Coarse-Grain Coherence Tracking: RegionScout and Region Coherence Arrays

COARSE-GRAIN COHERENCE TRACKING IS A NEW TECHNIQUE THAT EXTENDS A CONVENTIONAL COHERENCE MECHANISM AND OPTIMIZES COHERENCE ENFORCEMENT. IT MONITORS THE COHERENCE STATUS OF LARGE REGIONS OF MEMORY AND USES THAT INFORMATION TO AVOID UNNECESSARY BROADCASTS AND FILTER UNNECESSARY CACHE TAG LOOKUPS, THUS IMPROVING SYSTEM PERFORMANCE AND POWER CONSUMPTION.

Jason F. Cantin
James E. Smith
Mikko H. Lipasti
University of Wisconsin-Madison

Andreas Moshovos
University of Toronto

Babak Falsafi
Carnegie Mellon University

Cache-coherent shared-memory multiprocessors have wide-ranging applications, from commercial transaction processing and database services to large-scale scientific computing. As system architectures have grown, incorporating larger numbers of faster processors, the memory system has emerged as the critical factor affecting overall system performance and scalability. Consequently, designers seek techniques for improving memory system bandwidth, latency, and power efficiency. (The "Related work" sidebar summarizes some of these techniques.)

To maintain coherence and exploit fast cache-to-cache transfers, shared-memory multiprocessor systems commonly broadcast memory requests to all the other processors in the system. Examples of such systems are the Sun Fire servers and the IBM eServer Power4 systems.[1,2] Although broadcasting is a quick and simple way to find cached copies, correctly order requests, and locate the appropriate memory controller, it consumes considerable interconnect bandwidth and, as a byproduct, increases access latency for nonshared data. Furthermore, broadcasting consumes substantial amounts of power, both in system interconnect and cache tag arrays.[3]

Shared-memory multiprocessor systems broadcast many memory requests unnecessarily—for example, when the accessed data is not currently shared, or the request is an instruction fetch and the instructions have not been modified. Theoretically, a system could send these requests directly to memory without a broadcast and without violating coherence. However, detecting these cases requires *a priori* knowledge of the coherence status of lines in other caches.

We propose a new technique called coarse-grain coherence tracking (CGCT), which allows a processor to determine in advance that a request does not require a broadcast. CGCT can be implemented as a layered extension to a conventional multiprocessor system with a modest amount of hardware. This article presents two CGCT implementations, RegionScout and Region Coherence

Published by the IEEE Computer Society

Arrays, and provides simulation results for a broadcast-based multiprocessor system running commercial, scientific, and multiprogrammed workloads.

Overview

CGCT uses a conventional cache coherence protocol to maintain coherence over the processors' caches. Unlike a conventional system, however, each processor contains additional hardware for monitoring the coherence status of large, aligned memory regions, each region encompassing a power-of-two number of cache lines. This hardware keeps track of memory regions from which the processor is caching lines and snoops external requests to provide a region snoop response. This response is piggybacked onto the conventional snoop response and used by the requesting processor to determine whether broadcasts are necessary for subsequent requests.

For example, consider a shared-memory multiprocessor with a cache in each processor. Processor A performs a load operation to address X. This load misses in the cache, so processor A reads its region state information for corresponding region Rx. The state of region Rx in other caches is unknown, so processor A broadcasts a read request for address X. The other processors snoop the request, check their region state information, and send a region snoop response back to the requesting processor with the conventional response. No processors are currently caching data from region Rx, so processor A records that the region is nonshared. Until another processor makes a request for a line in region Rx, processor A can access any line in Rx without a broadcast.

CGCT can extend broadcast-based systems to achieve many of the benefits of directory-based systems, including low interconnect and cache tag traffic and low-latency access to nonshared data. Moreover, with an underlying broadcast protocol, CGCT keeps intervention latency low. It accomplishes this by exploiting spatial locality beyond the cache line, without the increased false sharing and internal fragmentation that result from increasing the cache line size.

Performance potential

Figure 1 illustrates CGCT's performance potential. The graph shows the percentage of

Related work

Several earlier proposals led to the development of CGCT. Moshovos et al. proposed Jetty, a method in which each node avoids snoop-induced cache tag lookups that would otherwise miss.[1] Nodes maintain two structures that respectively represent a subset of cached lines (exclusive Jetty) and a superset of cached lines (inclusive Jetty). In contrast, CGCT enables a requesting node to determine in advance that a request would miss in all other nodes.

Ekman, Dahlgren, and Stenström proposed the page-sharing table (PST), a snoop-energy reduction technique for chip multiprocessors with virtual caches.[2] This technique uses vectors that identify sharing at the page level. Every node keeps precise information about the pages it is caching. This information forms a page-level sharing vector in response to coherence requests. Subsequent requests are snooped only by nodes that have lines on the same page, reducing energy consumption. Additional bus lines are required for broadcasting and collecting the sharing vectors. Occasionally, flushing the cache contents is necessary to maintain correctness.

Some architectures, such as the PowerPC, provide bits that the operating system can use to mark memory pages as "coherence not required."[3] The hardware does not need to broadcast requests for data in these pages. In practice, however, using these bits is difficult because they require operating system support and complicate process migration.

References

1. A. Moshovos et al., "JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers," *Proc. 7th Int'l Symp. High Performance Computer Architecture* (HPCA 01), IEEE Press, 2001, pp. 85-96.
2. M. Ekman, F. Dahlgren, and P. Stenström, "TLB and Snoop Energy-Reduction Using Virtual Caches in Low-Power Chip-Multiprocessors," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 02), ACM Press, 2002, pp. 243-246.
3. C. May et al. (eds.), *The PowerPC Architecture: A Specification for a New Family of RISC Processors* (2nd ed.), Morgan Kaufmann, 1994.

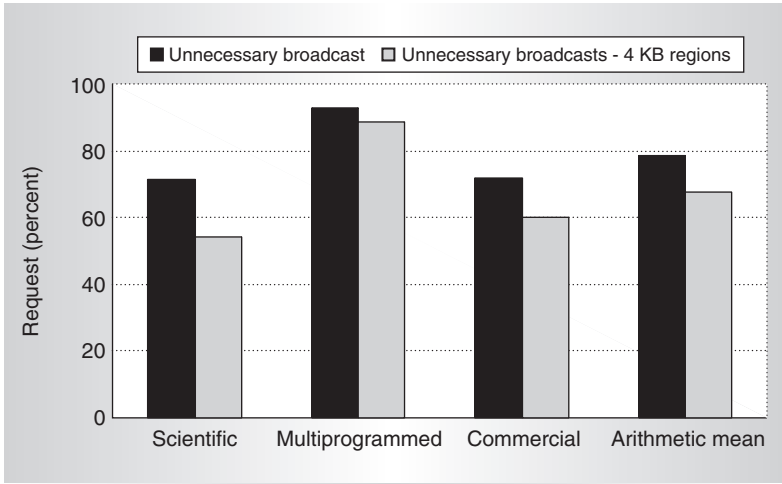

Figure 1. Unnecessary broadcasts in a four-processor system running various applications. According to snoop responses, 71 to 93 percent of requests don't need a broadcast to access the line. From 54 to 89 percent of requests can access a 4-Kbyte region around the requested line without a broadcast.

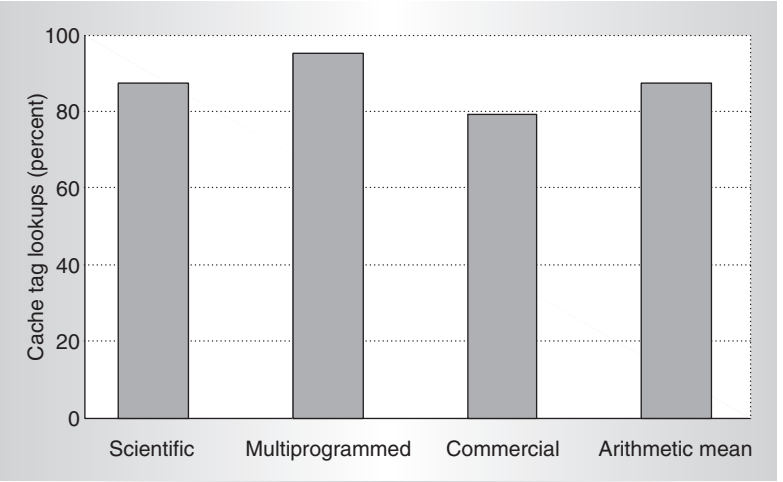

Figure 2. Unnecessary snoop-induced cache tag lookups in a four-processor system. From 79 to 95 percent of external snoop requests do not need to check the cache.

broadcasts that are theoretically unnecessary for a four-processor system. On average, 79 percent of requests don't need a broadcast. More compellingly, an average of 68 percent of requests not only can access the 64-byte line without a broadcast but can access any line in an aligned 4-Kbyte region around the requested line. CGCT potentially can eliminate more than two thirds of the broadcast traffic.

In practice, eliminating a significant number of unnecessary broadcasts can greatly reduce broadcast traffic, queuing delays, and contention. Memory latency will decrease because many data requests will go directly to memory, without first going to a central arbitration point for ordering and to broadcast to all the other nodes. Some requests can be completed without leaving the processor, such as requests to upgrade a shared copy to a modifiable state or to invalidate or flush cached copies.

Power-saving potential

Figure 2 illustrates some of the power-saving potential of CGCT. The figure shows the percentage of unnecessary snoop-induced cache tag lookups for the system shown in Figure 1. On average, 87 percent of snoop-induced cache tag lookups are unnecessary, 79 percent from unnecessary broadcasts, and 8 percent from snoop requests that hit in some (but not all) caches.

Unnecessary broadcasts and cache tag lookups directly increase power consumption and waste energy. Indirectly, they might increase static power consumption because designers often resort to replicating structures to implement interconnects and cache tag arrays that can keep up with the rate of external requests. This increases the amount of circuitry and static power consumption.

CGCT eliminates a significant percentage of unnecessary broadcasts and filters most of the unnecessary cache tag lookups from the remaining snoops. In fact, by filtering snoop-induced cache tag lookups, CGCT can compensate for unnecessary broadcasts that initially went undetected.

CGCT implementations

Two CGCT implementations are presented, RegionScout filters[4] and Region Coherence Arrays (RCAs),[5] representing two design points in cost and accuracy tradeoffs. Whereas RegionScout filters require less storage and are less complex than RCAs, the latter eliminate more unnecessary broadcasts and filter more external snoop requests.

RegionScout filters target the common case of memory requests to regions for which no lines are cached by any other processors. They employ nontagged hash tables to efficiently track regions from which the processor is caching lines, and they respond to external requests with an indication of whether the processor is caching lines from the region. The nontagged hash tables lead to imprecise (but conservative) region tracking. To eliminate as many unnecessary broadcasts as possible, RCAs use tagged structures to precisely track regions from which the processor is caching lines. They provide external requests with a response indicating whether the processor is sharing or modifying lines from the region.

RegionScout filters

A RegionScout filter consists of two structures located in each processor: a cached-region hash (CRH) for tracking data cached by the processor and a nonshared-region table (NSRT) for tracking the global state of regions. The CRH is a hash table of line counts. The NSRT is an associative array, each entry containing an address tag and a valid bit.

When the cache allocates or evicts a line, the CRH count indexed by the corresponding region's address is incremented or decrement-

ed, respectively. The CRH is not tagged, so if lines are cached from multiple regions mapping to the same CRH entry, the line count is the sum of the counts for the regions. Hence, the CRH indicates that there *may* be lines from a region in the cache (the count is nonzero) or that there are no lines from a region in the cache (the count is zero). When a processor broadcasts a request, the other processors' CRHs are checked to determine whether they may be caching lines from the same region. If the region snoop response indicates that no processors are caching any lines in the region, an entry for the region is allocated in the requesting processor's NSRT, meaning that lines in the region can be accessed subsequently without a broadcast. The broadcast also invalidates any matching entries in other processors' NSRTs for correctness.

Because RegionScout uses a nontagged hash table to track cached data, it is space efficient and simple to implement. Regions are not evicted to make room for others, so there is no need to evict lines from the cache to maintain inclusion, no constraint on the data that can be simultaneously cached, and no minimum number of required entries. It can be implemented as a layered extension to an existing system.

Figure 3 shows an example of how RegionScout works. In Figure 3a, node N requests line L in region RL (step 1). After checking its NSRT and finding no matching entry for region RL, node N broadcasts the request. All remote nodes probe their CRHs and report that they don't cache any line in region RL (step 2). Node N records RL as nonshared and increments the corresponding CRH entry (step 3). In Figure 3b, node N is about to request line L′ in region RL and first checks its NSRT (step 1). Because a valid entry is found, it sends the request only to memory (step 2). In Figure 3c, node N′ requests line L′′ in region RL. It first checks its NSRT (step 1). The region is not recorded in its NSRT, so it broadcasts its request (step 2). Node N invalidates its NSRT entry because now RL is shared (step 3).

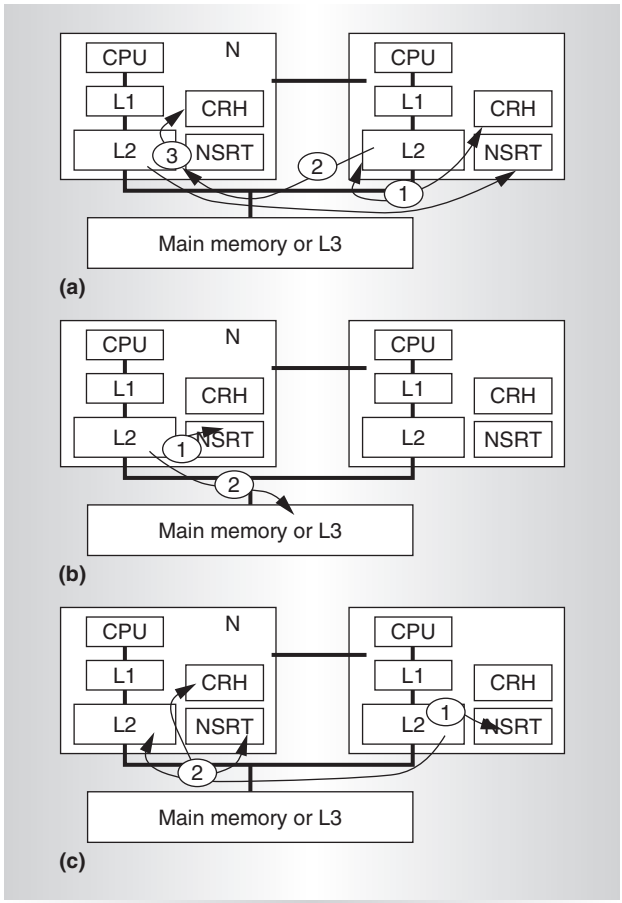

Figure 3. RegionScout example: first request in a region (a); subsequent request in same region (b); another node requests a block in the region (c). RegionScout discovers a nonshared region (a), avoids subsequent broadcasts (b), and later determines that the region has become shared (c).

Region Coherence Arrays

RCAs are tagged structures located in each processor. Each entry contains an address tag, a count of lines cached by the processor, and a protocol state (region state) that summarizes the local and external states of lines in the region.

When a line is allocated in the cache, an entry for the corresponding region is allocated in the RCA (if not already present), and the line count is incremented. The entry's region state encodes the fact that lines in the region are cached by the processor, and, if applicable, that lines are in a potentially modified state. The region state also encodes whether other processors have cached copies of lines in the region and whether any of them are in a potentially modified state (we present a complete discussion of the protocol elsewhere[5]).

Local cache requests and external snoop requests access the RCA. Local cache requests access the RCA in parallel with the cache so that the region state is available on a cache miss. The request type and the region state determine whether a broadcast is necessary. For example,

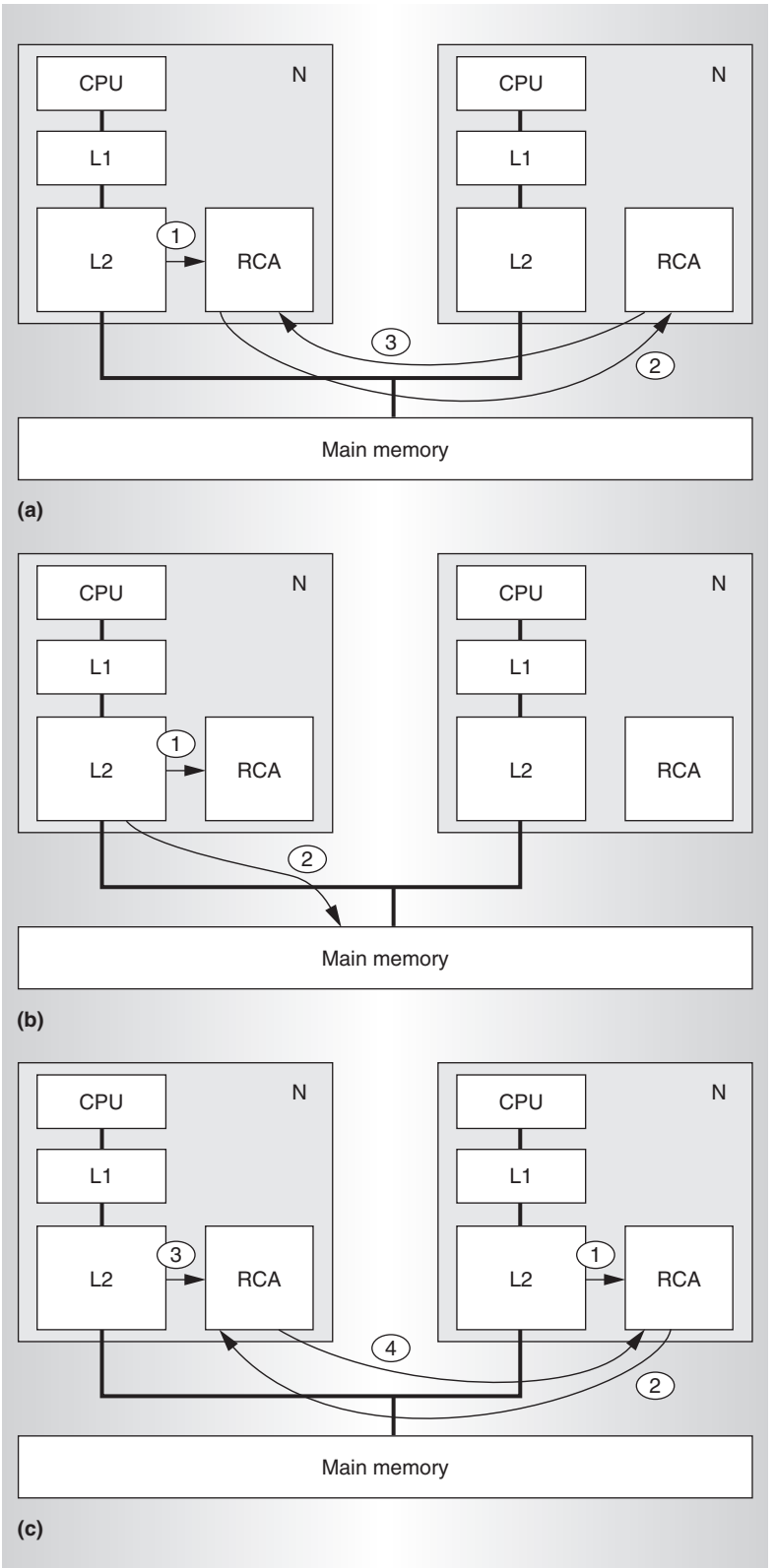

Figure 4. Region Coherence Array example: first access to a region (a); subsequent access to nonshared region (b); region becomes shared (c).

data reads and writes don't need a broadcast to access nonshared data, and instruction fetches don't need a broadcast if the instructions haven't been modified. For external snoop requests, the RCA is accessed to provide a region snoop response indicating whether the processor is caching any lines from the region, and whether any of those lines are modified.

Because RCAs are tagged structures, the information they store for each region is precise and not hashed with other regions. However, to maintain inclusion over the cache, cache lines must occasionally be evicted. That is, when a region must be evicted from the RCA to make room for another, the corresponding lines must be evicted from the cache. For this reason, an RCA should have a number of entries greater than the number of cache locations divided by the number of lines in a region.

Figure 4 shows an example of how RCAs work. In Figure 4a, node N requests line L in region RL and checks its RCA (step 1). It finds no matching entry, so the RCA allocates an entry, and node N broadcasts the request (step 2). All remote nodes check their RCAs and respond that they don't cache any line in region RL. Node N receives the response and updates the RL region state to nonshared (step 3). In Figure 4b, node N is about to request line L′ in region RL and first checks its RCA (step 1). Finding an entry in a nonshared state, node N sends the request directly to memory (step 2). In Figure 4c, node N′ requests line L′′ in region RL. It checks its RCA (step 1). It doesn't find a matching entry, so it broadcasts its request (step 2). Upon receiving the request, node N downgrades its region state to shared and checks the cache for the line (step 3).

Eliminating broadcasts

To avoid broadcasting memory requests, processors must have a way to send memory requests to memory controllers without using the broadcast interconnect. A system with on-chip memory controllers can potentially send requests to memory via on-chip communication paths. To reach memory controllers on other chips, the system can leverage an existing data network to send request packets. If only a single interconnection network is available, the system can still optimize requests by tagging them as "memory only" so that other processors don't snoop them.

Table 1. Simulation parameters.

System	
Processor cores per chip	2
Chips per data switch	2
Processor	
Processor clock	1.5 GHz
Processor pipeline	15 stages
Fetch queue size	16 instructions
Branch target buffer	4K sets, 4-way
Branch predictor	16K-entry Gshare
Return address stack	8 entries
Decode, issue, commit width	4, 4, 4
Issue window size	32 entries
ROB	64 entries
Load/store queue size	32 entries
Integer ALU, integer multiplier	2, 1
FP ALU, FP multiplier	1, 1
Memory ports	1
Caches	
L1 I-cache	32-Kbyte 4-way, 64-byte lines, 1 cycle
L1 D-cache	64-Kbyte 4-way, 64-byte lines, 1 cycle
L2 cache	512-Kbyte 2-way, 64-byte lines, 12 cycles
Prefetching	Power4-style, 8-stream, 5-line run-ahead, R10000-style exclusive
Cache coherence protocols	Write-invalidate MOESI (L2), write-back MSI (L1)
Memory consistency model	Sequential consistency
Interconnect	
System clock	150 MHz
Snoop latency	106 ns (16 cycles)
DRAM latency	106 ns (16 cycles)
DRAM latency (snoop overlapped)	47 ns (7 cycles)
Transfer latency (same switch)	20 ns (3 cycles)
Transfer latency (same board)	47 ns (7 cycles)
Transfer latency (remote)	80 ns (12 cycles)
Memory	
Memory controllers	2
Interleaving granularity	16 Kbytes
DMA buffer size	512 bytes
Coarse-grain coherence tracking	
Region Coherence Array	512, 1K, 2K, and 4K sets, 2-way set-associative
RegionScout NSRT	16 sets, 4-way set-associative
RegionScout CRH	2K, 4K, 8K, 16K, and 32K counts indexed by low address bits
Region sizes	128 bytes, 256 bytes, 512 bytes, 1 Kbytes, 2 Kbytes, and 4 Kbytes

Evaluation methodology

We performed detailed timing evaluation with an execution-driven multiprocessor simulator.[6] We modeled a four-processor system with a Fireplane-like interconnect and 1.5-GHz processors with resources similar to those of the UltraSparc IV. Unlike the UltraSparc IV, however, the processors feature out-of-order instruction issue and an on-chip 1-Mbyte L2 cache (512 Kbytes per processor). We evaluated region sizes ranging from 128 bytes to 4 Kbytes. Table 1 provides a list of parameters.

Assuming 50-bit physical addresses, a 512-Kbyte, 2-way set-associative cache with 64-byte lines requires a 32-bit tag. A RegionScout

Table 2. Simulated workloads.

Category	Benchmark	Description
Scientific	Ocean	Splash-2 ocean simulation, 514 × 514 grid
	Raytrace	Splash-2 ray-tracing application, car
	Barnes	Splash-2 Barnes-hut N-body simulation, 8K particles
Multiprogramming	SPECint2000Rate	Standard Performance Evaluation Corp.'s 2000 CPU integer benchmarks, combination of reduced-input runs
	SPECint95Rate	Standard Performance Evaluation Corp.'s 1995 CPU integer benchmarks
Commercial	SPECweb99	Standard Performance Evaluation Corp.'s Zeus Web server 3.3.7, 300 HTTP requests
	SPECjbb2000	Standard Performance Evaluation Corp.'s Java business benchmark, IBM Java Development Kit 1.1.8 with JIT, 20 warehouses, 2,400 requests
	TPC-W	Transaction Processing Council's Web e-commerce benchmark, database tier, browsing mix, 25 Web transactions
	TPC-B	Transaction Processing Council's original OLTP benchmark, IBM DB2 version 6.1, 20 clients, 1,000 transactions
	TPC-H	Transaction Processing Council's decision support benchmark, IBM DB2 version 6.1, query 12 on a 512-Mbyte database

CRH indexed by the low address bits needs a maximum count per entry equal to the number of lines per region multiplied by the cache associativity, added to the maximum number of outstanding requests. For 4-Kbyte regions, this is an 8-bit counter. With a parity bit, a total of 9 bits per entry is necessary. A RegionScout NSRT entry contains an address tag and a valid bit. For a RegionScout filter with 4-Kbyte regions, 8K CRH entries, and a 64-entry NSRT, the storage overhead is about 9.5 Kbytes.

An RCA, on the other hand, needs an address tag, 3 state bits, a count of lines in the region cached, least-recently-used information, and parity for each entry. For 4-Kbyte regions and a 4K-set, 2-way set-associative structure, storage is approximately 37 bits per entry, or 38 Kbytes for the entire structure. For the same number of entries, an RCA requires about four times as much storage as a RegionScout filter.

Table 2 lists the commercial, scientific, and multiprogrammed benchmarks we simulated in our evaluations. Simulations started from checkpoints and included operating system code. We included cache checkpoints to warm up the caches before simulation.

Evaluation results

We evaluated CGCT in terms of how well it avoids broadcasts and cache tag lookups.

Avoiding broadcasts

Figure 5 shows the percentage of requests sent directly to memory or avoided altogether by various RCA and RegionScout CGCT implementations, compared to the baseline system. The figure also shows the broadcasts that can be avoided by a theoretically optimal CGCT implementation. The RCA and RegionScout implementations substantially reduce the number of broadcasts. However, they behave differently as region size increases. The RegionScout filter continues to improve in performance as the region size increases, up to the 4-Kbyte physical page size. The increasing region size extends the NSRT's reach, while decreasing the probability that regions will collide in the CRH. The RCA, on the other hand, achieves its best performance for region sizes of 1 to 2 Kbytes. The benefit of the 4-Kbyte region's additional reach is offset by increased false sharing of the region (two or more processors accessing different lines in the same region). RCAs must evict cache lines to maintain inclusion, so we only show data for configurations in which the number of entries in the RCA multiplied by the number of lines in a region is greater than the number of entries in the cache. Note that for the optimal implementation (oracle), false sharing of the region is the only factor limiting performance, and false sharing increases with region size.

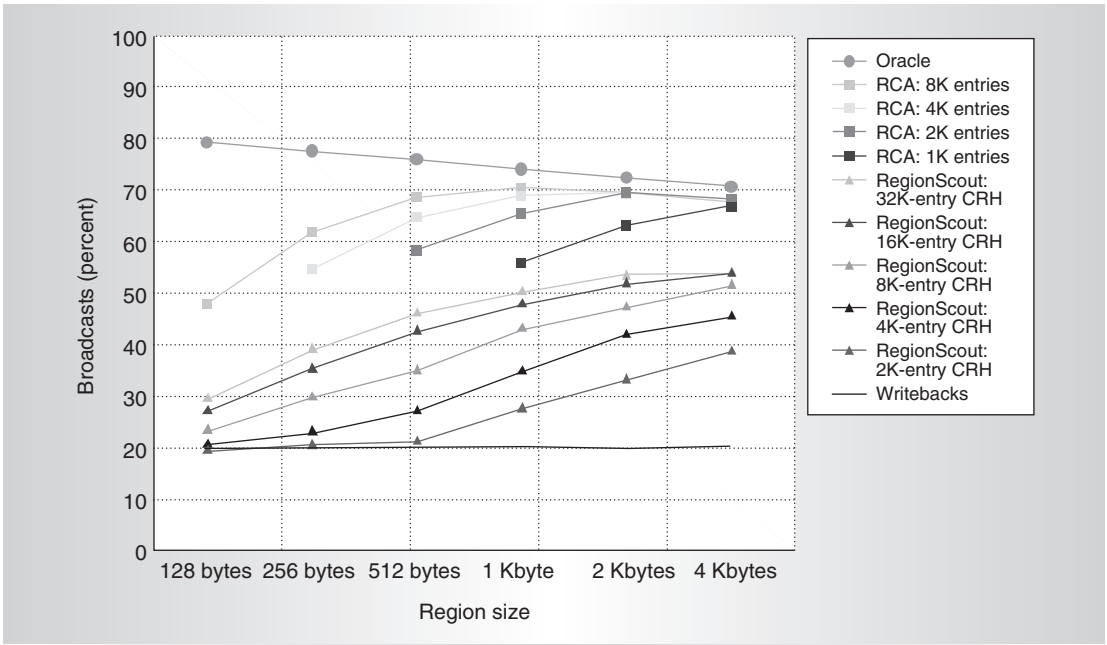

Figure 5. CGCT's effectiveness in avoiding broadcasts in a four-processor system.

Figure 5 shows write-backs separately to distinguish their contribution from that of other requests. Write-backs usually don't require broadcasting, but processors sometimes broadcast them to find the appropriate memory controller and simplify ordering[2]. However, a system with CGCT already has the means to send requests directly to memory controllers and keeps track of the appropriate memory controller to avoid broadcasting requests. With CGCT, therefore, avoiding broadcasts for write-backs can be less expensive.

Avoiding cache tag lookups

Figure 6 shows the percentage of external cache tag lookups from the baseline system eliminated by CGCT. For the most part, the reduction in lookups is a direct result of the reduction in broadcasts. Nevertheless, CGCT filters significant percentages of remaining snoop-induced cache tag lookups.

For RegionScout filters with 8K or fewer CRH entries, the ability to filter snoop-induced cache tag lookups improves with increasing region size. The larger region size extends the reach of the CRH and reduces collisions. However, beyond 8K entries, the additional reach of a large region size is offset by the increased probability of caching a line in that region.

The RCA's ability to filter snoop-induced cache tag lookups decreases as region size increases, and is insensitive to the number of entries it has. This is because it precisely tracks regions from which the processor is caching lines and does not combine information from multiple regions as with RegionScout.

In the work described here, we used CGCT only to avoid unnecessary broadcasts and cache tag lookups. Other optimizations are possible. For example, *a priori* knowledge of whether other processors are sharing a line provides a way to prioritize DRAM accesses. Giving priority to nonspeculative DRAM accesses can reduce wasted DRAM bandwidth and improve latency in systems that overlap DRAM accesses with snoops.

CGCT also has the potential for enhancing prefetching techniques. The CGCT hardware can detect that other processors are modifying lines and can throttle prefetching of those lines. Conversely, by detecting that lines are not shared, CGCT can identify lines that can be prefetched safely and aggressively. Furthermore, when a processor accesses a region that other processors don't share, it accesses a high percentage of the lines. Hence, safe and efficient region prefetching is possible.

In addition, CGCT may be used to efficiently exploit store memory-level parallelism

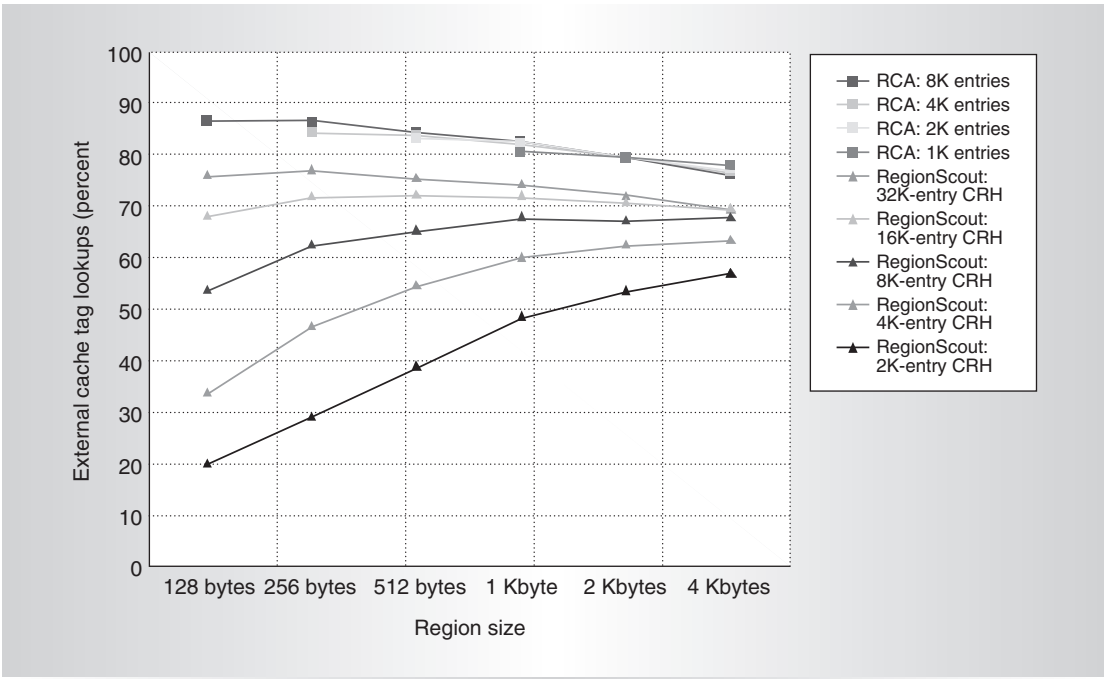

Figure 6. CGCT's effectiveness in filtering external cache tag lookups in a four-processor system.

(MLP)[7]. By efficiently identifying lines that are not shared by other processors, stores that miss in the cache can write their data to the cache before the rest of the line is retrieved from memory.[7] This reduces pressure on processor queues for buffering stores, and allows more stores to be handled in parallel.

Finally, CGCT might have applications in power- and area-optimized memory hierarchy structures that provide caching information for a large number of requests using very small structures.

Acknowledgments

Thanks to Angelos Bilas, Harold Cain, Brian Fields, Mark Hill, Andrew Huang, Ibrahim Hur, Candy Jelak, Steven Kunkel, Kevin Lepak, Martin Licht, Don McCauley, Dionisios Pnevmatikatos, and William Starke for their help. This research was supported by NSF grants CCR-0083126, CCR-0133437, and CCF-0429854; SRC contract 901.001; an NSERC Discovery grant; a CFI grant; and fellowships from the NSF and IBM.

References

1. A. Charlesworth, "The Sun Fireplane System Interconnect," *Proc. Conf. Supercomputing* (SC 01), ACM Press, 2001, p. 7.
2. J. Tendler, S. Dodson, and S. Fields, *IBM eServer Power4 System Microarchitecture*, tech. white paper, IBM Server Group, 2001.
3. A. Moshovos et al., "JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers," *Proc. 7th Int'l Symp. High Performance Computer Architecture* (HPCA 01), IEEE Press, 2001, pp. 85-96.
4. A. Moshovos, "RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence," *Proc. Int'l Symp. Computer Architecture* (ISCA 05), ACM Press, 2005, pp. 234-245.
5. J. Cantin, M. Lipasti, and J. Smith, "Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking," *Proc. Int'l Symp. Computer Architecture* (ISCA 05), ACM Press, 2005, pp. 246-257.
6. H. Cain et al., "Precise and Accurate Processor Simulation," *Proc. Workshop Computer Architecture Evaluation Using Commercial Workloads*, 2002, http://www.hpcaconf.org/hpca8/caecw02.pdf.
7. Y. Chou, L. Spracklen, and S. G. Abraham, "Store Memory-Level Parallelism Optimizations for Commercial Applications," *Proc. 38th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (Micro-38), IEEE Press, 2005, pp. 183-196.

Jason F. Cantin is pursuing a PhD in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison. His research interests include shared-memory multiprocessor systems, fault-tolerant computing, and VLSI. Cantin has BS degrees in electrical engineering and computer engineering from the University of Cincinnati and an MS in computer engineering from the University of Wisconsin.

James E. Smith is a professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison. His research interests include high-performance and power-efficient processor implementations, processor performance modeling, and virtual machines. Smith has a PhD in computer science from the University of Illinois. He is a member of the IEEE and the ACM.

Mikko H. Lipasti is an associate professor in the Department of Electrical and Computer Engineering at the University of Wisconsin-Madison. His research interests include high-performance processor and multiprocessor architecture and the interaction between hardware and modern system software. Lipasti has a PhD in electrical and computer engineering from Carnegie Mellon University. He is a member of the IEEE and Tau Beta Pi.

Andreas Moshovos is an assistant professor in the Electrical and Computer Engineering Department of the University of Toronto. His research interests include microarchitectural optimizations for high-performance processors and systems. Moshovos has a Ptyhion degree and an MSc, both in computer science, from the University of Crete, Greece, and a PhD in computer science from the University of Wisconsin-Madison. He is a member of IEEE and the ACM.

Babak Falsafi is an associate professor of electrical and computer engineering and a Sloan Research Fellow at Carnegie Mellon University. His research interests include computer architecture with emphasis on high-performance memory systems, architectural support for gigascale integration, and computer system performance evaluation tools. Falsafi has a PhD in computer science from the University of Wisconsin. He is a member of the IEEE and the ACM.

Direct questions and comments about this article to Jason F. Cantin, Dept. of Electrical and Computer Engineering, University of Wisconsin, 1415 Engineering Dr., Madison, WI 53706; jason@jasoncantin.com.

For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.