains BLOF indicating miss* *2. Use BST LRU to select replacement block and its backpointer to locate RVA entry R'* *3. Update R' and evict the block* *4. Update R and place new block*
Miss	*Miss*	*1. Check RVA + BST in parallel, no RVA match* *2. Use RVA LRU to select entry R' to evict* *3. Copy evicted R' to ERB* *4. Allocate new RVA entry R and continue to (Hit, Miss) scenario*
Miss	*Hit*	*Not possible*
ERB	***Block***	***Actions***
Hit	*Hit*	*Same as the (Region hit, Block hit) case above.*
Hit	*Miss*	*Bring block into cache and update the ERB, do not promote entry into the RVA (see text)*

A. Functional Description

Table 1 shows the various block access scenarios. The common scenario is a hit for the region and the block. As shown, the access proceeds in parallel to the RVA and

BST. In addition the ERB is probed for matches. 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 holds optional backpointers to avoid searching multiple sets. These backpointers become stale the moment an RVA entry is evicted into the ERB. Stale pointers are easily detected by lack of a matching BLOF. We disallow promoting ERB entries to the RVA, but promotions would not introduce any problems since the RVA entry would necessarily be in the same RVA set pointed to by any existing backpointers. In our simulations, regions moved to the ERB can only be returned to the RVA after all blocks have been evicted.

B. Previous Coarse-Grain Cache Designs

RT builds upon previous dual-grain caches, namely the sectored cache (SC) [7], the sector pool cache (SPC) [10], and decoupled-sectored cache (DSC) [11]. These designs focused solely on reducing tag requirements. As a result they do not meet the different goals of this work.

DSC offers the best hit rate vs. resource trade-off, but does not meet the region functionality required for this work. DSC combines a region tag array with a status table (BST). Each BST entry contains a pointer which, when combined with the set index, uniquely identifies a tag in the region tag array. Cache lookups first locate the correct tag corresponding to the requested block, and then compare multiple BST pointers to find one that points to the correct tag. In the DSC design, a region tag suggests that blocks *may be* cached, but finding which blocks are cached requires scanning multiple sets. This functionality is required for prefetching [2,13] and for sector evictions. Like DSC, RT relies on the decoupled organization of the region tags and the data array to avoid a hit rate decrease. As a result, they both achieve similar miss-rates. However, RT encodes information differently. RT uses shorter BST pointers than DSC and instead relies on information stored in the BLOFs. While this results in a larger overall area, it provides additional functionality. A single RVA access in RT can determine the precise location of all data blocks in a region. This information can be used to avoid some BST accesses (e.g., for read accesses), or to simplify reading out all blocks in a region (e.g., for migrating a region to another cache).

SC, SPC and RT all use a *region 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 2 illustrates the differences amongst SC, SPC and RT, showing the relationship between RVA entries and data array blocks. Assuming a direct indexing scheme, the blocks belonging to a region 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 per DSR and per data way. SPC adds flexibility by having more RVAs per DSR. RT offers much of the flexibility of SPC by increasing instead the number of RVA sets. Our experiments show, depending on cache size, SC increases miss rate by 4% to 26% on average, and up to 146% in the worst case, while a 32-way to 44-way RVA is necessary for SPC to approach within 1% the hit rate of a 16-way cache.

Finally, RT incorporates mechanisms to smooth RVA evictions and to maintain low associativity lookups for block replacements.

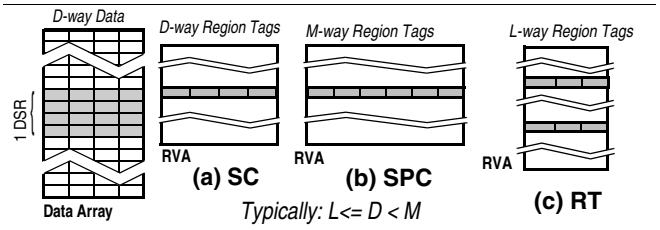

Figure 2: *Dual-grain tracking cache designs. (a) Sectored cache. (b) Sector pool cache. (c) RegionTracker.*

IV. Experimental Analysis

We simulated a four-core CMP with a shared L2 cache using the *Flexus* simulator [5]. Table 2 details the processor cores. We used the commercial workloads detailed in [13]. Performance simulations use the SMARTS sampling methodology [15]. Each sample measurement involves 100K cycles of detailed warming followed by 50K cycles of measurement. Performance results were measured using matched-pair sampling [3] and reported errors are for a 95% confidence interval. Performance is measured as the aggregate number of user instructions committed each cycle [14]. Miss-rates are measured using functional simulation of one billion instructions on each core after warming each core for one billion instructions. The base L2 uses block interleaving.

Table 2. Base processor configuration

Branch Predictor	**Fetch Unit**
8k GShare +16K bi-modal + 16K selector 2k entry, 16 set-associative 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	8-64MB, 16-way set-associative, 8 banks, 64B blocks, LRU replacement, 6/18 cycle tag/data latency

Figure 3 reports the *relative* increase in miss rate for RT, SC, SPC, and DSC as a function of relative area and RVA associativity. The data capacity of all caches

remains constant at 64MB (we observed similar trends for caches in the range of 8MB to 256MB). For SPC, DSC and RT we use 1KB regions. The *x*-axis shows relative size with respect to a conventional tag array. The "Sqrt(2) Rule" line shows the extrapolated average trend of miss rate increase vs. cache area. Designs that fall under the line perform better than a conventional cache of the corresponding size. We estimated the relative size from full-custom implementations on a 130nm commercial technology.

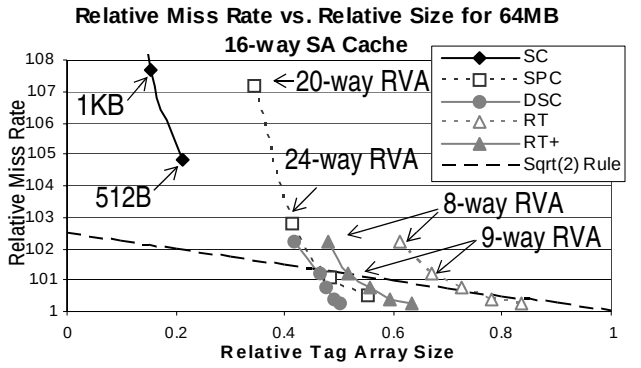

Figure 3: *Relative miss rate vs. tag size for an 64MB L2 cache.*

While the SC requires the fewest possible resources, it increases miss rate by 5% even with 512B regions. The SPC needs at least a 32-way set associative RVA to achieve a miss rate that is within 1% of that of the conventional cache. The "RT+" design omits the BST backpointers and thus requires up to four set accesses on block evictions. A 10-way set-associative RVA is sufficient for RT or RT+ to achieve a miss rate that is within 1% of that of the conventional cache. We measured performance with the 12-way RT for an 8MB L2 cache and found that it remains virtually unchanged with a maximum slowdown of 0.8% ±2.7 % for Apache. All differences are within the measured error.

To illustrate the potential of RT as a building block we show that it can implement the RegionScout snoop broadcast elimination technique [9] with less resources and higher benefits. The RT-based implementation works as follows: The first access into a region uses broadcast. All remote nodes inspect their RVAs and report whether there are any blocks from that region cached. If no nodes have any cached blocks, then the originating node marks the region as non-shared using an extra *non-shared* bit in the RVA entry. Subsequent requests to the region from the same node do not use broadcasts. If another node accesses a block within the region, it uses broadcast and the non-shared bit is invalidated. Figure 4 shows the reduction in coherence traffic for a 4-core CMP with 512KB private L2 caches. The first three bars show conventional RegionScout implementations with 4K, 8K and 16K CRH tables and 8KB regions [9] (smaller regions reduces benefits). The last bar shows an RT implementation that uses 1KB regions. The RT implementation uses precise information to reduce traffic even further and with less resources. Per node, the original RS requires approximately 178.3Kbits while the RT requires just 2K non-shared bits and is actually 5.5% smaller than a conventional tag array.

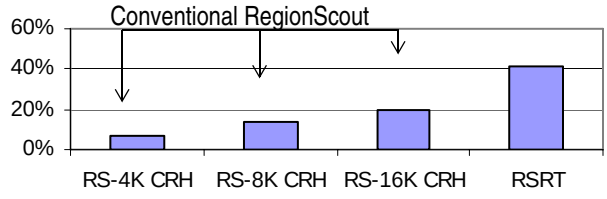

Figure 4: *Coherence traffic reduction with original RegionScout (RS) and RegionTracker-embedded RegionScout (RSRT).*

References

[1] J. Cantin et. al. *Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking*. In Proc. Intl' Symposium on Computer Architecture, June 2005.

[2] J. Cantin, et. al. *Stealth Prefetching*. In the Proc. of the 12th Intl Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 2006.

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

[4] K. Gharachorloo, et. al. *Two techniques to enhance the performance of memory consistency models*. In Proc. of the 1991 Intl' Conference on Parallel Processing, Aug. 1991.

[5] N. Hardavellas, et. al. *SimFlex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture*. SIGMETRICS Performance Evaluation Review, Apr. 2004.

[6] 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.

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

[8] M. K. Martin, et. al. *Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared Memory Multiprocessors.* In Proc. of the 30th Intl' Symp. on Computer Architecture, June 2003.

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

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

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

[12] B. Sinharoy, et. al. *POWER5 System Microarchitecture*. IBM Journal of Research and Development, 49(4): 505-522, 2005.

[13] S. Somogyi, et. al. *Spatial Memory Streaming*. In Proc. Intl' Symposium on Computer Architecture, June 2006.

[14] T. F. Wenisch, et. al. *Temporal streaming of shared memory*. In Proc. Intl' Symposium on Computer Architecture, June 2005.

[15] R. E. Wunderlich, et. al. *SMARTS: Accelerating microarchitecture simulation through rigorous statistical sampling*. In Proc. of the 30th Intl' Symposium on Computer Architecture, June 2003