Stealth Prefetching

Jason F. Cantin † Mikko H. Lipasti ‡ James E. Smith ‡

† International Business Machines Corp.
11400 Burnet Road
Austin, TX 78727
jfcantin@us.ibm.com

‡ Dept. of Electrical and Computer Engineering
University of Wisconsin, Madison
1415 Engineering Drive
Madison, WI 53705
{mikko, jes}@engr.wisc.edu

Abstract

Prefetching in shared-memory multiprocessor systems is an increasingly difficult problem. As system designs grow to incorporate larger numbers of faster processors, memory latency and interconnect traffic increase. While aggressive prefetching techniques can mitigate the increasing memory latency, they can harm performance by wasting precious interconnect bandwidth and prematurely accessing shared data, causing state downgrades at remote nodes that force later upgrades.

This paper investigates Stealth Prefetching, a new technique that utilizes information from Coarse-Grain Coherence Tracking (CGCT) for prefetching data aggressively, stealthily, and efficiently in a broadcast-based shared-memory multiprocessor system. Stealth Prefetching utilizes CGCT to identify regions of memory that are not shared by other processors, aggressively fetches these lines from DRAM in open-page mode, and moves them close to the processor in anticipation of future references. Our analysis with commercial, scientific, and multiprogrammed workloads show that Stealth Prefetching provides an average speedup of 20% over an aggressive baseline system with conventional prefetching.

Categories and Subject Descriptors C.1.2 [**Computer Systems Organization**]: Processor Architectures: Multiprocessors.

General Terms Design, Performance.

Keywords Multiprocessors, Prefetching, Coherence.

1. Introduction

Cache-coherent shared-memory 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 grown to incorporate larger numbers of faster processors, the memory system has become critical to overall system performance and scalability. Improving memory latency and bandwidth, and doing so power-efficiently, has become a key design challenge.

Modern systems commonly prefetch instructions and data to improve system performance [1, 2, 3, 4]. Prefetching is an effective way to overlap memory latency with computation by speculatively transferring data in anticipation of future references. As memory latency increases, more aggressive prefetching techniques are needed to fully overlap this latency. Many aggressive prefetching techniques have been proposed, however these techniques tend to ignore the important case of shared-memory multiprocessor systems [5, 6, 7, 8, 9, 10, 11].

In shared-memory multiprocessor systems, memory latencies are longer, interconnect bandwidth is more precious, and prefetching data prematurely can hurt performance by causing state downgrades in remote nodes [12]. In shared-memory multiprocessor systems, prefetching must be performed aggressively to overlap the long memory latencies, efficiently to conserve interconnect bandwidth, and with special care not to downgrade or invalidate cached copies of data in other nodes.

This paper proposes and investigates Stealth Prefetching (SP), a new performance-enhancing technique that utilizes Coarse-Grain Coherence Tracking to aggressively, stealthily, and efficiently prefetch data in broadcast-based shared-memory multiprocessor systems. SP does not increase request traffic, take exclusive copies of lines away from other processors, inhibit other processors from obtaining exclusive copies of lines, nor pollute the caches.

1.1 Coarse-Grain Coherence Tracking

Coarse-Grain Coherence Tracking (CGCT) is a recently-proposed technique to reduce broadcast traffic and snoop-induced cache tag lookups in broadcast-based shared-memory multiprocessor systems [13, 14, 15, 16]. CGCT monitors the coherence status of large, aligned memory regions, where a *region* encompasses a power-of-two number of cache lines. CGCT identifies regions of memory that are not shared by other processors, and utilizes this information to avoid unnecessary broadcasts and filter unnecessary snoop-induced cache tag lookups. One way to implement CGCT is with a Region Coherence Array (RCA) [15, 16]. An RCA is an associative array in which each entry consists of a region address tag, a region coherence state, and a count of the lines in the region currently cached by the processor. The region coherence state indicates whether the processor or other processors are caching or modifying lines in the region. The region state is checked on cache misses to determine if the memory request needs a broadcast to obtain a coherent copy of the data.

In this paper, CGCT with RCAs will be utilized to identify non-shared regions not only for efficient coherence maintenance, but to identify regions for stealth prefetching. It will be extended to also track which lines in a region were used in the past in order to improve the accuracy of future prefetches.

1.2 Stealth Prefetching

Stealth Prefetching (SP) is based on the observation that most memory accesses are to regions of memory that are not shared at the time of the access, and a processor accesses most of the lines in such regions. As a result, accesses to a non-shared region can be treated as a prefetch trigger, and the remaining lines in the region can be prefetched to the requesting processor in anticipation of future references.

After a region has been identified as non-shared and a threshold number of lines in the region have been touched, a prefetch request is sent to memory. This prefetch request is piggybacked onto the demand request that met the threshold, and contains a bit-mask of lines to prefetch (one bit for each line in the region). Because the region is non-shared, the request is sent directly to memory, without broadcasting to the other processors in the system. The memory controller fetches the requested lines from DRAM in open-page mode, and sends them back to the requesting processor. The prefetched lines are buffered in a special, *no-permission* coherence state until requested coherently by the processor, invalidated by requests from other processors, or evicted to make room for other prefetched lines.

Prefetched lines are not individually kept coherent. To access prefetched lines, the processor must have access permissions to the region. The lines contain usable data as long as other processors' requests for lines in the region are broadcast, and hence visible to the processor buffering them. The prefetched lines must be discarded if another processor gains exclusive access to the region; the other processor may send requests directly to memory and make the prefetched data stale. Correctness can be maintained simply by invalidating any remaining prefetched lines when the region is evicted or invalidated.

Prefetching in this manner can be aggressive in that a large region of data can be prefetched at once, stealthy in that prefetch requests are not broadcast and prefetching does not interfere with other processors' ability to obtain exclusive copies of lines, and efficient because multiple lines can be obtained with a single request and fetched from DRAM in open-page mode.

As an example, consider a system of processors with caches. Each processor has an RCA and a prefetch buffer. Processor *X* performs a load of line *A* that misses in the cache. Line *A* is part of region *R*, which has not been accessed previously by processor *X*. A broadcast is performed and all other processors are checked for line *A* and their RCAs are checked for region *R*. The combined snoop response indicates that other processors cache neither line *A*, nor any line in *R*. The memory controller fetches line *A* from DRAM and sends this data back to processor *X*. Processor *X* loads line *A* into its cache in an exclusive state and region *R* into its RCA in a non-shared state. When a threshold *T* number of lines in the region are requested by *X*, a prefetch request is piggybacked onto the last request with a bit-mask of lines to prefetch. The memory controller fetches the requested line and prefetches the other lines from DRAM, and sends the data back to processor *X*, where the prefetched lines are buffered for later use. As long as region *R* remains in a non-shared state in processor *X*'s RCA, subsequent requests from processor *X* can obtain data from the prefetch buffer without an external request. If another processor *Y* subsequently requests a line *B* in region *R*, its broadcast is observed by processor *X*. Processor *X* downgrades *R* to an externally-shared state in its RCA, and invalidates line *B* in the buffer.

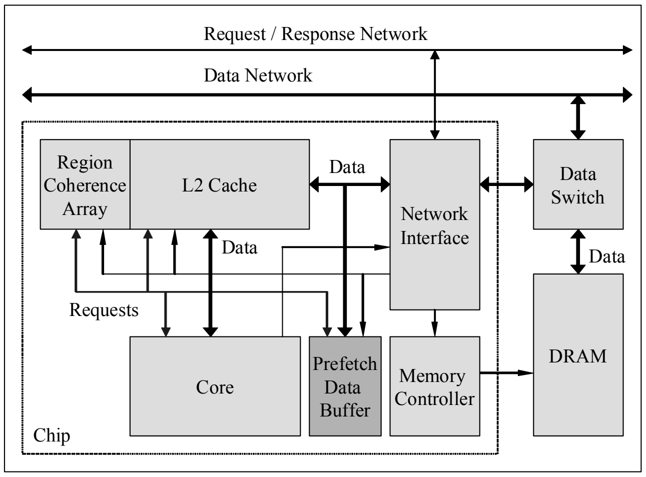

Figure 1. System modified to implement Stealth Prefetching.

Figure 1 illustrates a system modified to implement SP. The starting point is a broadcast-based shared-memory multiprocessor system that implements CGCT with a Region Coherence Array. A prefetch data buffer is added for the prefetched data.

1.3 Performance Potential

Figure 2 illustrates the performance-enhancing potential of SP for a set of commercial, scientific, and multiprogrammed workloads (See Table 3 for a description of the workloads and datasets). A large percentage of the lines from a region are touched while the region is in a non-shared state (on average 45-75%, depending on region size). SP will exploit this behavior to prefetch regions of data aggressively. However, not all applications use most of the lines in a region, so care must be taken to ensure that SP does not waste large amounts of bandwidth.

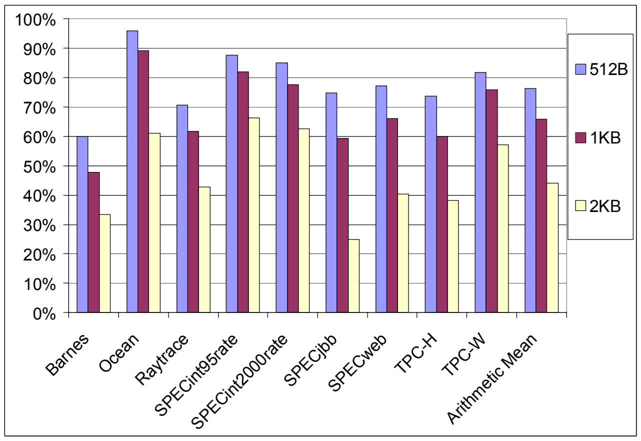

Figure 2. Average lines touched from non-shared regions. Chart shows the average number of lines touched while a non-shared region is resident in the Region Coherence Array for 512B-2KB regions (8-32 lines). Note that one memory access must be performed to bring the region into the RCA; this chart illustrates subsequent accesses that may be prefetched.

1.4 Paper Overview

This paper presents SP and provides simulation results for a broadcast-based multiprocessor system running commercial, scientific, and multiprogrammed workloads. The rest of the paper is organized as follows: The next section surveys related work. Section 3 describes our implementation of Stealth Prefetching. Sections 4 and 5 present our evaluation methodology and experimental results, respectively. Section 6 concludes the paper and lists directions for future work.

2. Related Work

CGCT techniques have been proposed to reduce broadcast traffic and snoop-induced cache tag lookups in broadcast-based shared-memory multiprocessor systems [13, 14, 15, 16]. Moshovos proposed RegionScout, a simple mechanism that uses a hash table to track data cached by the processor and a small tagged array for non-shared regions currently used by the processor [13, 14]. Cantin, Lipasti, and Smith proposed Region Coherence Arrays, which are tagged arrays that use a protocol to track the local and global coherence status of regions used by the processor [15, 16]. Both techniques avoid broadcasts with corresponding improvements in scalability, power consumption, and performance [16]. However, these initial investigations did not evaluate the prefetching potential of CGCT.

Lin, Reinhardt, and Burger proposed Scheduled Region Prefetching (SRP) [5]. SRP aggressively prefetches data at the granularity of regions to exploit spatial locality beyond the cache line. To avoid hurting performance, prefetches are performed only when the Rambus DRAM channels are idle, and prefetched lines are inserted into the cache with low replacement priority. Lin, Burger, and Puzak extended this work with density vectors to mitigate the copious data traffic created by SRP [6]. Density vectors contain a bit for each line in a region, set when the line is accessed during an epoch. Only lines accessed previously are prefetched again to avoid wasted bandwidth. Wang, Burger, Reinhardt, McKinley, and Weems extended SRP again by proposing Guided Region Prefetching (GRP). GRP uses software hints to further improve prefetch accuracy and avoid superfluous prefetches [7]. Later, Wang, Burger, and McKinley proposed pushing prefetched data from GRP into the L1 cache, and adding *evict-me*, a compiler-assisted cache-replacement scheme to reduce cache pollution [8]. While SRP and GRP are aggressive prefetchers, they are focused on uniprocessor systems, and do not address shared data.

Nesbit and Smith proposed prefetching with a *Global History Buffer (GHB)* [9]. The global history buffer holds the most recent miss addresses in FIFO order, and links these addresses together to implement new correlating prefetchers. This technique has only been evaluated for uniprocessor systems, and does not distinguish shared data from non-shared data in a multiprocessor system. Modifications may be necessary to ensure that shared data structures are not prefetched too early.

Hughes and Adve investigated *Memory-Side Prefetching for Linked Data Structures* [10], a technique that uses a programmable prefetch engine located close to memory that traverses linked data structures for aggressive prefetching. The prefetch engine can run far ahead of the processor to overlap long memory latencies for otherwise hard to prefetch data, but also does not make any distinction between shared and non-shared data.

Somogyi, Wenisch, Ailamaki, Falsafi, and Moshovos proposed *Spatial Memory Streaming (SMS)* [11]. SMS is an aggressive prefetching technique that uses code-correlation to predict spatial access patterns, and stream predicted blocks to the processor ahead of their use. Though evaluated on a multiprocessor system, SMS makes no attempt to detect shared data, or avoid prefetching it prematurely.

3. Implementation

Stealth Prefetching can be implemented in a broadcast-based shared-memory multiprocessor system with CGCT (in this paper we use an implementation of CGCT with an RCA). It is implemented with a prefetch buffer [17], a protocol for managing prefetched data, a policy for determining when and what to prefetch, some modifications to the RCA and the memory controller, and a bit-mask in the request packets that are sent directly to the memory controller.

3.1 Stealth Data Prefetch Buffer

The Stealth Data Prefetch Buffer (SDPB) is a small, tagged, sectored data array. Each sector contains a region address tag, LRU bits, multiple lines of data, and two bits for each line for the protocol state. For simplicity, each sector should be large enough to buffer one full region of lines. For performance, the SDPB should have space to buffer several regions.

Sectors in the SDPB are allocated when a region is prefetched. The newly allocated sector is set as the MRU in the set. If a sector must be evicted to make space for a newly allocated region, the prefetched lines that have not yet been accessed by the processor are discarded.

Once lines in the buffer are accessed by the processor, they are moved to the cache and given a valid state. The corresponding sector in the SDPB is set as the MRU in the set to keep regions with useful data buffered. Additionally, the replacement policy avoids evicting regions for which the data has not yet arrived, and invalidates regions with no prefetched lines remaining so that they may be replaced first.

The SDPB does not obtain or maintain coherence permissions over its lines; it relies on the CGCT mechanism to ensure that accesses to the prefetched data are coherent. Lines from a region must be invalidated if another processor gains exclusive access to a region (e.g., the region is self-invalidated from the RCA to give another processor exclusive access, or the region is evicted and another processor requests the region afterward). In this study, prefetched lines from a region are invalidated whenever a region is evicted or invalidated from the RCA, and a prefetched line is invalidated whenever there is an external request for that line.

3.2 Protocol

A simple four-state protocol is used to manage data in the SDPB (See Figure 3). All lines start out in the *Invalid* state, indicating the line is not valid and there is no prefetch for the line in progress.

When a prefetch is initiated, lines are allocated in the SDPB in the *Pending Data* state. This state indicates a prefetch is in progress, and the line has not yet arrived from memory. From this state, the entry can be upgraded to *Present* when data arrives, invalidated if there is an external request for the line (or the processor flushes the lines), or changed to a second pending state if there is a processor request that hits on the entry.

The second pending state *Pending Requested Data* also indicates that the data has been prefetched and has not yet arrived from memory, however there is now a processor request waiting for it. When the data arrives it will be forwarded to the cache and the line will go to the *Invalid* state. Lines in the Pending Requested Data state cannot be invalidated or discarded by other requests; the request for the line has been globally ordered by the CGCT mechanism. Subsequent external invalidates or processor requests to flush the line will hit on the pending state in the cache, and will be handled by the cache coherence protocol.

If the data arrives and there are no pending requests for the data, the line in the SDPB goes to the *Present* state. This state indicates that prefetched data is present and may be accessed by the processor as long as the processor has access to the region. When accessed by the processor, the data is moved to the cache and the SDPB entry is invalidated.

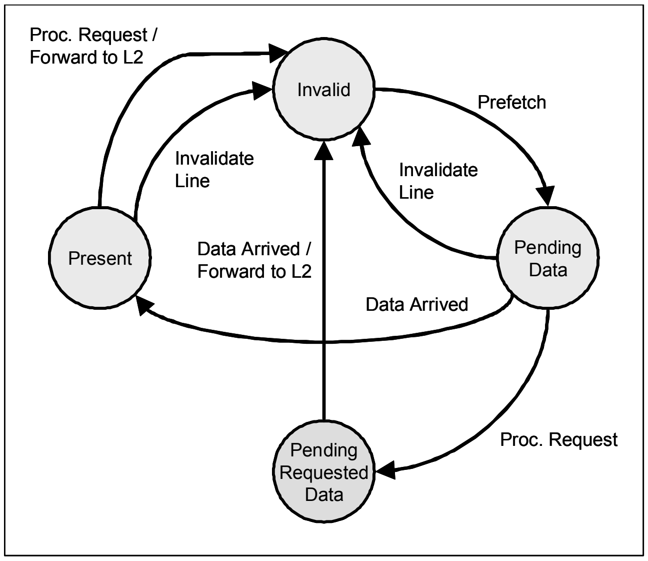

Figure 3. Stealth Data Prefetch Buffer protocol.

3.3 Prefetch Policy

Two key issues are when to first start prefetching a region of data, and when to prefetch lines from that region again. If prefetching is performed too aggressively, a lot of useless data will be transferred. Conversely, if too many lines in the region must be touched before prefetching the rest of the lines, potential is lost and prefetches become less timely. Also, regions tend to stay in the RCA a long time, much longer than lines typically stay in the cache, so it is important not to limit prefetching to when regions are allocated. In this section, we study the effect of having a threshold number of lines touched (the number of lines brought into the L2 cache before a region is prefetched) on Stealth Prefetching.

Figure 4 depicts the average cumulative distribution of lines touched per non-shared region (see Table 3 for a description of the workloads). The X-axis is the number of lines touched, and the Y-axis is the percentage of all non-shared regions touched. Ideally, the graph would resemble a step-function with every region having all lines touched. In that case touching a single line in the region would be an indication that the rest of the lines would be used in the future, and prefetching could be performed aggressively. In reality, not all of the lines are used from 50% of the non-shared regions.

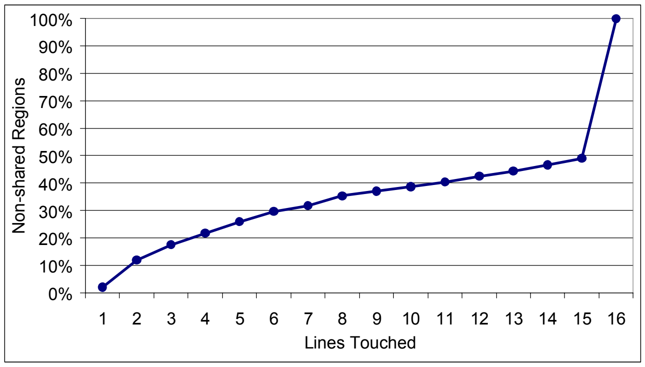

Figure 4. Average cumulative distribution of lines touched per non-shared region for a 1KB (16-line) region.

Based on this data, some threshold number of lines should be touched before initiating a region prefetch to avoid prefetching too much useless data. To determine this threshold, one can take the data from Figure 4, and for a given threshold number of lines calculate the probability that one additional line will be used, two additional lines, and so on. The probability that one additional line will be used is multiplied by one; the probability that two additional lines will be used is multiplied by two, and so on. Each probability is multiplied by the number of lines that would hypothetically be used. The sum of these products is the average number of useful lines prefetched for the given threshold. One can simply subtract that sum from the remaining lines in the region to determine the average number of useless lines prefetched for that threshold. Figure 5 displays the result of this computation for 1KB regions, along with the average number of lines touched per non-shared region for an upper bound.

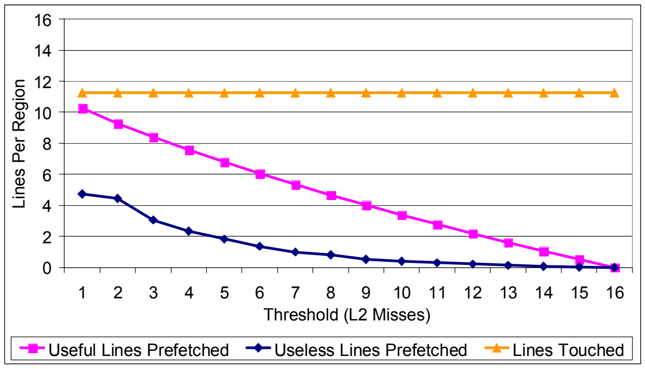

Figure 5. Average useful lines prefetched for a 1KB (16-line) non-shared region with varying thresholds, and the average number of lines touched per non-shared region.

With a threshold of two lines touched, 9.3 of the 16 lines in a region are prefetched and are useful prefetches. Of the leftover 6.7 lines, approximately 2.3 are touched before the prefetch is initiated, and 4.4 are prefetched and not useful. Note that 2.3 and not 2 lines were touched before the prefetch. While the threshold is two lines, sometimes a second line is requested before the region state is known, and so accessing a third line initiates the prefetch. Both the number of useful and useless lines prefetched decrease with increasing threshold. The number of useless lines prefetched drops quickly after a threshold of two lines, leveling off after three lines. It drops below one line for a threshold of seven lines.

Though both the number of useful and useless lines prefetched decrease with increasing threshold, the ratio of useful to useless lines prefetched increases with increasing threshold. Figure 6 shows this trend for 1KB regions, taking the ratio of the useful and useless lines prefetched for varying thresholds from Figure 5. Increasing the threshold number of lines touched before initiating a prefetch will result in a higher ratio of useful to useless data, and can result in more efficient use of data bandwidth and storage. For example, for a threshold of five lines, the ratio of useful to useless prefetched is almost four-to-one; nearly 80% of the data prefetched is useful. However, it should be noted from Figure 5 that higher thresholds result in fewer total lines prefetched; less opportunity is exploited.

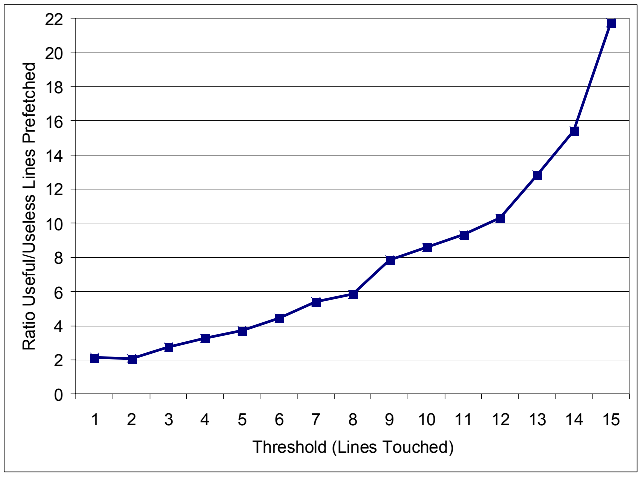

Figure 6. Ratio of useful to useless lines prefetched for a 1KB (16-line) non-shared region with varying thresholds.

Based on these findings, the threshold *T* was set to two. A region of data is prefetched initially after two lines are touched by the processor. For simplicity, a region is considered for prefetch again if two lines are brought into the cache since the last prefetch.

This simplifies implementation because it takes at least one access to a line in the region to bring the region into the RCA and detect that the region is non-shared. Prefetching a region on the first access would be unnecessarily complex. This prefetches large regions of data aggressively, maximizing prefetch timeliness and demonstrating the performance-enhancing potential of Stealth Prefetching in situations where data bandwidth is plentiful. In addition, it gives us an upper bound on the data bandwidth overheads and a lower bound on SDPB utilization. Data bandwidth overhead and SDPB contention can be decreased with higher thresholds.

To further improve the ratio of useful to useless data prefetched, only lines touched previously by the processor are prefetched when a region is prefetched a second time. The RCA keeps track of the lines in the region that have been touched since the last prefetch with a bit-mask; and only these lines are prefetched again. This prefetch-filtering technique is similar to the density vectors used in prior work [6].

3.4 Modifications to RCA

The RCA in each processor needs some small changes to better track which lines are cached, and which lines have been cached while the region is resident in the RCA. First, each RCA entry needs a set of presence bits to keep track of which lines from the region are in the cache to avoid prefetching data that is already cached. These presence bits can be used in place of the line count previously proposed to detective regions with no lines cached (for implementing region self-invalidation and favoring empty regions for replacement) [15]. Second, to improve prefetch efficiency each RCA entry needs a set of bits to keep track of the set of lines from the region that were referenced by the processor since the last prefetch (recently-touched bits). These two sets of bits will be combined to form the bit-mask sent to memory along with the request that triggered the prefetch. The bit-mask is formed by the bitwise logical product of the complemented presence bits, the recently-touched bits, and a complemented bit-mask of lines from the region resident in the SDPB. Essentially, Stealth Prefetching will prefetch only lines that are not resident in the cache or SDPB, and that have been touched since the last prefetch.

Table 1 shows the storage overhead for these extra bits in the Region Coherence Array. For each set there are two regions in a two-way set-associative RCA. For each of these two regions, the line count is replaced with two sets of bits each with (one bit for each line in the region). For example, for 512B regions there are eight 64B lines, so two 4-bit line counts are replaced with four 8-bit masks for a net increase of 24 bits per set. This increases the size of the RCA approximately 35% from the original proposal [15]. The last column shows the total cache storage overhead for the modified Region Coherence Array, which has increased from 5.5% to 7.5% for 512B regions.

Table 1. RCA overhead.

2-way assoc. RCA, 48-bit addresses	Additional Bits / Set	Total Kilobytes	RCA Overhead	Cache Overhead
512B Regions	24	92	35%	7.5%
1KB Regions	54	122	79%	9.9%
2KB Regions	116	184	171%	15.0%

3.5 Modifications to Memory Controller

The memory controller needs to know when to prefetch lines, and what lines to prefetch. It would be detrimental to performance to attempt to prefetch the entire set of lines in a region around every demand-requested line. Instead, the bit-mask sent to the memory controller is used to communicate when and what to prefetch; the memory controller does not prefetch lines unless instructed.

For reduced latency and power consumption, the memory controller can prefetch the requested lines from DRAM in open-page mode. The designer may further reduce latency and leave the DRAM page open after the initial accesses to a line in the region in anticipation of a region prefetch, at the cost of additional power to leave DRAM pages open. The page can be closed in cases where the combined snoop response indicates that the region is in fact shared.

4. Evaluation Methodology

Detailed timing evaluation was performed with PHARMsim [18], a multiprocessor simulator built on top of SimOS-PPC [19]. The simulator implements the PowerPC ISA, and runs both user-level and system code. We modeled a four-processor broadcast-based system with a Fireplane-like interconnect and 1.5GHz processors with resources similar to the UltraSparc-IV [20]. Unlike the UltraSparc-IV, the processors feature out-of-order issue and each processor has a 1MB L2 cache. The baseline system contains two conventional types of prefetching, Power4-style stream prefetching [2], and MIPS-R10000-style exclusive prefetching [21]. For simplicity, these conventional prefetchers are active in all experiments presented here. The Region Coherence Array used to implement CGCT has the same organization as the L2-cache tags: 8K sets and 2-way set-associative (16K entries). We evaluated Stealth Prefetching with a 4-way set-associative SDPB, with space for 256 lines, sectored such that one sector contains a tag and data for all the lines in one region, and a 2-cycle access latency. A complete list of parameters is in Table 2.

For benchmarks, we use a combination of commercial, scientific, and multiprogrammed workloads (Table 3). 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 [22]. For many of the graphs, error bars show the 95% confidence intervals. Arithmetic means are shown for each graph. These means were computed by averaging the results from the scientific, multiprogrammed, and commercial workloads separately, and then combining the resultant means to form an overall mean that weights each category equally.

Table 2. Simulation parameters.

System	
Processors	4
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	32KB 4-way, 64B lines, 1-cycle
L1 D-Cache	64KB 4-way, 64B lines, 1-cycle
L2 Cache	1MB 2-way, 64B lines, 12-cycle
Prefetching	Power4-style, 8 stream, 5 line runahead 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 (Snoop Overlapped)	47ns (7 cycles)
Transfer Latency (Same Switch)	20ns (3 cycles)
Transfer Latency (Same Board)	47ns (7 cycles)
Transfer Latency (Remote)	80ns (12 cycles)
Memory	
Memory Controllers	2
Interleaving Granularity	16KB
DMA Buffer Size	512B
Region Prefetching	
Region Coherence Array	8K sets, 2-way set-associative
Region Sizes	512B, 1KB, 2KB
Stealth Prefetch Data Buffer	256 lines, 1 region per sector, 8-way assoc. 2-cycle latency

Table 3. Workloads simulated.

Category	Benchmark	Comments
Scientific	Barnes	SPLASH-2 Barnes-Hut N-body Simulation, 8K Particles
	Ocean	SPLASH-2 Ocean Simulation, 514 x 514 Grid
	Raytrace	SPLASH-2 Raytracing application, Car
Multiprogramming	SPECint95Rate	Standard Performance Evaluation Corporation's 1995 CPU Integer Benchmarks
	SPECint2000Rate	Standard Performance Evaluation Corporation's 2000 CPU Integer Benchmarks, Combination of reduced-input runs
Commercial	SPECjbb2000	Standard Performance Evaluation Corporation's Java Business Benchmark, IBM jdk 1.1.8 with JIT, 20 Warehouses, 2400 Requests
	SPECweb99	Standard Performance Evaluation Corporation's World Wide Web Server, Zeus Web Server 3.3.7, 300 HTTP Requests
	TPC-H	Transaction Processing Council's Decision Support Benchmark, IBM DB2 version 6.1, Query 12 on a 512MB Database
	TPC-W	Transaction Processing Council's Web e-Commerce Benchmark, DB Tier, Browsing Mix, 25 Web Transactions

5. Results

5.1 Prefetch Effectiveness

Figure 7 shows the percentage of L2 misses that hit in the SDPB for each workload. The first three bars for each workload illustrate the effect of increasing the region size, and the rightmost bar illustrates the effect of a *perfect prefetcher* (all buffer accesses hit; buffer is accessed on a data request after the RCA determines a broadcast is not necessary to coherently acquire the data). On average, 31% of the L2 misses hit in the SDPB for 1KB regions (out of a perfect 51%), and do not suffer the latency of a miss to memory. Benefiting the most is Ocean, for which over 69% of the L2 misses are prefetched. The next best performer is TPC-W, for which over 52% of the L2 misses are prefetched. Overall, 1KB (16-line) regions perform best for the set, achieving a balance between aggressiveness, buffer utilization, and effectiveness at tracking of non-shared regions.

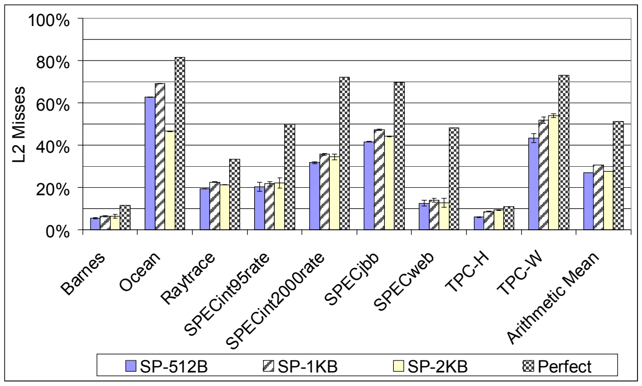

Figure 7. L2 miss ratio with Stealth Prefetching.

Figure 8 illustrates the rate of L2 misses per instruction. Bars for the baseline miss rate and the miss rate with CGCT are shown for comparison (in this graph CGCT is implemented with a RCA and uses 512B regions). Note that the miss rate with CGCT alone is slightly higher than the baseline miss rate due to the overhead of the RCA maintaining inclusion. Stealth Prefetching is based on CGCT, and this offsets some of the benefit.

Again, Ocean has the biggest reduction, followed by TPC-W and SPECjbb. These applications have relatively high miss rates and should gain performance from the reduction in L2 misses. On the other hand, SPECint2000rate does not have a large L2 miss rate, and does not benefit as much.

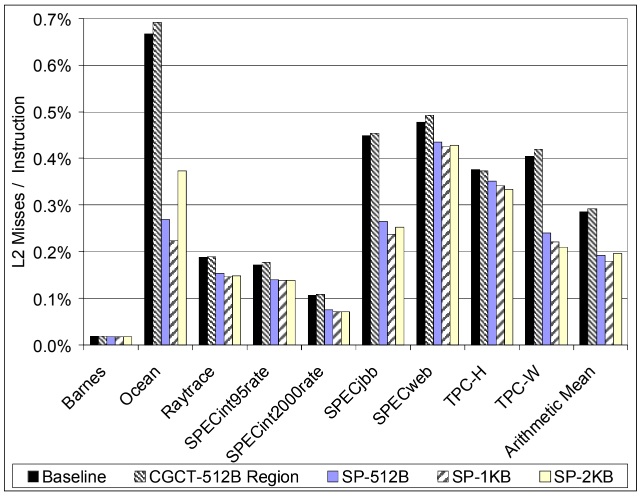

Figure 8. L2 Misses per instruction with Stealth Prefetching.

Figure 9 shows the percentage of line prefetches that were timely for each benchmark. That is, when there is an L2 miss, the data is present in the SDPB, and not on its way to the SDPB from memory. As the region size increases, so does the timeliness of prefetches. The average percentage of SDPB hits for which the data is present increases from 78% to 93% as the region size is increased from 512B to 2KB.

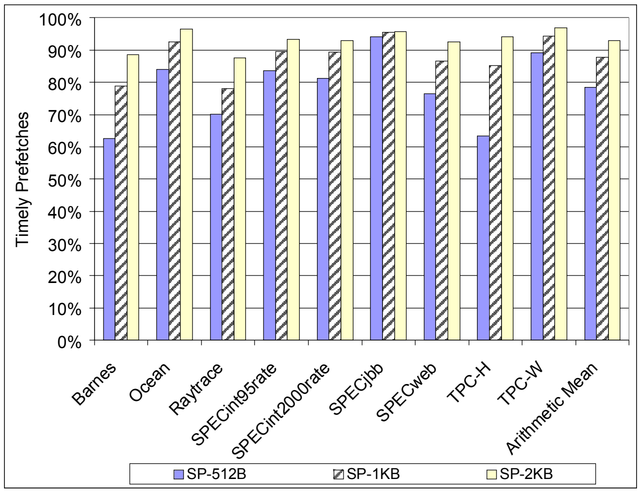

Figure 9. Timeliness of Stealth Prefetching.

5.2 Performance Improvement

Figure 10 contains the normalized execution time of each benchmark, starting with the baseline and CGCT with 512B regions, followed by Stealth Prefetching with 512B-2KB regions. For Stealth Prefetching and 1KB regions; execution time is on average 17% lower than for the baseline system and an additional 10% lower than for CGCT alone. As expected, Ocean and TPC-W have the largest execution time reductions of 46% and 38%, respectively.

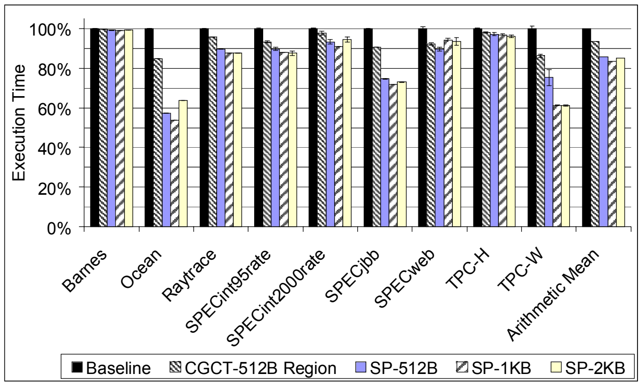

Figure 10. Execution time comparison.

Figure 11 displays the system speedup for each benchmark for Stealth Prefetching. The execution time reductions result in an average system speedup of 20% for Stealth Prefetching with 1KB regions. Similar speedups of 16% and 17% are achieved for 512B and 2KB regions, respectively.

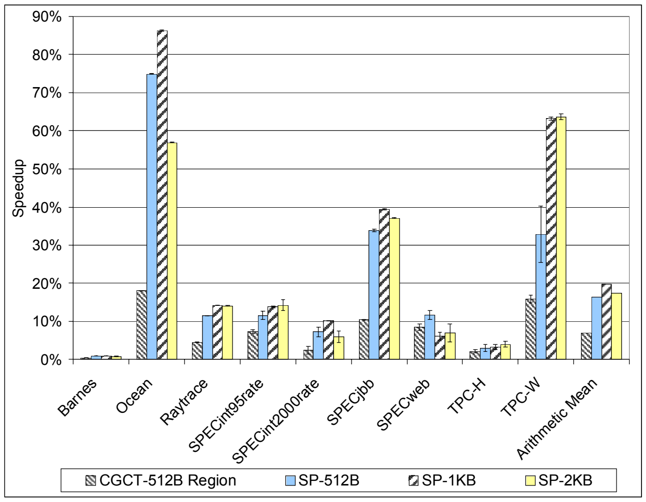

Figure 11. Stealth Prefetching speedup.

5.3 Data Utilization and Traffic

As with all prefetching techniques, some prefetched data is useless (or not used before replacement), and increases the amount of data fetched from memory. The normalized data traffic for each benchmark is shown in Figure 12 (measured as the total data messages over the execution of the application, normalized to that of the baseline). On average there is 25% more data traffic over the execution of these applications for 1KB regions. The additional data traffic increases with region size, and decreases with increasing threshold and SDPB size. Larger regions result in smaller numbers of larger sectors in the SDPB and prefetch more useless data than smaller ones. These two effects lead to less effective use of buffer space and lower utilization of prefetched data. For SPECjbb running with 2KB regions the effect on data traffic is severe; the data traffic is increased 75%.

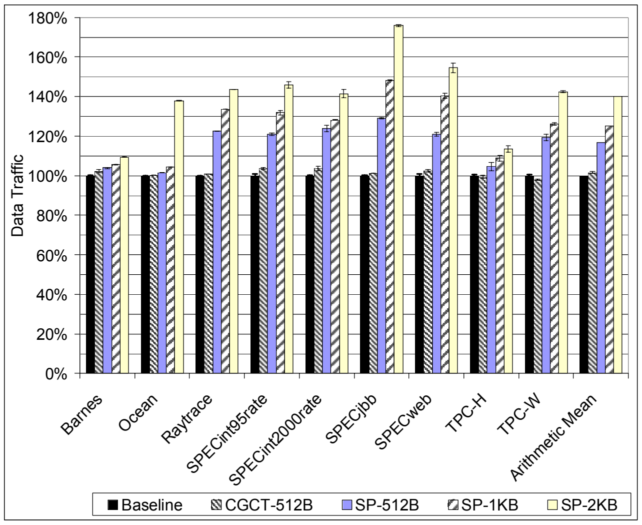

Figure 12. Normalized data traffic.

Figure 13 shows the average utilization of data in the SDPB. On average, 43-63% of the lines prefetched are used, depending on the region size. The utilization is just over 56% for 1KB regions, compared to the two-thirds utilization predicted by Figure 6. Many lines are replaced before they can be used, and hence data utilization will increase both with higher prefetching thresholds and larger SDPB sizes by allowing useful lines to stay in the SDPB longer.

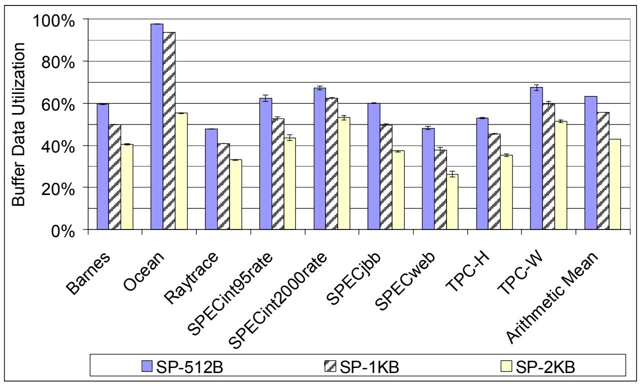

Figure 13. Stealth Data Prefetch Buffer utilization.

6. Conclusions and Future Work

Stealth Prefetching can significantly improve performance in a broadcast-based shared-memory multiprocessor system. SP does not increase broadcast traffic (in fact there is a large reduction due to CGCT), interfere with other processors sharing/modifying data, prevent other processors from obtaining exclusive copies of lines, nor pollute the cache. In addition, the implementation impact is manageable: only a small prefetch data buffer, a bit-mask in direct request packets, and minor medications to the RCAs and memory controllers and are needed.

One source of lost potential in this study was replacements in the SDPB. Prefetched lines were sometimes replaced before being accessed by the processor, suggesting that a larger buffer may be needed. In future work, we will investigate the impact of different SDPB sizes and organizations. One possibility is to implement a buffer with a sectored array for state information and a separate non-sectored array for data, allowing storage to be allocated only for lines actually prefetched, and not for an entire region.

Another source of lost potential we have identified is regions that become shared after the prefetch has taken place. In this paper, data in the SDPB was only if the processor had access permissions to the region. However, this is not necessary for correctness. Provided that a line is invalidated from the SDPB when there is an external request for the line, and that all the prefetched lines for a region are invalidated if the region is evicted/invalidated from the RCA, the lines in the SDPB may continue to be used after a region becomes externally-shared. The entire region was non-shared when the lines were obtained from memory, and the lines are still non-shared if they have not yet been accessed by another processor. Therefore, they can be treated as exclusive copies of lines that were prefetched at the time the region was prefetched, and used without first consulting the RCA. This will increase the utilization of prefetched data and reduce wasted bandwidth.

In this study, Stealth Prefetching used regions of the same size used by CGCT for tracking coherence permissions. To prefetch farther ahead, a larger region size would be needed. However, prior work showed that larger regions are not always better due to false-sharing of the region [15, 16]. Therefore, an interesting avenue of future would be to investigate prefetching of multiple non-shared regions at once; decoupling region size and prefetch granularity. The RCA can be sectored to improve space-efficiency and allow state for multiple adjacent regions to be obtained with a single access.

Finally, when regions were initially prefetched, all of the remaining lines in the region were prefetched. There is potential to prefetch only portions of a region, prefetch only addresses forward in memory of the lines initially touched, or infer a direction in memory to prefetch based on the order in which lines were first accessed.

Acknowledgements

We thank our anonymous reviewers for their comments and suggestions. We also thank Ibrahim Hur, Candy Jelak, Pattabi Seshadri, Dana Vantrease for help proofreading the final version. This research was supported in part by the National Science Foundation under grant CCR-0311361, CCR-0133437, CCR-0429854, grants from IBM and Intel. During the course of this research Jason Cantin was supported by a graduate fellowship from IBM.

References

[1] Charlesworth, A. *The Sun Fireplane System Interconnect*. In 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] Lin, W-F., Burger, D., *Reducing DRAM Latencies with an Integrated Memory Hierarchy Design*. In Proceedings of the 28th International Symposium on High-Performance Computer Architecture (HPCA), 2001.

[6] Lin, W-F., Burger, D., and Puzak, T., *Filtering Superfluous Prefetches using Density Vectors*. In Proceedings of the International Conference on Computer Design: VLSI in Computers & Processors (ICCD), 2001.

[7] Wang, Z., Burger, D., McKinley, K., Reinhardt, S., and Weems, C., *Guided Region Prefetching: A Cooperative Hardware/Software Approach*. In Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA), 2003.

[8] Wang, Z., McKinley, K., and Burger, D., *Combining Software/Hardware Prefetching and Cache Replacement*. IBM Austin Center for Advanced Studies Conference (CAS), 2004.

[9] Nesbit, K., and Smith, J., *Prefetching Using a Global History Buffer*. Proceedings of the 10th Annual International Symposium on High Performance Computer Architecture, 2004.

[10] Hughes, C., and Adve, S., *Memory-side Prefetching for Linked Data Structures for Processor-In-Memory Systems*. IEEE Journal on Parallel and Distributed Systems, Volume 65, Issue 4, 2005.

[11] Somogyi, S., Wenisch, T., Ailamaki, A., Falsafi, B., and Moshovos, A. *Spatial Memory Streaming*. Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA), 2006.

[12] Jerger, N., Hill, E., and Lipasti, M., *Friendly Fire: Understanding the Effects of Multiprocessor Prefetching*. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2006.

[13] Moshovos, A., *Exploiting Coarse-Grain Non-Shared Regions in Snoopy Coherent Multiprocessors*. Computer Engineering Group Technical Report, University of Toronto, December 2003.

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

[15] Cantin, J., Lipasti, M., and Smith J., *Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking*. In Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA), 2005.

[16] Cantin, J., Moshovos, A., Lipasti, M., Smith, J., and Falsafi, B., "*Coarse-Grain Coherence Tracking: RegionScout and Region Coherence Arrays*". IEEE Micro Special Issue on Top Picks from 2005 Computer Architecture Conferences, Jan-Feb 2006.

[17] Jouppi, N., *Improving Direct-Mapped Cache Performance by the Addition of a Small, Fully-Associative Cache and Prefetch Buffers*. In Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA), 1990.

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

[19] Keller, T., Maynard, A., Simpson, R., and Bohrer, P., *Simos-ppc Full System Simulator*.

http://www.cs.utexas.edu/users/cart/simOS.

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

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

[22] 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.