Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking

Jason F. Cantin, Mikko H. Lipasti, and James E. Smith
Department of Electrical and Computer Engineering
University of Wisconsin, Madison
{jcantin, mikko, jes}@ece.wisc.edu

Abstract

To maintain coherence in conventional shared-memory multiprocessor systems, processors first check other processors' caches before obtaining data from memory. This coherence checking adds latency to memory requests and leads to large amounts of interconnect traffic in broadcast-based systems. Our results for a set of commercial, scientific and multiprogrammed workloads show that on average 67% (and up to 94%) of broadcasts are unnecessary.

Coarse-Grain Coherence Tracking is a new technique that supplements a conventional coherence mechanism and optimizes the performance of coherence enforcement. The Coarse-Grain Coherence mechanism monitors the coherence status of large regions of memory, and uses that information to avoid unnecessary broadcasts. Coarse-Grain Coherence Tracking is shown to eliminate 55-97% of the unnecessary broadcasts, and improve performance by 8.8% on average (and up to 21.7%).

1. Introduction

Cache-coherent multiprocessor systems have wide-ranging applications from commercial transaction processing and database services to large-scale scientific computing. They have become a critical component of internet-based services in general. As system architectures have incorporated larger numbers of faster processors, the memory system has become critical to overall system performance and scalability. Improving both coherence and data bandwidth, and using them more efficiently, have become key design issues.

To maintain coherence and exploit fast cache-to-cache transfers, multiprocessors commonly broadcast memory requests to all the other processors in the system [1, 2, 3]. While broadcasting is a quick and simple way to find cached data copies, locate the appropriate memory controllers, and order memory requests, it consumes considerable interconnect bandwidth and, as a byproduct, increases latency for non-shared data. To reduce the bottleneck caused by broadcasts, high performance multiprocessor systems decouple the coherence mechanism from the data transfer mechanism, allowing data to be moved directly from a memory controller to a processor either over a separate data network [1, 2, 3], or separate virtual channels [4]. This approach to dividing data transfer from coherence enforcement has significant performance potential because the broadcast bottleneck can be sidestepped. Many memory requests simply do not need to be broadcast to the entire system, either because the data is not currently shared, the request is an instruction fetch, the request writes modified data back to memory, or the request is for non-cacheable I/O data.

1.1. Coarse-Grain Coherence Tracking

In this paper, we leverage the decoupling of the coherence and data transfer mechanisms by developing *Coarse-Grain Coherence Tracking*, a new technique that allows a processor to increase substantially the number of requests that can be sent directly to memory without a broadcast and without violating coherence. Coarse-Grain Coherence Tracking can be implemented in an otherwise conventional multiprocessor system. A conventional cache coherence protocol (e.g., write-invalidate MOESI [5]) is employed to maintain coherence over the processors' caches. However, unlike a conventional system each processor maintains a second structure for monitoring coherence at a granularity larger than a single cache line (Figure 1). This structure is called the *region coherence array* (RCA), and maintains coarse-grain coherence state over large, aligned memory *regions*, where a region encompasses a power-of-two number of conventional cache lines.

On snoop requests, each processor's RCA is snooped along with the cache line state, and the coarse-grain state is piggybacked onto the conventional snoop response. The requesting processor stores this information in it's RCA to avoid broadcasting subsequent requests for lines in the region. As long as no other processors are caching data in that region, requests for data in the region can go directly to memory and do not require a broadcast.

0-7695-2270-X/05/$20.00 (C) 2005 IEEE

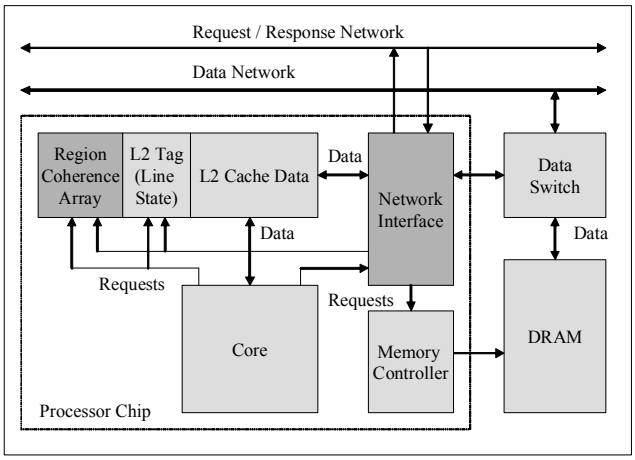

Figure 1. Processor node modified to implement Coarse-Grain Coherence. A Region Coherence Array is added, and the network interface may need modification to send requests directly to the memory controller.

As an example, consider a shared-memory multiprocessor with two-levels of cache in each processor. One of the processors, *processor A*, performs a load operation. The load misses in the L1 cache, and a read request is sent to the L2 cache. The L2 cache coherence state and the region coherence state are read in parallel to determine the status of the line. There is a miss in the L2 cache, and the region state is invalid, so the request is broadcast. Each other processor's cache is snooped, and the external status of the region is sent back to *processor A* with the conventional snoop response. Because no processors were caching data from the region, an entry for the region is allocated in *processor A*'s RCA with an exclusive state for the region. Until another processor makes a request for a cache line in that region, *processor A* can access any memory location in the region without a broadcast.

1.2. Performance Potential

Figure 2 illustrates the potential of Coarse-Grain Coherence Tracking. The graph shows the percentage of unnecessary broadcast requests for a set of workloads on a simulated four-processor PowerPC system. Refer to Section 4 for the system parameters and a description of the workloads used. On average, 67% of the requests could have been handled without a broadcast if the processor had oracle knowledge of the coherence state of other caches in the system. The largest contribution is from ordinary reads and writes (including prefetches) for data that is not shared at the time of the request. The next most significant contributor is write-backs, which generally do not need to be seen by other processors. These are followed by instruction fetches, for which the data is usually clean-shared. The smallest contributor, although still significant, is Data Cache Block (DCB) operations that invalidate, flush, or zero-out cached copies in the system. Most of these are Data Cache Block Zero (DCBZ) operations used by the AIX operating system to initialize physical pages.

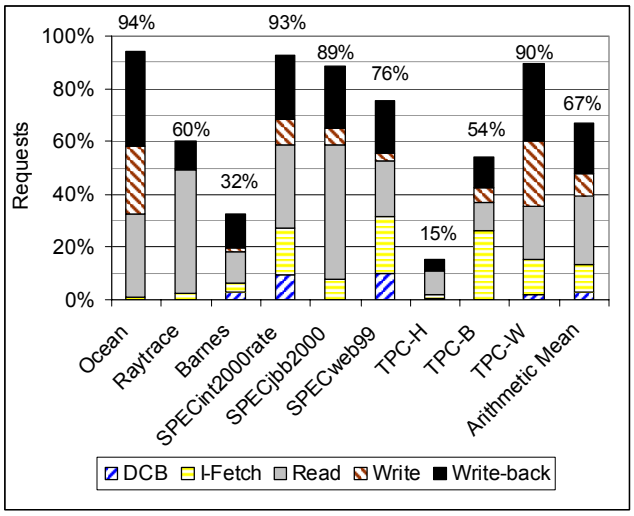

Figure 2. Unnecessary broadcasts in a four-processor system. From 15% to 94% of requests could have been handled without a broadcast.

If a significant number of the unnecessary broadcasts can be eliminated in practice, there will be large reductions in traffic over the broadcast interconnect mechanism. This will reduce overall bandwidth requirements, queuing delays, and cache tag lookups. Memory latency can be reduced because many data requests will be sent directly to memory, without first going to an arbitration point and broadcasting to all coherence agents. Some requests that do not require a data transfer, such as requests to upgrade a shared copy to a modifiable state and DCB operations, can be completed immediately without an external request.

As we will show, Coarse-Grain Coherence Tracking does, in fact, eliminate many of the unnecessary broadcasts and provides the benefits just described. In effect, it enables a broadcast-based system to achieve much of the benefit of a directory-based system (low latency access to non-shared data, lower interconnect traffic, and improved scalability) without the disadvantage of three-hop cache-to-cache transfers. It exploits spatial locality, but maintains a conventional line size to avoid increasing false-sharing, fragmentation, and transfer costs.

1.3. Paper Overview

This paper presents a protocol to implement Coarse-Grain Coherence Tracking and provides simulation results for a broadcast-based multiprocessor system running commercial, scientific, and multiprogrammed workloads. The paper is organized as follows. Related work is surveyed in

the next section. Section 3 discusses the implementation of Coarse-Grain Coherence Tracking, along with a proposed coherence protocol. Section 4 describes evaluation methodology, which is followed by simulation results in Section 5. Section 6 concludes the paper and presents opportunities for future work.

2. Related Work

Sectored caches provide management of cache data at granularities larger than a single line. Sectored caches reduce tag overhead by allowing a number of contiguous lines to share the same tag [6, 7] (see also the references in [8]). However, the partitioning of a cache into sectors can increase the miss rate significantly for some applications because of increased internal fragmentation [7, 8, 9]. There have been proposals to fix this problem, including Decoupled Sectored Caches [9] and the Pool of Subsectors Cache Design [8], both of which achieve lower miss rates for the same cache size by allowing sectors to share space for cache lines. Coarse-Grain Coherence Tracking is not focused on reducing tag overhead and does not significantly restrictions on the placement of data in the cache. It therefore does not significantly affect cache miss rate. Coarse-Grain Coherence Tracking optimizes request routing by maintaining information for larger regions of data, beyond what is in the cache.

Subline caches have been proposed to exploit more spatial locality while avoiding false-sharing in caches [10, 11]. A large transfer line exploits spatial locality, while coherence state is maintained on smaller *sublines* to avoid the increased false-sharing that results from a larger line size. However, transferring larger lines consumes bandwidth, and, like sectoring, the larger lines increase internal fragmentation. Note that the terms "sectored cache" and "subline cache" are often used interchangeably in the literature. Coarse-Grain Coherence Tracking does not necessarily transfer large numbers of cache lines at once.

Dubnicki and LeBlanc proposed an adjustable cache line size [12]. This allows the system to dynamically increase/decrease the line size to tradeoff spatial locality and false-sharing based on application needs. However, at any given time all lines are the same size, and the best size is limited by false-sharing and cache fragmentation. Coarse-Grain Coherence Tracking does not increase fragmentation or false-sharing.

Some architectures, such as PowerPC [13] provide bits that the operating system may use to mark virtual memory pages as *coherence not required* (i.e., the "WIMG" bits). Taking advantage of these bits, the hardware does not need to broadcast requests for data in these pages. However, in practice it is difficult to use these bits because they require operating system support, complicate process migration, and are limited to virtual-page-sized regions of memory [14].

Moshovos et al. proposed Jetty, a snoop-filtering mechanism for reducing cache tag lookups [15]. This technique is aimed at saving power by predicting whether an external snoop request is likely to hit in the local cache, avoiding unnecessary power-consuming cache tag lookups. Like our work, Jetty can reduce the overhead of maintaining coherence; however Jetty does not avoid sending requests and does not reduce request latency.

Moshovos has concurrently proposed a technique called RegionScout that is based on Jetty, and avoids sending snoop requests as well as avoiding tag lookups for incoming snoops [16]. RegionScout uses less precise information, and hence can be implemented with less storage overhead and complexity than our technique, but at the cost of effectiveness.

Saldanha and Lipasti proposed a speculative snoop-power reduction technique for systems with a broadcast tree for a request network [17]. The different levels of the broadcast tree are snooped serially, first checking the nearest neighbors, and then progressively checking nodes that are farther away when necessary. Latency and power are reduced for data in the nearest neighbors, however latency is increased for data on the farther nodes. Coarse-Grain Coherence Tracking, on the other hand, does not increase latency for requests to remote nodes.

Ekman, Dahlgren, and Stenström proposed a snoop-energy reduction technique for chip-multiprocessors with virtual caches based on TLBs [18]. This technique maintains a sharing list with each entry in the TLB, and broadcasts the list with each snoop request so only processors known to be sharing the virtual region need to check their cache tags and respond. This work is similar to ours because it maintains information in the processor that optimizes the handling of external requests, however requests are still broadcast.

There is also recent work in extending SMPs. Isotach networks extend SMPs by not requiring an ordered address interconnect [19]. These networks allow processors to control the timing of message delivery and pipeline messages without violating sequential consistency. Martin et al. proposed Timestamp snooping [20], an improved technique that introduces the concept of "slack" to cope with network delays, and requires fewer messages to implement. Martin et al. later proposed Token Coherence [21], which uses tokens to grant coherence permissions, eliminating the need for ordered networks or logical time, and creating a simple correctness substrate onto which speculative optimizations, such as destination-set prediction can be implemented [22]. Interestingly, the destination-set prediction work keeps information for predictions at a larger granularity than a cache line, called a *macroblock* [22]. This line of research concedes the difficulty of building multiprocessor systems with ordered broadcast

networks and tries to support the increasing rate of broadcast requests, whereas Coarse-Grain Coherence reduces unnecessary broadcasts and sends requests directly to memory with low latency.

3. Coarse-Grain Coherence Tracking Implementation

Coarse-Grain Coherence Tracking is implemented with a *region protocol* that monitors coherence events and tracks coherence state at a coarse granularity. The coarse-grain coherence state is stored in the RCA, and checked by the local processor and external snoop requests. In responses to requests, the region status is sent back via additional snoop response bits.

3.1. Region Protocol

The region protocol observes the same request stream as the underlying conventional protocol, updating the region state in response to requests from the local processor and other processors in the system. The local processor checks the region state before broadcasting requests to the rest of the system, and sends a request directly to memory if the region state indicates that a broadcast is unnecessary. The proposed protocol consists of seven stable states, which summarize the local and global coherence state of lines in the region. The states and their definitions are in Table 1.

Table 1. Region Protocol States.

	Processor	Other Processors	Broadcast Needed?
Invalid (I)	No Cached Copies	Unknown	Yes
Clean-Invalid (CI)	Unmodified Copies Only	No Cached Copies	No
Clean-Clean (CC)	Unmodified Copies Only	Unmodified Copies Only	For Modifiable Copy
Clean-Dirty (CD)	Unmodified Copies Only	May Have Modified Copies	Yes
Dirty-Invalid (DI)	May Have Modified Copies	No Cached Copies	No
Dirty-Clean (DC)	May Have Modified Copies	Unmodified Copies Only	For Modifiable Copy
Dirty-Dirty (DD)	May Have Modified Copies	May Have Modified Copies	Yes

Invalid indicates that no lines are cached by the processor, and the state of lines in other processors' caches is unknown. The first letter of the name of a valid state indicates whether there are clean or modified copies of lines in the region cached by the local processor. The second letter indicates whether other processors are sharing or modifying lines in the region.

The states CI and DI are the *exclusive states*, because no other processors are caching lines from the region, and requests by the processor do not need a broadcast. The CC and DC states are *externally clean*, only reads of shared copies (such as instruction fetches) can be performed without a broadcast. Finally, CD and DD are the *externally dirty* states, broadcasts must be performed on requests to ensure that cached copies of data are found.

The state transition diagrams depicted in Figures 3-5 illustrate the region protocol. For clarity, the exclusive states are solid gray, and the externally clean states are shaded.

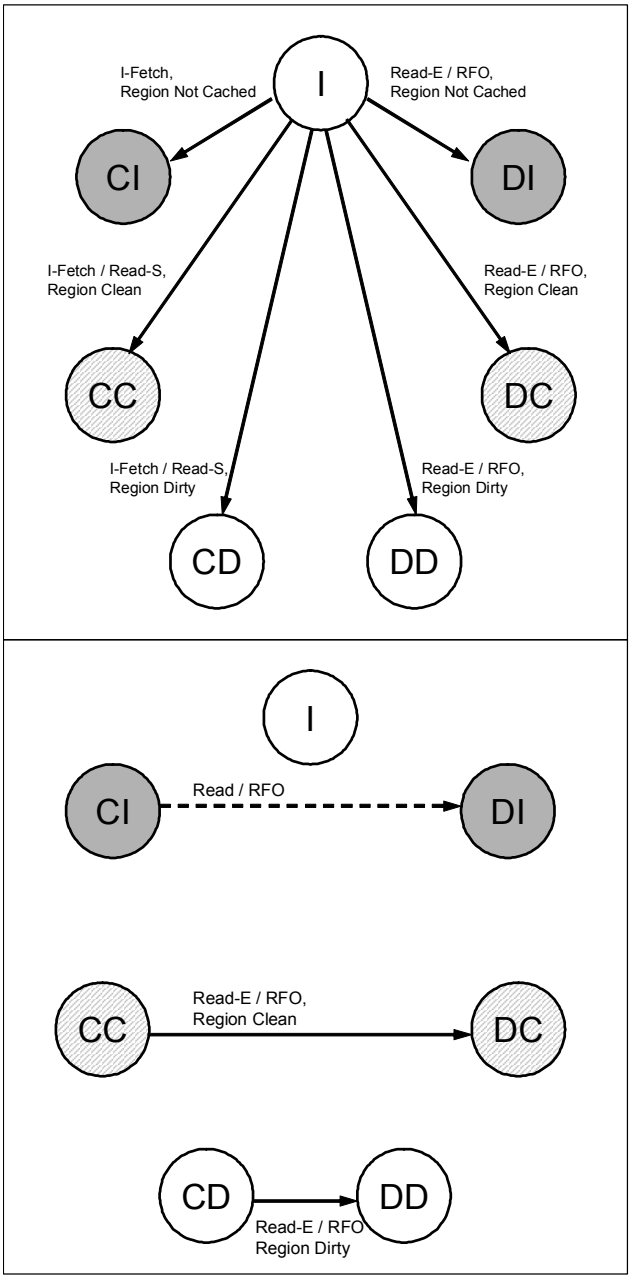

Figure 3. State transition diagrams for requests made by the processor for a given external region state.

From the Invalid state, the next state depends both on the request and the snoop responses (left side of Figure 3). Instruction fetches and Reads of shared lines will change the region state from Invalid to CI, CC, or CD, depending on the region snoop response. Read-For-Ownership (RFO)

operations and Reads that bring data into the cache in an exclusive state transition the region to DI, DC, or DD. If the region is already present in a clean state, than loading a modifiable copy updates the region state to the corresponding dirty state. A special case is for CI, which silently changes to DI when a modifiable copy of a line is loaded (dashed line).

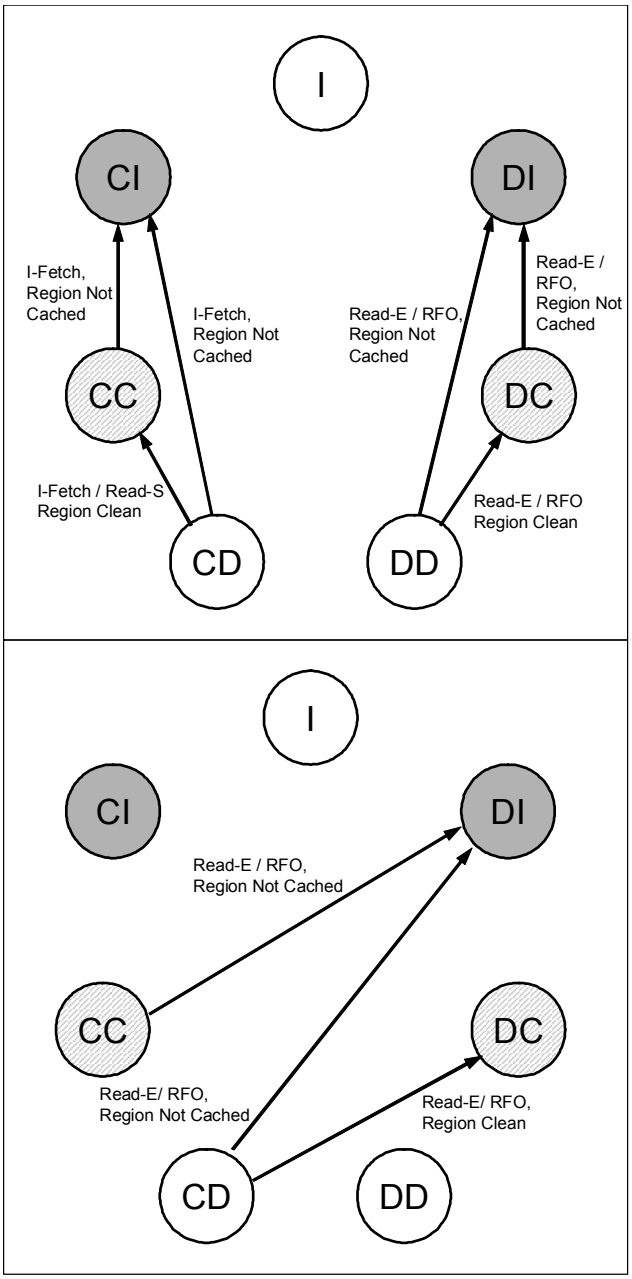

Figure 4. State transition diagrams for processor requests that upgrade the region state.

The transitions in Figure 4 are upgrades based on the snoop response to a broadcast. They not only update the status of the region to reflect the state of lines in the cache, but also use the region snoop response (if possible) to upgrade the region state to an externally clean, or an exclusive state. For example, a snoop is required when in the CC state for RFO operations. If the snoop response indicates that no processors are sharing the region anymore, the state can be upgraded to DI.

The top part of Figure 5 shows how external requests to lines in the region downgrade the region state to reflect that other processors are now reading or modifying lines in the region, indicating that a snoop may be required for subsequent accesses.

An important case occurs for external read requests resulting from loads. If the line snoop response is available to the region protocol, or if the local processor is caching the requested line, then it is known whether the read is going to get an exclusive copy of the line. The region protocol can transition to an externally clean region state (CC, DC) instead of externally dirty (CD, DD).

The bottom part of Figure 5 shows the state transitions for evictions. Also shown is that the protocol implements a form of self-invalidation (for the region state, not the cache line state as in prior proposals for dynamic self-invalidation [23]). When broadcasts cannot be avoided, it is often because the region is in an externally dirty state. Frequently this is overly conservative because there are no lines cached by the remote processors (possibly due to migratory data). Invalidating regions that have no lines cached improves performance significantly for the protocol. To accomplish this, a line count is added to each region to keep track of the number of its lines that are cached by the processor; the count is incremented on allocations and decremented on invalidations. If an external request hits in a region and the line count is zero, the region is invalidated so that later requests may obtain an exclusive copy of the region.

In the protocol loads are not prevented from obtaining exclusive copies of lines. In the state diagrams above, memory read-requests originating from loads are broadcast unless the region state is CI or DI. An alternative approach can avoid broadcasts by accessing the data directly and putting the line into a shared state, however this can cause a large number of upgrades. Future work will investigate adaptive optimizations to address this issue.

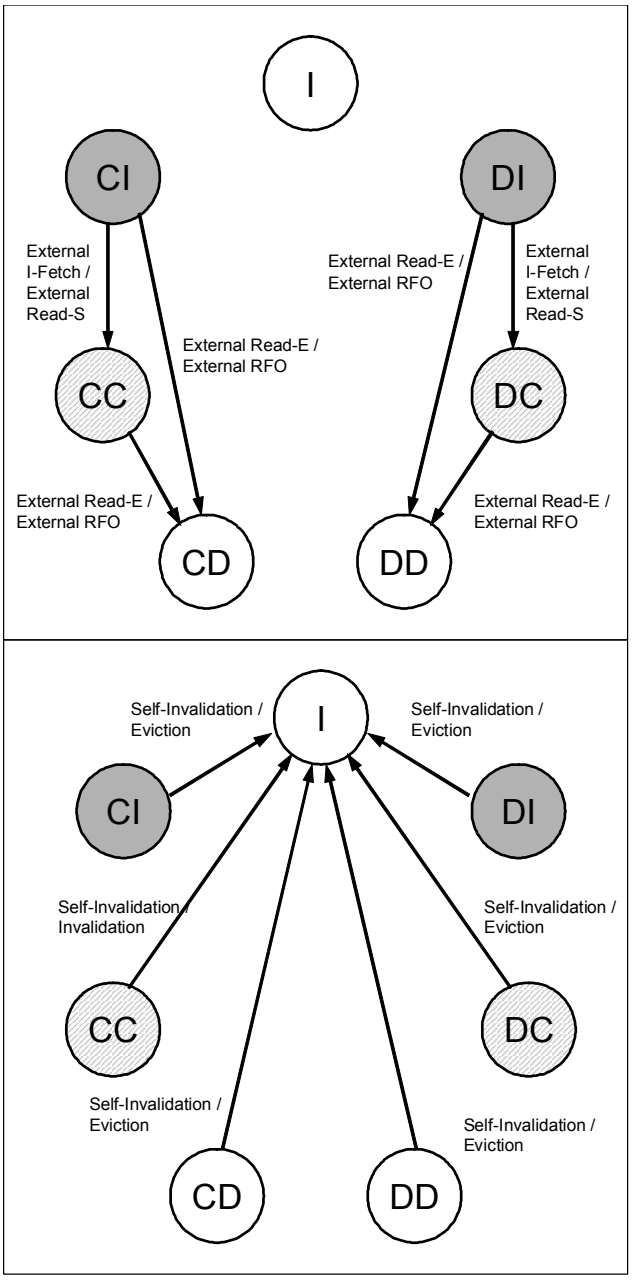

Figure 5. State transition diagrams for external requests.

3.2. Region Coherence Array

The RCA implements an array in each processor for maintaining the region state. An RCA is needed at the lowest levels of coherence above which inclusion is maintained. If the lower levels of the cache hierarchy are not inclusive, there should be region state for the higher levels. In systems with multiple processing cores per chip, only one RCA is needed for the chip [2, 3, 24], unless there is snooping between the cores and it is desirable to conserve on-chip bandwidth. The RCA may be implemented with multiple banks as needed to match the bandwidth requirements of the cache.

For processors to respond correctly to external requests, inclusion must be maintained between the region state and the cache line state. That is, if a line is cached, there must be a corresponding RCA entry so that a region snoop response does not falsely indicate that no lines in the region are cached. Similarly, every memory request for which the requesting processor's region state is invalid must be broadcast to the system to acquire permissions to the region and to inform other processors that may also be accessing lines in the region. Because inclusion is maintained, lines must sometimes be evicted from the cache before a region can be evicted from the RCA. However, the cost can be mitigated. The replacement policy for the RCA can favor regions that contain no cached lines. These regions are easily found via the line count mechanism. Favoring empty regions and using an RCA with 512B regions and the same associativity as the cache, yields an average of 65.1% empty evicted regions, followed by 17.2% and 5.1% having only one or two cached lines, respectively. The increase in cache miss ratio resulting from these evictions is approximately 1.2%.

For a system like the recent Ultrasparc-IV [1, 24], with up to 16GB of DRAM with each processor chip and up to 72 total processors, the physical address length is at least 40 bits. Assuming that each processor has a 1MB 2-way set-associative on-chip cache with 64-byte lines, each line needs 21 bits for the physical address tag, three bits for coherence state, and eight bytes to implement ECC. For each set there is a bit for LRU replacement and eight bits of ECC for the tags and state information (for a total of 23 bytes per set). The cache area overhead based on this design point is shown in Table 2 below.

For the same number of RCA entries as cache entries and 512-byte regions, the overhead is 5.9%. If the number of entries is halved, the overhead is nearly halved, to 3%. The relative overhead is less for systems with larger, 128-byte cache lines like the current IBM Power systems [2, 3].

3.3. Direct Access to Memory Controllers

Though systems such as those from Sun [1] and IBM [3] have the memory controllers integrated onto the processor chip, these controllers are accessed only via network requests. However, a direct connection from the processor to the on-chip memory controller should be straightforward to implement. To avoid broadcasts for other processors' memory, it will be necessary to add virtual channels to the data network so that request packets can be sent to the memory controllers on other chips in the system. Many memory requests are for memory on the same multi-chip module or board, and these can potentially be accessed

with lower latency than a global broadcast. Also, it is easier to add bandwidth to an unordered data network than a global broadcast network.

For systems like the AMD Opteron servers [4], no interconnect modifications are needed. The requests and data transfers use the same physical network, and requests are sent to the memory controller for ordering before being broadcast to other processors in the system. To implement Coarse-Grain Coherence Tracking in this system, a request can be sent to the memory controller as before, but the global broadcast can be skipped. The memory data would be sent back to the requestor, as it would if no other processors responded with the data.

3.4. Additional Bits in the Snoop Response

For the protocol described above, two additional bits are needed in the snoop response (or two encodings). One bit indicates whether the region is in a clean state in other processors' caches (Region Clean), and a second bit indicates whether the region is in other processors' caches a dirty state (Region Dirty). These bits summarize the region as a whole, and not individual lines. They are a logical sum of the region status in other processors' caches, excluding the requestor. This should not be a large overhead, because a snoop response packet may already contain several bits for an address, line snoop response, ECC bits, and other information for routing / request matching.

A scaled back implementation can use only one additional bit (or encoding) to signal whether the region is cached externally. Such a system would only need three region protocol states: exclusive, not-exclusive, or invalid.

4. Evaluation Methodology

Detailed timing evaluation was performed with a multiprocessor simulator [25] built on top of SimOS-PPC [26]. The simulator implements the PowerPC ISA, and runs both user-level and system code. We modeled a four-processor system with a Fireplane-like interconnect and 1.5GHz processors with resources similar to the UltraSparc-IV [24]. Unlike the UltraSparc-IV, the processors feature out-of-order issue, and an on-chip 2MB L2 cache (1MB per processor). For evaluating Coarse-Grain Coherence Tracking, we assume the RCA has the same organization as the L2-cache tags, with 8K sets and 2-way associative (16K entries). We evaluated region sizes of 256 Bytes, 512 Bytes, and 1 Kilobyte. A complete list of parameters is in Table 3.

Table 3. Simulation Parameters.

System	
Processors Cores Per Processor Chip	2
Processor Chips Per Data Switch	2
DMA Buffer Size	512-Byte
Processor	
Processor Clock	1.5GHz
Processor Pipeline	15 stages
Fetch Queue Size	16 instructions
BTB	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
Int-ALU/Int-MULT	2/1
FP-ALU/FP-MULT	1/1
Memory Ports	1
Caches	
L1 I-Cache Size/Associativity/Block-Size/Latency	32KB 4-way, 64B lines, 1-cycle
L1 D-Cache Size/Associativity/Block-Size/Latency	64KB 4-way, 64B lines, 1-cycle (Writeback)
L2 Cache Size/Associativity/Block-Size/Latency	1MB 2-way, 64B lines, 12-cycle (Writeback)
Prefetching	Power4-style, 8 streams, 5 line runahead MIPS R10000-style exclusive-prefetching
Cache Coherence Protocols	Write-Invalidate MOESI (L2), MSI (L1)
Memory Consistency Model	Sequential Consistency
Interconnect	
System Clock	150Mhz
Snoop Latency	106ns (16 cycles)
DRAM Latency	106ns (16 cycles)
DRAM Latency (Overlapped with Snoop)	47ns (7 cycles)
Critical Word Transfer Latency (Same Data Switch)	20ns (3 cycles)
Critical Word Transfer Latency (Same Board)	47ns (7 cycles)
Critical Word Transfer Latency (Remote)	80ns (12 cycles)
Data Network Bandwidth (per processor)	2.4GB/s (16B/cycle)
Coarse-Grain Coherence Tracking	
Region Coherence Array	8192 sets, 2-way set-associative
Region Sizes	256B, 512B, and 1KB
Direct Request Latency (Same Memory Controller)	0.7ns (1 cycle)
Direct Request Latency (Same Data Switch)	13ns (2 system cycles)
Direct Request Latency (Same Board)	27ns (4 system cycles)
Direct Request Latency (Remote)	40ns (6 system cycles)

Figure 6 illustrates the timing of the critical word for different scenarios of an external memory request. The baseline cases are taken from the Fireplane systems [1, 24]. For direct memory accesses employed by our proposed system, we assume that a request can begin one CPU cycle after the L2 access for memory co-located with the CPU (memory controller is on-chip), after two system cycles for memory connected to the same data switch, after four system cycles for memory on the same board, and after six system cycles for the memory on other boards. The Fireplane system overlaps the DRAM access with the snoop; so direct requests see a much longer DRAM latency (9 system cycles).

Table 2. Storage overhead for varying array sizes and region sizes.

	Address Tags (2)	State (2)	Line Count (2)	Mem-Cntrl ID (2)	LRU	ECC	Total Bits	Tag Space Overhead	Cache Space Overhead
4K-Entries, 256-Byte Regions	21	3	3	6	1	9	76	10.2%	1.6%
4K-Entries, 512-Byte Regions	20	3	4	6	1	9	76	10.2%	1.6%
4K-Entries, 1024-Byte Regions	19	3	5	6	1	9	76	10.2%	1.6%
8K-Entries, 256-Byte Regions	20	3	3	6	1	8	73	19.6%	3.0%
8K-Entries, 512-Byte Regions	19	3	4	6	1	8	73	19.6%	3.0%
8K-Entries, 1024-Byte Regions	18	3	5	6	1	8	73	19.6%	3.0%
16K-Entries, 256-Byte Regions	19	3	3	6	1	8	71	38.2%	5.9%
16K-Entries, 512-Byte Regions	18	3	4	6	1	8	71	38.2%	5.9%
16K-Entries, 1024-Byte Regions	17	3	5	6	1	8	71	38.2%	5.9%

As one can see, the request latency is shortest for requests to the on-chip memory controller; otherwise the reduction in overhead versus snooping is offset somewhat by the latency of sending requests to the memory controller. This makes the results conservative because the version of AIX used in the simulations makes no effort at data placement based on the non-uniformity of the memory system.

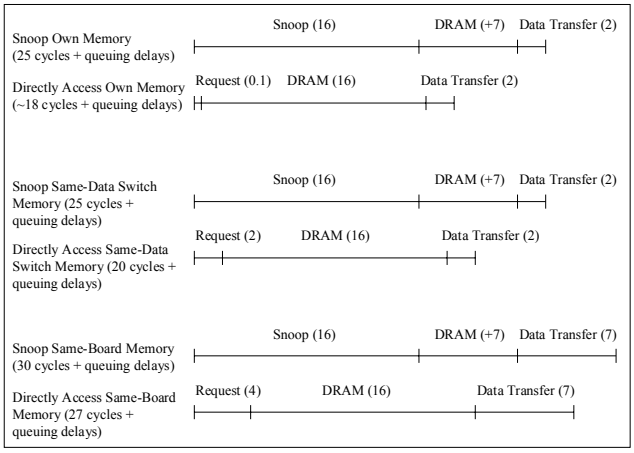

Figure 6. Memory request latency.

For benchmarks, we use a combination of commercial workloads, scientific benchmarks, and a multiprogrammed workload (Table 4). Simulations were started from checkpoints taken on an IBM RS/6000 server running AIX, and include OS code. Cache checkpoints were included to warm the caches prior to simulation. Due to workload variability we averaged several runs of each benchmark with small random delays added to memory requests to perturb the system [27]. The 95% confidence intervals for each workload are shown in the timing results that follow.

Table 4. Benchmarks for timing simulations.

Category	Benchmark	Comments
Scientific	Ocean	SPLASH-2 Ocean Simulation, 514 x 514 Grid
	Raytrace	SPLASH-2 Raytracing application, Car
	Barnes	SPLASH-2 Barnes-Hut N-body Simulation, 8K Particles
Multiprogramming	SPECint2000Rate	Standard Performance Evaluation Corporation's CPU Integer Benchmarks, Combination of reduced-input runs
Web	SPECweb99	Standard Performance Evaulation Corporation's World Wide Web Server, Zeus Web Server 3.3.7, 300 HTTP Requests
	SPECjbb2000	Standard Performance Evaulation Corporation's Java Business Benchmark, IBM jdk 1.1.8 with JIT, 20 Warehouses, 2400 Requests
	TPC-W	Transaction Processing Council's Web e-Commerce Benchmark, DB Tier, Browsing Mix, 25 Web Transactions
OLTP	TPC-B	Transaction Processing Council's Original OLTP Benchmark, IBM DB2 version 6.1, 20 clients, 1000 transactions
Decision Support	TPC-H	Transaction Processing Council's Decision Support Benchmark, IBM DB2 version 6.1, Query 12 on a 512MB Database

5. Results

5.1. Effectiveness at Avoiding Broadcasts

Figure 7 shows both the number of requests for which a broadcast is unnecessary (from Figure 2) and the number of requests that are sent directly to memory or avoided altogether by Coarse-Grain Coherence Tracking.

Except for Barnes and TPC-H, all the applications experience a large reduction in the number of broadcasts. Barnes experiences a 21-22% reduction, while TPC-H experiences only a 9-12% reduction. However, even for these cases, Coarse-Grain Coherence Tracking is capturing a significant fraction of the total opportunity. TPC-H, for example, benefits a great deal from Coarse-Grain Coherence Tracking during the parallel phase of the query, but later when merging information from the different processes there are a lot of cache-to-cache transfers, leaving a best-case reduction of only 15% of broadcasts.

We include write-backs in Figure 7, but put them on top of the stacks to clearly separate the contribution of the other requests. Write-backs do not need to be broadcast, strictly speaking, but they are typically broadcast to find the appropriate memory controller and simplify ordering. Because of the multitude of memory configurations resulting from different system configurations, DRAM sizes, DRAM slot occupancies, and interleaving factors, it is difficult for all the processors to track the mapping of physical addresses to memory controllers [14]. And, in a conventional broadcast system, there is little benefit in adding address decoding hardware, network resources for direct requests, and protocol complexity just to accelerate write-backs. In contrast, a system that implements Coarse-Grain Coherence Tracking already has the means to send requests directly to memory controllers, and one can easily incorporate an index for the memory controller into the region state. Consequently, there is a significant improvement in the number of broadcasts that can be avoided, but this will only affect performance if the system is network-bandwidth-constrained (not the case in our simulations).

5.2. Performance Improvement

Figure 8 shows the reduction in execution time for Coarse-Grain Coherence, with error bars for the 95% confidence intervals. The conversion of broadcasts to direct requests reduces the average memory latency significantly, particularly for 512B regions, leading to average performance gains of 10.4% for the commercial workloads and 8.8% for the entire benchmark set. The largest speedup is 21.7% for TPC-W with 512B regions.

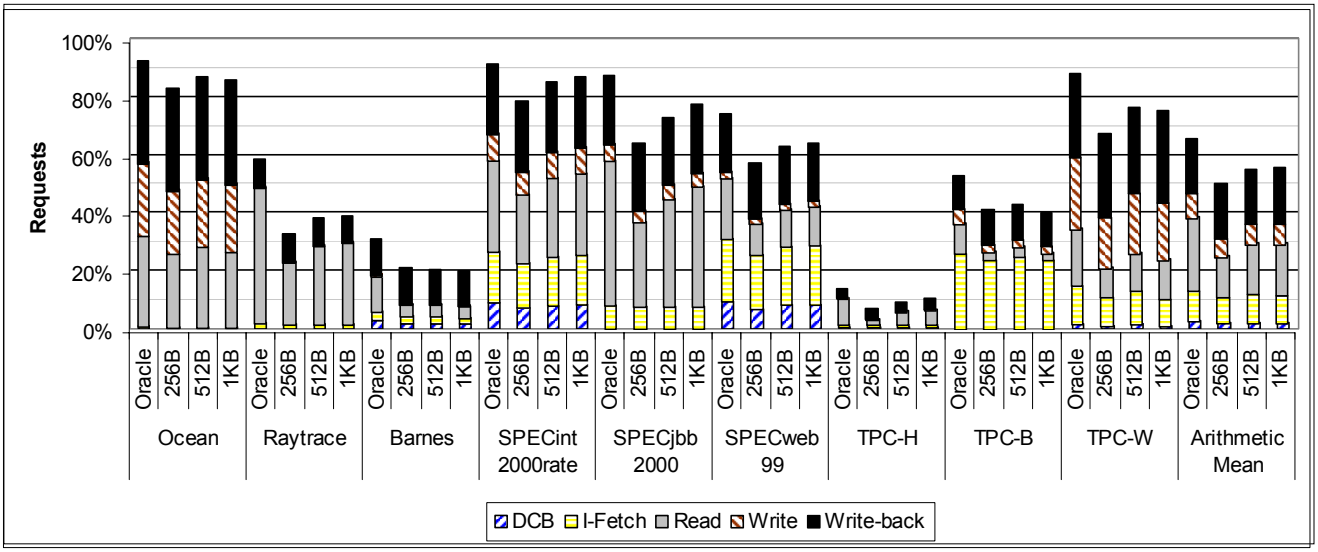

Figure 7. Effectiveness of Coarse-Grain Coherence Tracking for avoiding unnecessary broadcasts and eliminating unnecessary external requests in a four-processor system. The leftmost bar for each benchmark shows the requests for which broadcasts are unnecessary (from Figure 2), and the adjacent bars show the percentage avoided for each region size.

One important design issue is how large the RCA should be. In this study, we found that the average number of lines cached per region ranges from 2.8 to 5. Therefore, one should be able to use half as many sets in the RCA as in the cache and still maintain good performance. Figure 9 shows the runtime for the baseline, 512B regions, and 512B regions with half the number of sets as the cache tags. With half the number of entries (8K) the average performance improvement is 9.1% for the commercial workloads, and 7.8% for all benchmarks. This is only a small decrease in performance for a halved (3% total) cache storage overhead. It is a tradeoff between performance and space overhead that will depend on the complexity and actual physical space overhead of the implementation.

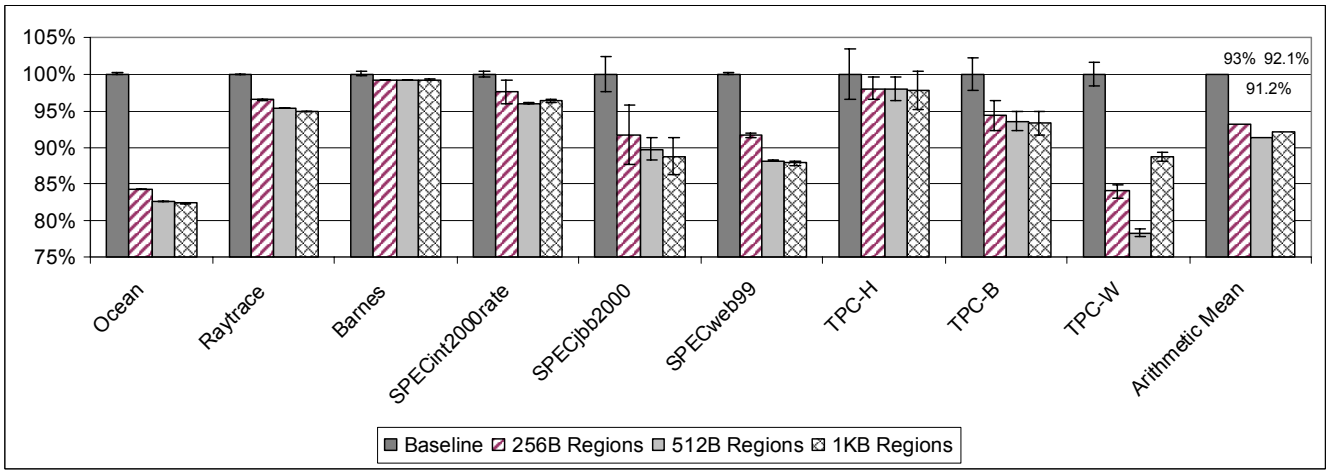

Figure 8. Impact on run time for different region sizes. For these workloads 512B appears to be the best region size, with an average 8.8% reduction in run time.

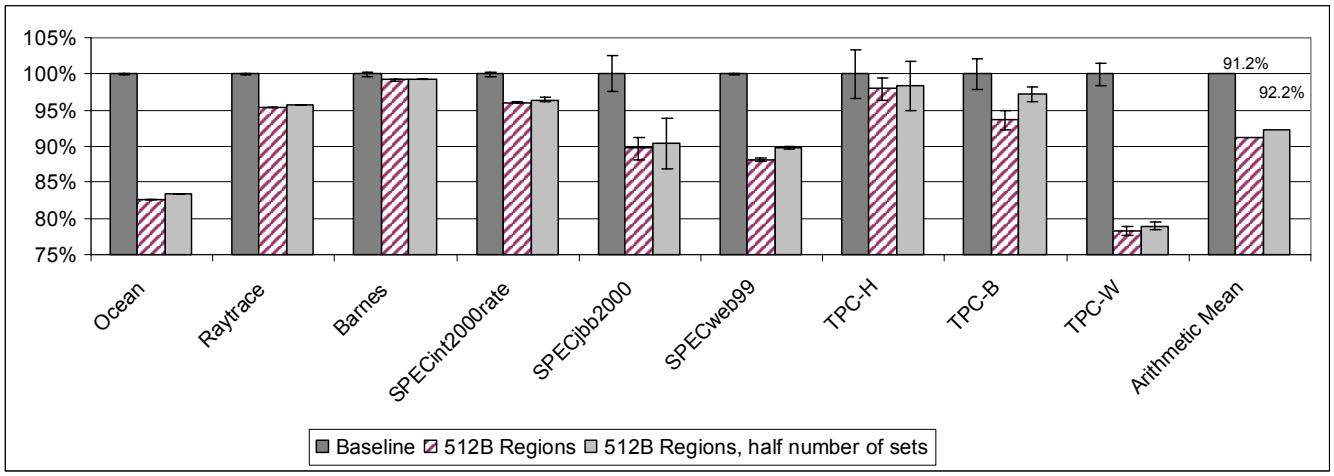

Figure 9. Impact on run time with a region coherence array with half the number of sets as the cache. There is only a 1% difference in the average run time reduction for these workloads.

5.3. Scalability Improvement

By reducing the number of broadcasts, scalability is also improved. In Figure 10, the number of broadcasts performed during the entire run of each application is divided by the number of cycles for both the baseline, and the design with 512B regions, and shown as the average number of broadcasts per 100,000 cycles. Figure 10 also shows the same ratio for the peak traffic, where the peak is the largest number of broadcasts observed for any 100,000 cycle interval. Both the average and peak bandwidth requirements of the system have been reduced to less than half that of the baseline. Coincidentally, the benchmarks used here that have the highest bandwidth requirements are also those that most significantly benefit from Coarse-Grain Coherence Tracking. And, the rate of broadcasts is lower for each benchmark despite the execution time also being shorter.

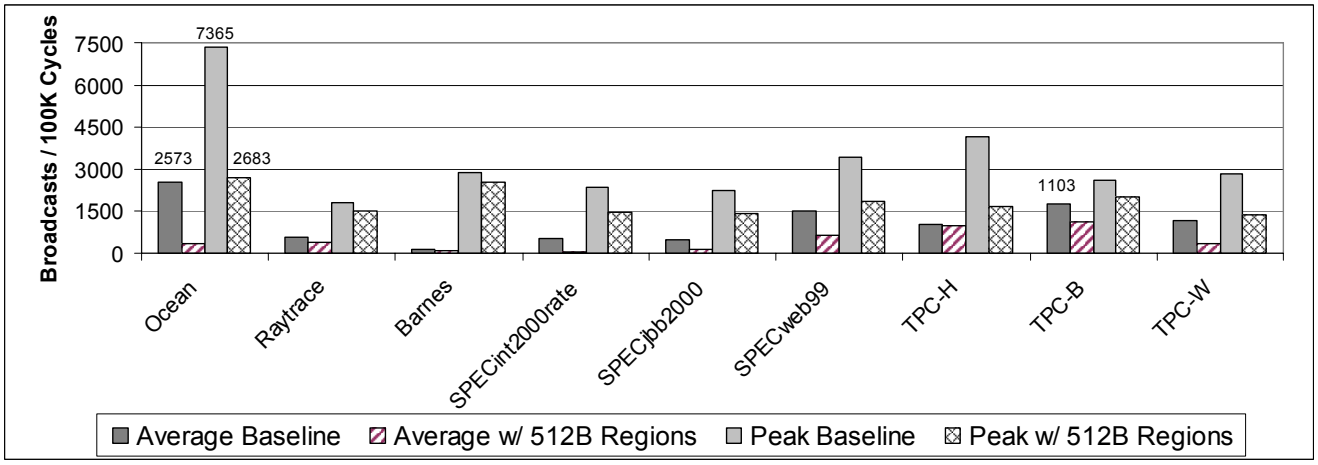

Figure 10: Average and peak broadcast traffic for the baseline and 512B regions. The highest average traffic for the set has gone down from nearly 2,573 broadcasts per 100K cycles for the baseline to 1,103. The peak traffic for any 100K interval in any benchmark has been reduced from 7,365 to 2,683.

6. Conclusions and Future Work

Coarse-Grain Coherence Tracking can significantly reduce average request latency and bandwidth in a broadcast-based multiprocessor system, and hence improve the performance of these systems. Coarse-Grain Coherence does not increase request latency, and the additional cache line evictions needed for maintaining inclusion are negligible. Finally, the implementation impact is manageable. In the hypothetical system we evaluated, it is only necessary to add two bits to the snoop response, and an array for the region state that adds 5.9% to the storage requirements of

the L2 cache. Furthermore, most of the benefit can be achieved with an array nearly half that size.

We used the region state only to summarize the local and remote coherence state of lines in the region. However, regions may also maintain other information. Knowledge of whether data is likely to be cached in the system can be used to avoid unnecessary DRAM accesses in systems that start the DRAM access in parallel with the snoop. The region state can also indicate where cached copies of data may exist, creating opportunities for improved cache-to-cache transfers and invalidations.

An important avenue of future research is in the power-saving potential of Coarse-Grain Coherence. In this paper we only measured performance improvements; however, by reducing network activity [17], tag array lookups [15, 18], and DRAM accesses power can be saved. However, the additional logic may cancel out some of that savings.

Finally, there is potential for extending Coarse-Grain Coherence Tracking with prefetching techniques. Though our system already uses two standard types of prefetching (i.e., MIPS R10000-style exclusive-prefetching [28] and IBM Power4-style stream prefetching [2]), there are new opportunities. The region coherence state can indicate when lines may be externally dirty and hence may not be good candidates for prefetching. The region state can help identify lines that are good candidates for prefetching by indicating when a region of memory is not shared and a prefetch can go directly to memory. Furthermore, prefetching techniques can aid Coarse-Grain Coherence Tracking by prefetching the global region state, going after the 4% of requests for which a broadcast is unnecessary, but the region state was Invalid.

7. Acknowledgements

We thank Harold Cain, Brian Fields, Mark Hill, Andrew Huang, Ibrahim Hur, Candy Jelak, Steven Kunkel, Kevin Lepak, Martin Licht, Don McCauley, and William Starke, for comments on drafts of this paper. We would also like to thank our many anonymous reviewers for their many comments. This research was supported by NSF grant CCR-0083126, and fellowships from the NSF and IBM.

References

[1] Charlesworth, A. *The Sun Fireplane System Interconnect*. Proceedings of SC2001.

[2] Tendler, J., Dodson, S., and Fields, S. *IBM eServer Power4 System Microarchitecture*, Technical White Paper, IBM Server Group, 2001

[3] Kalla, R., Sinharoy, B., and Tendler, J. *IBM Power5 Chip: A Dual-Core Multithreaded Processor* IEEE Micro, 2004.

[4] Weber, F., *Opteron and AMD64, A Commodity 64 bit x86 SOC*. Presentation. Advanced Micro Devices, 2003.

[5] Sweazy, P., and Smith A., *A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus*. Proceedings of the 13th Annual International Symposium on Computer Architecture (ISCA), 1986.

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

[7] Hill, M., Smith, A., *Experimental Evaluation of On-Chip Microprocessor Cache Memories*. Proceedings of the 15th International Symposium on Computer Architecture, 1984.

[8] Rothman, J., and Smith, A., *The Pool of Subsectors Cache Design*. Proceedings of the 13th International Conference on Supercomputing (ICS), 1999.

[9] Seznec, A., *Decoupled Sectored Caches: conciliating low tag implementation cost and low miss ratio*. Proceedings of the 21st Annual International Symposium on Computer Architecture (ISCA), 1994.

[10] Kadiyala, M., and Bhuyan, L. *A Dynamic Cache Subblock Design to Reduce False Sharing*. International Conference on Computer Design, VLSI in Computers and Processors, 1995.

[11] Anderson, C., and Baer, J-L. *Design and Evaluation of a Subblock Cache Coherence Protocol for Bus-Based Multiprocessors*. Technical Report UW CSE TR 94-05-02, University of Washington, 1994.

[12] Dubnicki, C., and LeBlanc, T. *Adjustable Block Size Coherent Caches*. Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA), 1992.

[13] May, C., Silha, E., Simpson, R., and Warren, H. (Eds). *The PowerPC Architecture: A Specification for a New Family of RISC Processors* (2nd Edition). Morgan Kaufmann Publishers, Inc.,1994.

[14] Steven R. Kunkel, Personal Communication, March 2004.

[15] Moshovos, A., Memik, G., Falsafi, B., and Choudhary, A. *JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers*. Proceedings of 7th International Symposium on High-Performance Computer Architecture (HPCA), 2001.

[16] Moshovos, A., *RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence*. Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA). 2005.

[17] Saldanha, C., and Lipasti, M. *Power Efficient Cache Coherence*. Workshop on Memory Performance Issues, in conjunction with the International Symposium on Computer Architecture (ISCA), 2001.

[18] Ekman, M., Dahlgren, F., and Stenström, P. *TLB and Snoop Energy-Reduction using Virtual Caches in Low-Power Chip-Multiprocessors*. Proceedings of ISLPED, 2002.

[19] Reynolds, P., Williams, C., and Wagner, R., *Isotach Networks*. IEEE Transactions on Parallel and Distributed Systems. Vol. 8, No. 4, 1997.

[20] Martin, M., Sorin, D., Ailamaki, A., Alameldeen A., Dickson, R., Mauer C., Moore K., Plakal M., Hill, M., and Wood, D. *Timestamp Snooping: An Approach for Extending SMPs.* Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2000.

[21] Martin, M, Hill, M, Wood, D. *Token Coherence: Decoupling Performance and Correctness.* Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA), 2003.

[22] Martin, M., Harper, P., Sorin, D., Hill, M., and Wood, D., *Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared-Memory Multiprocessors.* Proceedings of the 30th International Symposium on Computer Architecture, 2003.

[23] Lebeck, A., and Wood, D. *Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors*. Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA), 1995.

[24] *UltraSPARC IV Processor, User's Manual Supplement*, Sun Microsystems Inc, 2004.

[25] Cain, H., Lepak, K., Schwartz, B., and Lipasti, M. *Precise and Accurate Processor Simulation*. Proceedings of the 5th Workshop on Computer Architecture Evaluation Using Commercial Workloads, pp. 13-22, 2002.

[26] Keller, T., Maynard, A., Simpson, R., and Bohrer, P. *Simos-ppc Full System Simulator.* http://www.cs.utexas.edu/users/cart/simOS.

[27] Alameldeen, A., Martin, M., Mauer, C., Moore, K., Xu, M., Hill, M., and Wood, D. *Simulating a $2M Commercial Server on a $2K PC.* IEEE Computer, 2003.

[28] Gharachorloo, K., Gupta, A., and Hennessy, J. *Two Techniques to Enhance the Performance of Memory Consistency Models*. Proceedings of the International Conference on Parallel Processing (ICPP), 1991.