In Proceedings of the 33rd Annual International Symposium on Computer Architecture, June 2006

Spatial Memory Streaming

Stephen Somogyi, Thomas F. Wenisch,
Anastassia Ailamaki, Babak Falsafi and Andreas Moshovos†
http://www.ece.cmu.edu/~stems

Computer Architecture Laboratory (CALCM)
Carnegie Mellon University

†Dept. of Electrical & Computer Engineering
University of Toronto

Abstract

Prior research indicates that there is much spatial variation in applications' memory access patterns. Modern memory systems, however, use small fixed-size cache blocks and as such cannot exploit the variation. Increasing the block size would not only prohibitively increase pin and interconnect bandwidth demands, but also increase the likelihood of false sharing in shared-memory multiprocessors.

In this paper, we show that memory accesses in commercial workloads often exhibit repetitive layouts that span large memory regions (e.g., several kB), and these accesses recur in patterns that are predictable through code-based correlation. We propose Spatial Memory Streaming, a practical on-chip hardware technique that identifies code-correlated spatial access patterns and streams predicted blocks to the primary cache ahead of demand misses. Using cycle-accurate full-system multiprocessor simulation of commercial and scientific applications, we demonstrate that Spatial Memory Streaming can on average predict 58% of L1 and 65% of off-chip misses, for a mean performance improvement of 37% and at best 307%.

1. Introduction

Poor memory system behavior limits the performance of commercial applications in high-end servers. One-half to two-thirds of execution time in commercial online transaction processing (OLTP), decision support system (DSS), and web server workloads is spent on memory system-related stalls [2, 3,13,21,26,30]. These workloads challenge system designers because they rely on complex algorithms and data structures, have large code footprints, operate on data sets that greatly exceed physical memory, and use sophisticated fine-grain synchronization to maximize concurrency.

Microarchitectural innovations, such as out-of-order execution, non-blocking caches, and run-ahead execution [19], improve memory system performance by increasing the off-chip memory-level parallelism (MLP). However, to discover parallel misses, these approaches must correctly predict and execute the instruction stream, which restricts the depth of the instruction window they can explore. In OLTP and web applications, these innovations provide limited benefit because of frequent, long chains of dependent memory accesses [21,30]. Although modern servers provide copious memory bandwidth [7], the memory system remains underutilized because these dependence chains severely limit available MLP—one study reports an average of just 1.3 parallel off-chip misses for these applications on a current-generation out-of-order system [6]. Enhancing the parallelism and hiding the latency of memory accesses are the keys to server performance improvement.

Although computer architects have demonstrated great success in improving MLP and hiding memory latency in desktop and scientific applications through memory prefetching or streaming (e.g., [18,20,24,25,29]), few studies have demonstrated success at improving memory system performance for commercial applications. Instruction stream buffers reduce primary instruction cache stalls [21]. Software prefetching can accelerate certain database operations, such as hash joins [5]. Temporal streaming reduces coherence stalls by streaming repetitive, temporally-correlated coherence miss sequences [30]. However, all of these approaches target only limited classes of memory accesses and a significant fraction of memory stalls remain exposed.

Despite their complexity, commercial applications nonetheless utilize data structures with repetitive layouts and access patterns—such as database buffer pool pages or network packet headers. As these applications traverse their data sets, recurring patterns emerge in the relative offsets of accessed data. Unfortunately, these accesses are frequently non-contiguous and do not follow a constant stride (e.g., binary search in a B-tree). Because sparse patterns may span large regions (e.g., an operating system page), we use the term *spatial correlation* rather than spatial locality to describe the relationship among accesses. Increasing cache block size to capture spatial correlation leads to inefficient storage and bandwidth utilization.

Past research on uniprocessor systems has shown that spatial correlation can be predicted in hardware by correlating patterns with the code and/or data address that initiates the pattern [4,17]. Whereas existing spatial pattern prefetching designs are effective for desktop/engineering applications [4], the only practical implementation evaluated on server workloads provides less than 20% miss rate reduction [17].

In this paper, we reconsider prediction and streaming of spatially-correlated access patterns to improve MLP and to hide the long latencies of secondary cache and off-chip memory accesses. Our design, *Spatial Memory Streaming (SMS)*, targets commercial server applications and can reduce both

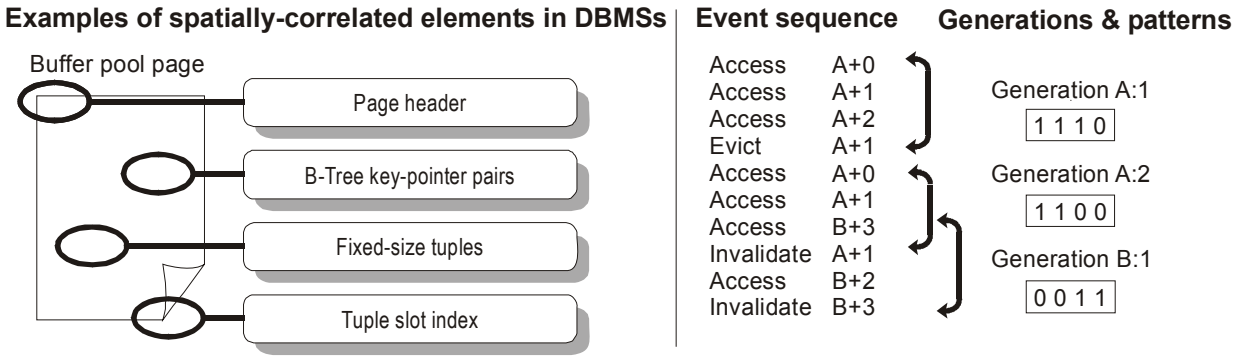

FIGURE 1. Examples of spatial correlation and spatial region generations. The left figure shows example sources of spatial correlation in databases. The right figure illustrates an event sequence and the corresponding spatial region generations and patterns.

primary cache and off-chip misses in multiprocessor servers. SMS exploits repetitive access patterns to predict and stream blocks to the primary cache ahead of demand misses. We evaluate SMS through a combination of trace-based and cycle-accurate full-system simulation of scientific and commercial applications. This work demonstrates:

- **Effective spatial correlation and prediction.** Contrary to previous findings [17], address-based correlation is not needed to predict the access stream of commercial workloads. Instead, we show a strong correlation between code and access patterns, which SMS exploits to predict patterns even for previously-unvisited addresses. Because there are far fewer distinct code sequences than data addresses, SMS provides nearly four times the prediction coverage of an address-based predictor with equivalent storage.
- **Accurate tracking of spatial correlation.** We show that the cache-coupled structures used in previous work ([4,17]) are suboptimal for observing spatial correlation. Accesses to multiple independent patterns are frequently interleaved, which induce conflict behavior in prior detection structures. Instead, we propose a decoupled detection structure that identifies fewer and denser patterns, halving predictor storage requirements and increasing coverage by up to 20%.
- **Performance enhancement.** For commercial workloads, we show that SMS predicts on average 55% and at best 78% of off-chip read misses, providing a mean speedup of 1.22 and at best 1.48 over a system without SMS. In contrast, the global history buffer (GHB) [20], the best proposed prefetcher for desktop/engineering applications, eliminates only 30% of off-chip misses on average and 62% at best. In scientific applications, SMS matches GHB's coverage to eliminate on average 81% of off-chip misses, and yields speedups ranging from 1.26 to 4.07.

The remainder of this paper is organized as follows. We describe Spatial Memory Streaming in Section 2 and present the details of our hardware implementation in Section 3. In Section 4, we evaluate our design and present performance results. We discuss related work in Section 5 and conclude in Section 6.

2. Spatial Memory Streaming

Spatial Memory Streaming (SMS) improves the performance of scientific and commercial server applications by exploiting spatial relationships among data beyond a single cache block.

In choosing a cache block size, system designers are forced to balance the competing concerns of spatial locality, transfer latency, cache storage utilization, memory/processor pin bandwidth utilization, and false sharing. Typically, the optimal cache block size sacrifices opportunity to exploit spatial locality for dense data structures to avoid excessive bandwidth overheads for sparse data structures. For simple data structures, such as arrays, spatial relationships can be exploited through simple prefetching schemes, such as stride prefetching [24].

Commercial applications exhibit complex access patterns that are not amenable to simple prefetching or streaming schemes. Nevertheless, data structures in these applications frequently exhibit spatial relationships among cache blocks. For example, in databases, pages in the buffer pool share common structural elements, such as a log serial number in the page header and a slot index that indicates tuple offsets in the page footer, that are always accessed prior to scanning/modifying the page. In web servers, packet headers and trailers have arbitrarily complex but fixed structure. Further examples appear in Figure 1 (left). Although accesses within these structures may be non-contiguous, they nonetheless exhibit recurring patterns in relative addresses. We call the relationship between these accesses *spatial correlation*.

SMS extracts spatially-correlated access patterns at runtime and predicts future accesses using these patterns. SMS then streams the predicted cache blocks into the processor's primary cache as rapidly as allowed by available resources and bandwidth, thereby increasing memory level parallelism and hiding lower-level cache and off-chip access latencies.

2.1. Spatial Patterns and Generations

We formalize our notion of spatial correlation similar to prior studies of spatial footprints [4,17]. We define a *spatial region* as a fixed-size portion of the system's address space, consisting of multiple consecutive cache blocks. A *spatial*

region generation is the time interval over which SMS records accesses within a spatial region. We call the first access in a spatial region generation the *trigger access*. A *spatial pattern* is a bit vector representing the set of blocks in a region accessed during a spatial region generation. Thus, a spatial pattern captures the layout of cache blocks accessed near one another in time. Upon a trigger access, SMS predicts the spatial pattern that will be accessed over the course of the spatial region generation.

The precise interval over which a spatial region generation is defined can significantly impact the accuracy and coverage of spatial patterns [17]. A generation must be defined to ensure that, when SMS streams blocks into the cache upon a future trigger access, no predicted block will be evicted or invalidated prior to its use. Therefore, we choose the interval from the trigger access until any block accessed during the generation is removed from the processor's primary cache by replacement or invalidation. A subsequent access to any block in the region is the trigger access for a new generation. This definition ensures that the set of blocks accessed during a generation were simultaneously present in the cache. Figure 1 (right) shows an example of three spatial region generations and their corresponding patterns.

2.2. Identifying Recurring Spatial Patterns

Upon a trigger access, SMS predicts the subset of blocks within the region that are spatially correlated and therefore likely to be accessed. Thus, a key problem in SMS is finding a prediction index that is strongly correlated to recurring spatial patterns.

Spatial correlation arises because of repetition and regularity in the layout and access patterns of data structures. For instance, spatial correlation can arise because several variables or fields of an aggregate are frequently accessed together. In this case, the spatial pattern correlates to the address of the trigger access, because the address identifies the data structure. Spatial correlation can also arise because a data structure traversal recurs or has a regular structure. In this case, the spatial pattern will correlate to the code (program counter values) executing the traversal.

A variety of prediction indices have been investigated in the literature. All prior studies found that combining both the address and program counter to construct an index consistently provides the most accurate predictions when correlation table storage is unbounded [4,17]. By combining both quantities, which we call PC+address indexing, a predictor generates distinct patterns when multiple code sequences lead to different traversals of the same data structure. However, this prediction index requires predictor storage that scales with data set size, and predictor coverage drops precipitously with realistic storage constraints.

For SPEC CPU 2000 applications, PC+address indexing can be approximated by combining the PC with a *spatial region offset* [4,17]. The spatial region offset of a data address is the distance, in cache blocks, of the address from the start of the spatial region. The spatial region offset allows the predictor to distinguish repetitive patterns generated by the same code fragment that only differ in their alignment relative to spatial region boundaries. PC+offset indexing considerably reduces prediction table storage requirements because applications have far fewer distinct miss PCs than miss addresses.

We observe that PC+offset indexing, in addition to its storage savings, is fundamentally more powerful than address-based indexing because it can eliminate cold misses. When a code sequence repeats the same access pattern over a large data set, the PC-correlated spatial patterns learned at the start of the access sequence will provide accurate predictions for data that have never previously been visited. Database scan and join operations, which dominate the execution of decision support queries [23], contain long repetitive access patterns that visit data only once. In these applications, PC+offset indexing substantially outperforms address-based schemes.

3. Design

We now describe our design for Spatial Memory Streaming. Unlike prior proposals, we target our design at high-performance commercial server applications in a multiprocessor context. Our most significant departure from prior designs is that those designs target decoupled sectored [22] or sub-blocked caches. Integrating spatial pattern prediction with such caches simplifies the hardware design because the training structures for spatial region accesses can be integrated with the sub-blocked cache tag array. However, interleaved accesses to different spatial regions cause conflict behavior within the sub-blocked tags, fragmenting spatial region generations and reducing the accuracy of observed patterns. Therefore, we design SMS to integrate with a traditional cache hierarchy.

SMS comprises two hardware structures. The *active generation table* records spatial patterns as the processor accesses spatial regions and trains the predictor. The *pattern history table* stores previously-observed spatial patterns, and is accessed at the start of each spatial region generation to predict the pattern of future accesses. The next two subsections describe these structures and their operation.

3.1. Observing Spatial Patterns

Spatial Memory Streaming learns spatial patterns by recording which blocks are accessed over the course of a spatial region generation in the active generation table (AGT). When a spatial region generation begins, SMS allocates an entry in the AGT. As cache blocks are accessed, SMS updates the recorded pattern in the AGT. At the end of a generation (eviction/invalidation of any block accessed during the generation), the AGT transfers the spatial pattern to the history table and the AGT entry is freed.

Although the AGT is logically a single table, we implement it as two content addressable memories, the accumulation table and the filter table, to reduce the size of the associative search within each memory and the overall size of the structure. Because the AGT processes each L1 data access, it is necessary that both tables be able to match the L1 data access bandwidth. The AGT is not on the L1 data access critical path, and thus does not impact cache access latency.

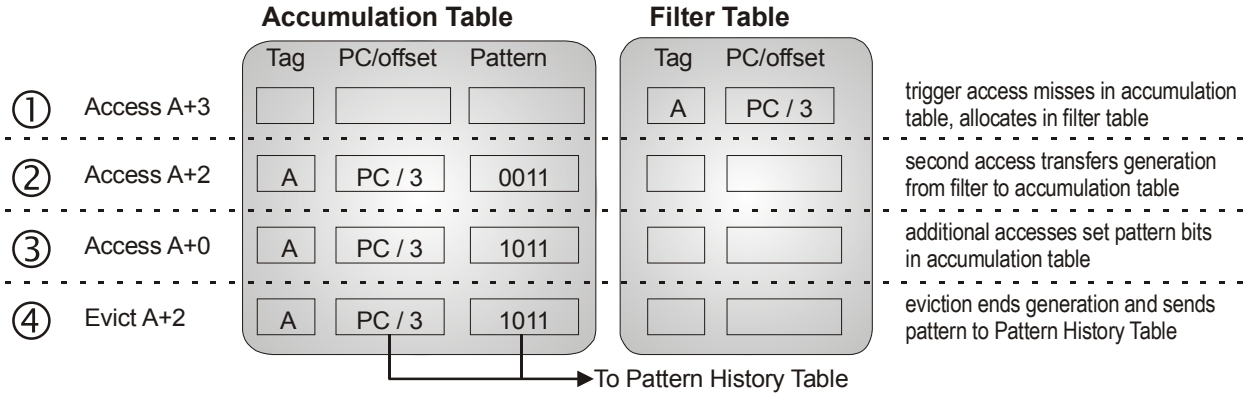

FIGURE 2. Active Generation Table. The AGT consists of an accumulation table and a filter table. The figure illustrates the actions taken over the course of one spatial region generation.

Spatial patterns are recorded in the accumulation table. Entries in the accumulation table are tagged by the *spatial region tag*, the high order bits of the region base address. Each entry stores the PC and spatial region offset of the trigger access, and a spatial pattern bit vector indicating which blocks have been accessed during the generation.

New spatial region generations are initially allocated in the filter table. The filter table records the spatial region tag, and the PC and spatial region offset of the trigger access, for spatial regions that have had only a single access in their current generation. A significant minority of spatial region generations never have a second block accessed; there is no benefit to predicting these generations because the only access is the trigger access. By restricting such generations to the filter table, SMS reduces pressure on the accumulation table.

The detailed operation of the AGT is depicted in Figure 2. Each L1 access first searches the accumulation table. If a matching entry is found, the spatial pattern bit corresponding to the accessed block is set. Otherwise, the access searches for its tag in the filter table. If no match is found, this access is the trigger access for a new spatial region generation and a new entry is allocated in the filter table (step 1 in Figure 2). If an access matches in the filter table, its spatial region offset is compared to the recorded offset. If the offsets differ, then this block is the second distinct cache block accessed within the generation, and the entry in the filter table is transferred to the accumulation table (step 2). Additional accesses to the region set corresponding bits in the pattern (step 3).

Spatial region generations end with an eviction or invalidation (step 4). Upon these events, both the filter table and accumulation table are searched for the corresponding spatial region tag. (Note that this search requires reading the tags of replaced cache blocks even if the replaced block is clean). A matching entry in the filter table is discarded because it represents a generation with only a trigger access. A matching entry in the accumulation table is transferred to the pattern history table. If either table is full when a new entry must be allocated, a victim entry is selected and the corresponding generation is terminated (i.e., the entry is dropped from the filter table or transferred from the accumulation table to the pattern history table). In Section 4.5, we observe that small (e.g., 32- or 64-entry) accumulation and filter tables make this occurrence rare.

3.2. Predicting Spatial Patterns

SMS uses a pattern history table (PHT) for long-term storage of spatial patterns and to predict the pattern of blocks that will be accessed during each spatial region generation. The implementation of the PHT and the address stream prediction process is depicted in Figure 3. The PHT is organized as a set-associative structure similar to a cache. The PHT is accessed using a prediction index constructed from the PC and spatial region offset of the trigger access for a generation.

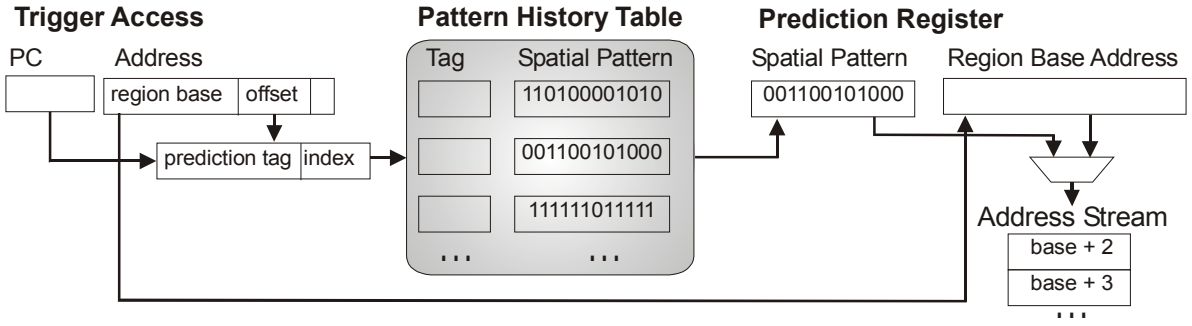

FIGURE 3. Pattern History Table and prediction process. Upon a trigger access that matches in the PHT, the region base address and spatial pattern are transferred to a prediction register, beginning the streaming process.

TABLE 1. System and application parameters.

Processing Nodes	UltraSPARC III ISA 4 GHz 8-stage pipeline; out-of-order 8-wide dispatch / retirement 256-entry ROB, LSQ; 64-entry store buffer
L1 Caches	Split I/D, 64KB 2-way, 2-cycle load-to-use 4 ports, 32 MSHRs, 16 SMS stream requests
L2 Cache	Unified, 8MB 8-way, 25-cycle hit latency 1 port, 32 MSHRs
Main Memory	3 GB total memory 60 ns access latency 64 banks per node 64-byte coherence unit
Protocol Controller	1 GHz microcoded controller 64 transaction contexts
Interconnect	4x4 2D torus 25 ns latency per hop 128 GB/s peak bisection bandwidth

Online Transaction Processing (TPC-C)	
Oracle	100 warehouses (10 GB), 16 clients, 1.4 GB SGA
DB2	100 warehouses (10 GB), 64 clients, 450 MB buffer pool
Decision Support (TPC-H on DB2)	
Qry 1	Scan-dominated, 450 MB buffer pool
Qry 2	Join-dominated, 450 MB buffer pool
Qry 16	Join-dominated, 450 MB buffer pool
Qry 17	Balanced scan-join, 450 MB buffer pool
Web Server	
Apache	16K connections, FastCGI, worker threading model
Zeus	16K connections, FastCGI
Scientific	
em3d	3M nodes, degree 2, span 5, 15% remote
ocean	1026x1026 grid, 9600s relaxations, 20K res., err tol 1e-07
sparse	4096x4096 matrix

Each entry in the PHT stores the spatial pattern that was accumulated in the AGT.

Upon a trigger access, SMS consults the PHT to predict which blocks will be accessed during the generation. If an entry in the PHT is found, the spatial region's base address and the spatial pattern are copied to one of several prediction registers. As SMS streams each block predicted by the pattern into the primary cache, it clears the corresponding bit in the prediction register. The register is freed when its entire pattern has been cleared. If multiple prediction registers are active, SMS requests blocks from each prediction register in a round-robin fashion. SMS stream requests behave like read requests in the cache coherence protocol.

4. Results

We evaluate SMS using a combination of trace-driven and cycle-accurate full-system simulation of a shared-memory multiprocessor using FLEXUS [14]. FLEXUS can execute unmodified commercial applications and operating systems. FLEXUS extends the *Virtutech Simics* functional simulator with cycle-accurate models of an out-of-order processor core, cache hierarchy, protocol controllers and interconnect. We simulate a 16-processor directory-based shared-memory multiprocessor system running *Solaris 8*. We employ a wait-free implementation of the total store order memory consistency model [1,10]. We perform speculative load and store prefetching [9], and speculatively relax memory ordering constraints at memory barrier and atomic read-modify-write memory operations [10]. We list other relevant parameters of our system model in Table 1 (left).

Table 1 (right) enumerates our commercial and scientific application suite. We include the TPC-C v3.0 OLTP workload on two commercial database management systems, *IBM DB2 v8 ESE*, and *Oracle 10g Enterprise Database Server*. We select four queries from the TPC-H DSS workload based on the categorization in [23]: one scan-dominated query, two join-dominated queries, and one query exhibiting mixed behavior. All four DSS queries are run on DB2. We evaluate web server performance with the SPECweb99 benchmark on *Apache HTTP Server v2.0* and *Zeus Web Server v4.3*. We drive the web servers using a separate client system and a high-bandwidth link tuned to ensure that the server system is fully saturated (client activity is not included in trace or timing results). Finally, we include three scientific applications to provide a frame of reference for our commercial application results.

Our trace-based analyses use memory access traces collected from FLEXUS with in-order execution, no memory system stalls, and a fixed IPC of 1.0. For OLTP and web workloads, we warm main memory with functional simulation for at least 5000 transactions (or web requests) prior to starting traces, and then trace at least 1000 transactions. For DSS queries, we analyze traces of over three billion total instructions taken from the query execution at steady-state. We have experimentally verified that varying trace start location has minimal impact on simulation results. For scientific applications, we analyze traces of five to ten iterations. We use half of each trace for warm-up prior to collecting experimental results. All results prior to Section 4.7 use this trace-based methodology.

For cycle-accurate simulations, we use a sampling approach developed in accordance with SMARTS [32]. Our samples are drawn over an interval of 10 to 30 seconds of simulated time (as observed by the operating system in functional simulation) for OLTP and web applications, over the complete query execution for DSS, and over a single iteration for scientific applications. We show 95% confidence intervals that target ±5% error on change in performance, using paired-measurement sampling [31]. We launch measurements from checkpoints with warmed caches, branch predictors, and predictor table state, then run for 100,000 cycles to warm queue and interconnect state prior to collecting measurements of 50,000 cycles. We use the aggregate number of user instructions committed per cycle (i.e., committed user instructions summed over the 16 processors divided by total elapsed

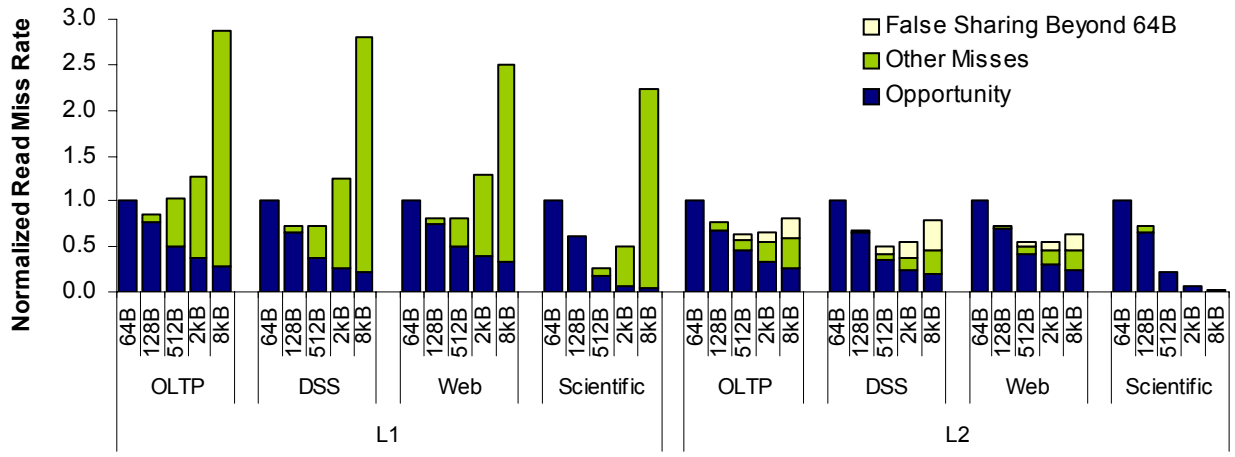

FIGURE 4. L1 and L2 (off-chip) miss rates versus block/region size. Opportunity represents an oracle spatial predictor that incurs one miss per spatial region generation.

cycles) as our performance metric, which is proportional to overall system throughput [30].

4.1. Spatial Characterization

We begin by quantifying the spatial characteristics of our application suite and identifying the maximum opportunity to reduce miss rates with SMS. We show that there is substantial spatial correlation over regions as large as the operating system page size (8kB). However, an increased cache block size cannot exploit this correlation because of increased conflict misses, false sharing, and inefficient bandwidth utilization. No single cache block size can capture spatial correlation efficiently because access density varies within and across applications. SMS does not suffer these inefficiencies because it tracks spatial correlation at fine granularity.

We quantify the opportunity for SMS to exploit spatial correlation across a range of region sizes, and compare against the effectiveness of increasing cache block size in Figure 4. To assess opportunity, at each region size, we consider an oracle predictor that incurs only one miss per spatial region generation (labelled "opportunity"). We also show the miss rate achieved by a cache with block size equal to the region size. (We hold cache capacity fixed across all region/block sizes). For block sizes larger than 64B, we separate misses caused by false sharing (labelled "false sharing beyond 64B") from all other misses (labelled "other misses"). We report results in terms of misses per instruction, normalized to a cache with 64B blocks and no predictor.

Our oracle study demonstrates substantial opportunity for SMS to eliminate read misses. Across applications and cache hierarchy levels, SMS opportunity increases as spatial regions are extended to the OS page size.

Increased cache block size leads to drastic increases in L1 miss rates because of conflict behavior. The commercial workloads use only a subset of the data in large regions and interleave accesses across regions. Thus, as the cache block size increases, conflicts increase, and the effective capacity of the L1 cache is reduced, leading to a sharp increase in miss rate with block sizes beyond 512B. The data sets of the scientific applications are more tightly packed, but nevertheless suffer from similar conflict behavior.

The larger capacity of L2 reduces the prevalence of conflict effects as compared to L1. However, commercial workloads instead incur misses from false sharing, which accounts for 26%–42% of L2 misses at the 8kB block size.

The inefficient bandwidth utilization of larger blocks makes it unclear if even block sizes of 512B, despite lower miss rates, can improve performance over 64B blocks at any hierarchy level. Unless data is densely packed, as in the scientific applications, larger block sizes lead to the transfer of more unused data. In the commercial applications, bandwidth efficiency drops exponentially as block size increases above 512B.

Huh and co-authors demonstrate that the latency penalty of false sharing can be eliminated through coherence decoupling—speculative use of incoherent data [15]. However, even if false sharing is eliminated, true sharing and replacement misses nonetheless result in nearly double the L2 miss rate of the oracle opportunity at 8kB blocks. Furthermore, coherence decoupling does not eliminate bandwidth wasted by false sharing, and therefore cannot scale to the same region sizes as SMS.

The root cause of the inefficiency of large cache blocks is the variability of memory access density within and across applications. We quantify memory access density as the fraction of cache misses occurring in spatial region generations that contain a particular number of misses. Figure 5 presents a breakdown of memory access density for each application for a 2kB region size (we establish 2kB as the best choice for region size in Section 4.4). For example, in OLTP-DB2, 22% of L1 misses come from spatial generations in which between four and seven blocks are missed upon during the generation. With the exception of ocean and sparse, all applications exhibit wide variations in their memory access density at both L1 and L2. Thus, no single block size can simultaneously exploit the available spatial correlation while using bandwidth and storage efficiently.

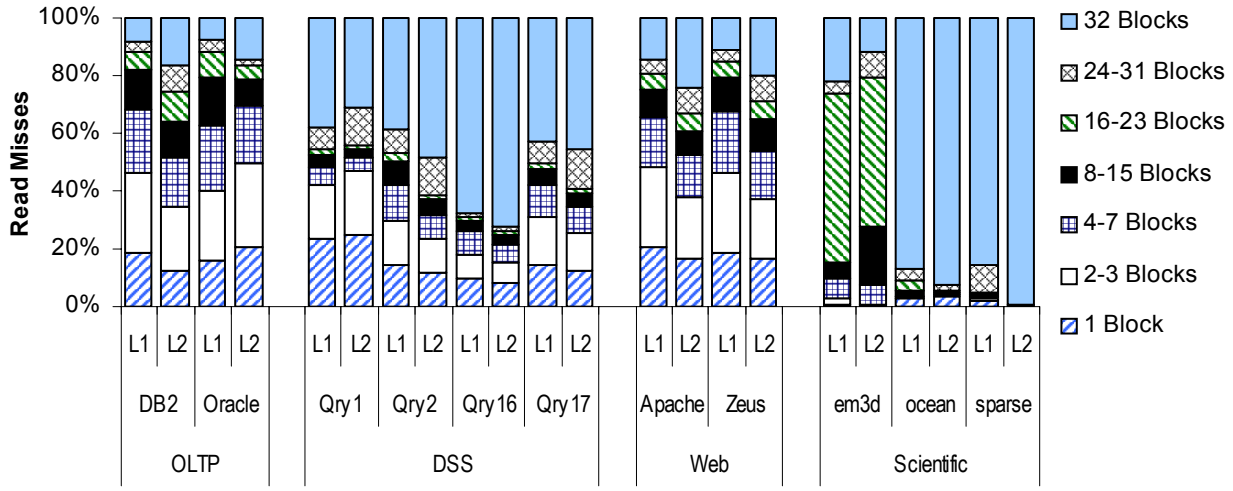

FIGURE 5. Memory access density. Each segment represents the percentage of L1 or L2 misses from generations of the indicated density (i.e., the number of blocks in the 2kB spatial region that incur misses during the generation).

Because SMS learns and predicts spatial patterns over large regions at fine granularity, SMS can approach the miss rates indicated by our opportunity study without the inefficiencies of large blocks. SMS fetches only the 64B blocks within a region that are likely to be used, and therefore does not incur the conflict, false sharing, or bandwidth overhead of larger blocks. Our opportunity results for L1 indicate that accurate spatial pattern prediction allows SMS to deliver blocks directly into L1, despite its small capacity.

4.2. Indexing

Prior studies of spatial predictors [4,17] advocate predictor indices that include address information. In this section, we show that PC+offset indexing yields the same or significantly higher coverage than address-based indices, as well as lower storage requirements.

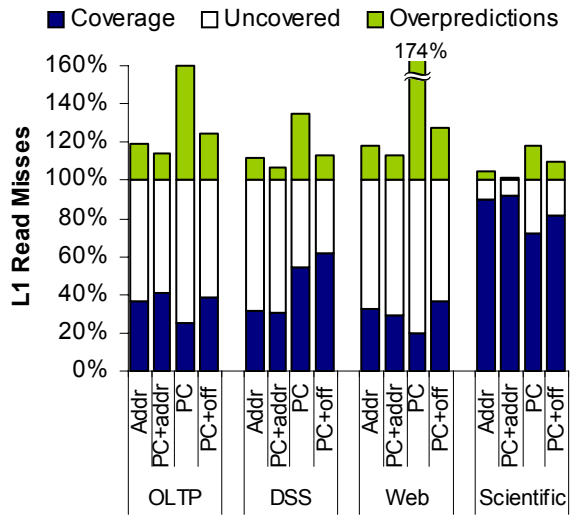

FIGURE 6. Index comparison. The index type is given below each bar. PHT size is unbounded.

We compare the Address, PC+address, PC, and PC+offset indexing schemes in Figure 6, using an infinite PHT to assess the true opportunity without regard to storage limitations. Coverage represents the fraction of L1 read misses that are eliminated by SMS. Overpredictions represent blocks that are fetched but not used prior to eviction or invalidation, and thus waste bandwidth. Overpredictions can also cause cache pollution; this effect is implicitly taken into account because the additional misses are categorized as uncovered.

In OLTP and web applications, a majority of spatially-correlated accesses arise from heavily-visited code sequences and data structures. Hence, both data addresses and PCs correlate to similar spatial patterns, and the Address, PC+address, and PC+offset indexing schemes perform similarly. PC indexing (without any address information) is less accurate because it cannot distinguish among distinct access patterns to different data structures by the same code (e.g.,

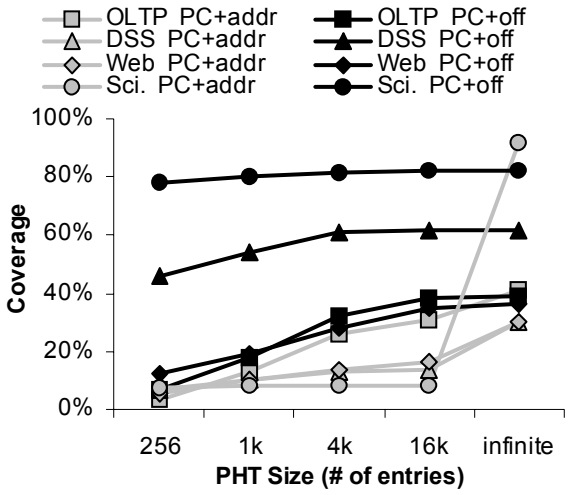

FIGURE 7. PHT storage sensitivity for PC+address and PC+offset indexing. The finite PHTs are 16-way set-associative.

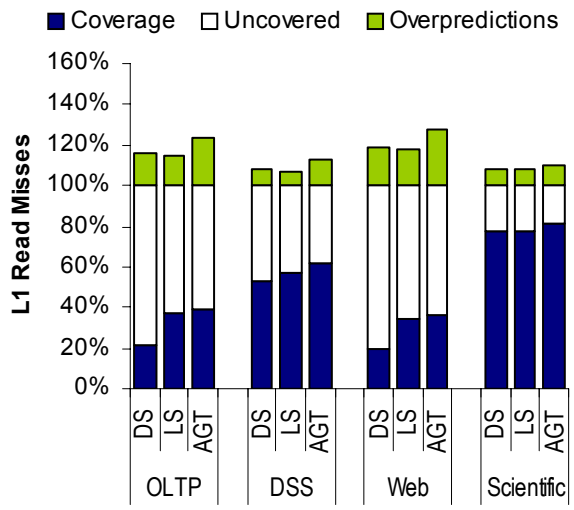

FIGURE 8. Comparison of training structures. DS=Decoupled Sectored. LS=Logical Sectored. AGT=Active Generation Table. PHT size is unbounded.

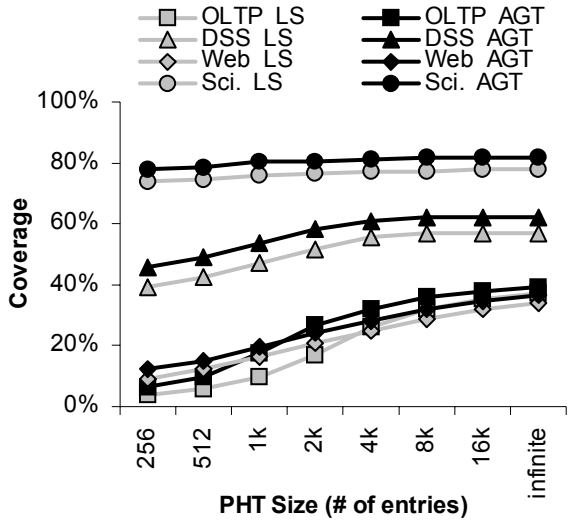

FIGURE 9. PHT storage sensitivity. LS=Logical Sectored. AGT=Active Generation Table.

accesses to database tuples of different sizes). PC+offset indexing can distinguish patterns based on the spatial region offset, which is sufficient to capture the common cases. Our result contradicts a prior study of uniprocessor OLTP and web traces [17], which indicated that PC+address provides superior coverage.

Indices that correlate primarily based on program context (PC, PC+offset) are fundamentally more powerful than alternatives that include complete addresses (Address, PC+address) because they can predict accesses to data that have not been used previously—a crucial advantage for DSS. The scan and join operations that dominate DSS access many data only once. Address-based indices cannot predict previously-unvisited addresses and thus fail to predict many spatially-correlated accesses. Both the PC and PC+offset schemes can predict unvisited addresses, but, as with OLTP and web applications, the ability of PC+offset to distinguish among traversals allows it to achieve the highest coverage.

For scientific applications, we corroborate the conclusions of prior work [4] that indicate PC+offset indexing generally approaches the peak coverage achieved by the PC+address indexing scheme.

A second advantage of PC+offset indexing over alternatives that include complete addresses is that its storage requirements are proportional to code size rather than data set size. Figure 7 compares PC+offset and PC+address at practical PHT sizes. PC+offset attains peak coverage with 16k entries—roughly the same hardware cost as a 64kB L1 cache data array. For PC+address, in all workloads except OLTP, 16k entries is far too small to capture a meaningful fraction of program footprint and provide significant coverage. In OLTP, where most coverage arises from frequent accesses to relatively few structures, PC+address achieves 75% of peak coverage with a 16k-entry PHT.

4.3. Decoupled Training

The training structure (e.g., the AGT) is a key component in any spatial-correlation predictor because structural limitations can prematurely terminate spatial region generations—particularly when accesses to different regions are interleaved—and thus reduce predictor coverage and/or fragment prediction entries, consequently polluting the PHT.

Past predictors [4,17] couple the predictor training structure to a sectored (i.e., sub-blocked) cache tag array. In a sectored cache, the valid bits in the tag array for each sector implicitly record a spatial pattern, thus requiring only minimal hardware changes to train a predictor (e.g., to track PC/address of the trigger access). However, sectored caches are less flexible than traditional caches and experience worse conflict behavior. To mitigate this disadvantage, the spatial footprint predictor [17] employed a decoupled sectored cache [22], whereas the spatial pattern predictor [4] provided a logical sectored-cache tag array alongside a traditional cache. The logical sectored-cache tag array calculates cache contents as if the cache was sectored, but does not affect actual cache replacements. Nevertheless, both these organizations incur more address conflicts than a traditional cache, and thus cannot accurately track available spatial correlation.

We compare the AGT to both of these organizations in Figure 8. We measure coverage by comparing the miss rate of each implementation against a baseline traditional cache. We model an infinite PHT to factor out predictor storage limitations from this analysis.

In commercial workloads, the additional constraints that the decoupled sectored cache (DS) places on cache contents lead to considerably more misses than in both other approaches. The conflict effects are magnified in applications where few generations are dense (OLTP and web; see Figure 5). In the scientific applications, blocks in the same

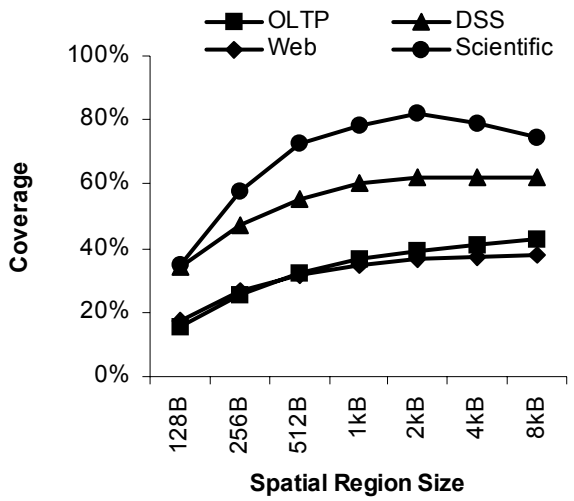

FIGURE 10. Spatial region size. SMS with PC+offset indexing and AGT training. PHT size is unbounded.

sector tend to be replaced together, and thus the decoupled and logical sectored tags behave identically.

Although the logical sectored scheme achieves similar coverage to AGT, when accesses across regions are interleaved, logical tag conflicts still fragment generations and create more history patterns. Figure 9 compares the two approaches in terms of PHT storage requirements. In general, for any coverage that the logical sectored design can achieve, it requires twice the PHT storage of AGT. The gap is largest for OLTP, which exhibits the most interleaving.

4.4. Spatial Region Size

Our oracle study (Figure 4) indicates that there is increasing opportunity as spatial region size increases to 8kB. For SMS to exploit this opportunity, accesses in a region must be repetitive and correlate to the trigger access. However, larger regions are more likely to span unrelated data structures, and therefore some accesses may not be repetitive with respect to the trigger access. We explore SMS's sensitivity to region size in Figure 10, using AGT training and unlimited PHT storage. We vary region size from 128B (two blocks) to the OS page size of 8kB (128 blocks).

In the database workloads, spatial regions do not span data structures, because the structures are aligned to database pages. Thus, in OLTP, coverage increases with region size. In DSS, most patterns are dense, so the benefit to merging adjacent spatial regions (i.e., eliminating the trigger misses of additional regions) is negligible.

In scientific applications, at region sizes above 2kB, we observe the negative effect of spanning data structures. Using PC+address (rather than PC+offset) indexing can mitigate this effect by learning specific patterns for each boundary between data structures, at the cost of drastically increased PHT storage requirements.

Choosing a spatial region size involves a tradeoff between coverage and storage requirements. Storage is dominated by PHT size, which scales linearly with the size of spatial regions. All applications except OLTP exhibit peak coverage with 2kB regions. The 2% coverage increase for OLTP when increasing region size to 4kB does not justify the doubled PHT size. Unless otherwise specified, we use 2kB spatial regions for the results in this paper.

4.5. Active Generation Table

The AGT is responsible for recording all blocks accessed during a spatial region generation. If the AGT is too small, generations will be terminated prematurely by replacement, leading to reduced pattern density and increased PHT storage requirements. Fortunately, SMS is able to attain the same coverage with a practical AGT as with an infinite AGT—across all applications, a 32-entry filter table and a 64-entry accumulation table are sufficient. OLTP-Oracle places the largest demand on the accumulation table; it is the only application to require more than 32 accumulation table entries.

4.6. Comparison to State-of-the-Art Prefetchers

Although many prefetching and streaming techniques have been proposed, they do not target general memory access patterns for commercial workloads. We compare SMS against the Global History Buffer (GHB) [20], whose PC/DC (program counter / delta correlation) variant was shown to be the most effective prefetching technique for desktop/engineering applications [12]. Like SMS, GHB-PC/DC exploits spatial relationships between addresses. However, GHB seeks to predict the sequence of offsets across consecutive memory accesses by the same instruction.

We consider GHB with two history buffer sizes: 256 entries (sufficient for SPEC applications [12,20]) and 16k entries (to roughly match the capacity of the SMS PHT). The GHB lookup mechanism requires multiple buffer accesses upon each prefetch; as such, GHB was proposed for and is only applicable to L2 caches. Thus, we compare the off-chip miss coverage of GHB and SMS in Figure 11.

SMS outperforms GHB in OLTP and web applications. These applications interleave accesses to multiple spatial regions. SMS captures these accesses because the trigger access in each region independently predicts a pattern for the region. With GHB, however, when multiple access sequences are interleaved, the offset sequences are disrupted. Therefore, GHB can only predict interleaved sequences if the interleaving itself is repetitive.

The DSS workloads access fewer regions in parallel; hence, interleaving is less frequent. Furthermore, DSS access sequences are highly structured—scans and joins, instead of the searches common in OLTP—which allow GHB to nearly match SMS's coverage. Likewise, in the scientific applications, both predictors capture the repetitive access sequences.

4.7. Performance Results

We evaluate the performance impact of SMS on scientific and commercial applications with respect to a baseline system without SMS. Figure 12 show the performance improvement for each application with 95% confidence inter-

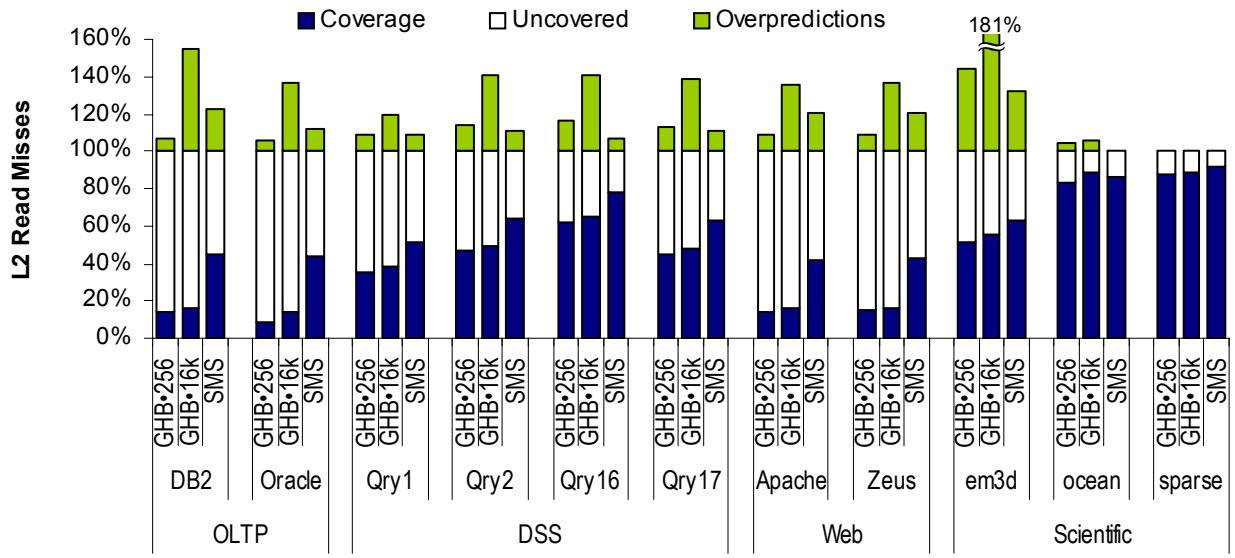

FIGURE 11. Practical SMS configuration and comparison to the Global History Buffer. GHB•256 and GHB•16k refer to PC/DC GHB with 256- and 16k-entry history buffers, respectively. SMS uses AGT training (with a 32-entry filter table and 64-entry accumulation table), 2kB spatial regions, and a 16k-entry, 16-way set-associative PHT.

vals given by our sampling methodology. Figure 13 presents execution time breakdowns for both systems. The two bars for each application are normalized to represent the same amount of completed work. Thus, the relative height of the bars indicates speedup, while the size of each component indicates the time per unit of forward progress spent on the corresponding activity. User busy and system busy time indicate cycles in which at least one instruction is committed. The three depicted stall categories represent stalls waiting for load data from off-chip, from an on-chip cache (e.g., L2), and store-buffer-full stalls. Finally, the remaining category accumulates all other stall sources (e.g., branch mispredictions, instruction cache misses, etc.).

In all workloads, SMS improves performance by reducing off-chip read stalls. We observe performance improvements of over 20% in the web, scientific, and DSS (except Qry1) workloads.

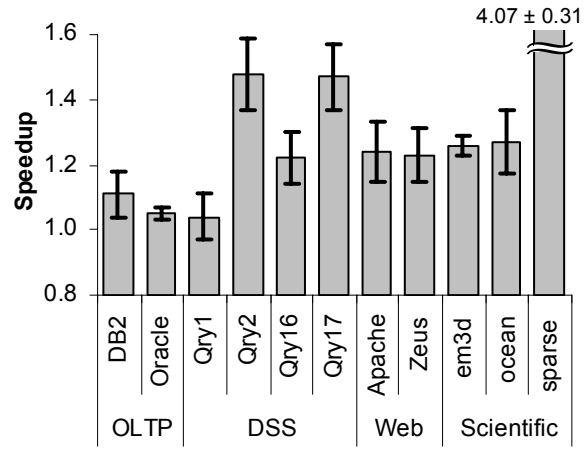

FIGURE 12. Speedup with 95% confidence intervals. Geometric mean speedup is 1.37.

In OLTP workloads, many of the misses that SMS predicts coincide with misses that the out-of-order core is able to overlap. Even though overall MLP is low [6], misses that the core can issue in parallel also tend to be spatially correlated (e.g., accesses to multiple fields in a structure). Therefore, the impact of correctly predicting these misses is reduced and speedup is lower than our coverage results suggest.

In the scan-dominated Qry1, SMS has no statistically significant effect, despite high prediction coverage. In this query, a large amount of data is copied to a temporary database table, which rapidly fills the store buffer with requests that miss in the cache hierarchy. Hence, store-buffer-full stalls limit performance improvement. In this situation, load streaming by SMS is counterproductive, because the read-only blocks fetched by SMS must all be upgraded (i.e., write permission obtained via the coherence protocol), delaying the critical path of draining the store buffer.

One surprising effect we see is an apparent reduction in system busy time with SMS for web and DSS workloads. However, the absolute fraction of system busy time (i.e., not normalized to forward progress) is identical between the base and SMS systems. We infer that the OS activity during these system-busy intervals is not on behalf of the application, but instead OS work that is proportional to time—for example, servicing traffic from a saturated I/O subsystem.

In em3d, MLP is high (>4.5) and SMS coverage (63%) is insufficient to predict all misses in a burst. Therefore much of the latency for each burst remains exposed. In sparse, because prediction coverage is high (92%), SMS eliminates nearly all off-chip miss time, improving performance by 307%.

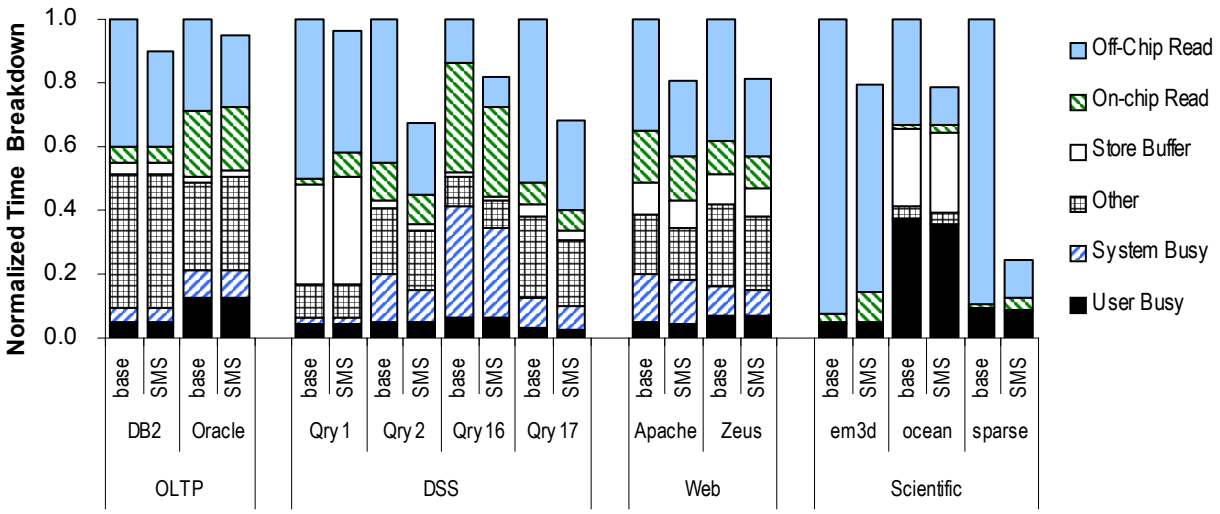

FIGURE 13. Time breakdown comparison. The base and SMS bars for each application are normalized to represent the same amount of completed work. Their relative height indicates speedup.

5. Related work

Several prior proposals use offline analysis to exploit variations in spatial locality. Vleet and co-authors [28] propose offline profiling to select fetch size upon a miss. Guided region prefetching [29] uses compiler hints to direct both spatial and non-spatial (e.g., pointer-chasing) prefetches. However, the complex access patterns and rapid changes in data set common in commercial applications present a challenge for static and profile-based approaches. Moreover, these approaches require application changes or recompilation, whereas SMS is software transparent and adapts at runtime to changing application behavior.

A variety of hardware approaches exploit variations in spatial locality at runtime, including the dual data cache [11], the spatial locality detection table [16], and caches that dynamically adjust block size [8,27]. All of these techniques exploit spatial locality variation at coarse granularity, thus sacrificing either bandwidth efficiency or prefetch opportunity. SMS does not modify the fetch/block size, and instead predicts, at fine granularity, precisely which blocks to fetch from a larger region.

A large class of prediction approaches exploit temporal rather than spatial correlation among addresses (e.g., [25, 30]). These approaches eliminate recurring pairs or sequences of consecutive misses. SMS identifies multiple simultaneously-active spatially-correlated regions whose accesses are interleaved. Such spatially-correlated accesses appear uncorrelated to temporal predictors because of the interleaving. Furthermore, the storage requirements of temporal predictors are proportional to data set size, and are therefore larger than for SMS.

6. Conclusion

In this paper, we showed that memory accesses in commercial workloads are spatially correlated over large memory regions (e.g., several kB) and that this correlation is repetitive and predictable. We demonstrated that code-based correlation is fundamentally superior to address-based correlation because it can predict previously-unvisited addresses. We proposed Spatial Memory Streaming, a practical on-chip hardware technique that identifies code-correlated spatial patterns and streams predicted blocks to the primary cache ahead of demand misses. Using cycle-accurate full-system multiprocessor simulation running commercial and scientific applications, we demonstrated that SMS can on average predict 58% of L1 and 65% of off-chip misses, for an average speedup of 1.37 and at best 4.07.

Acknowledgements

The authors would like to thank the SimFlex team, and Michael Ferdman, Jared Smolens, and the anonymous reviewers for their feedback on drafts of this paper. This work was partially supported by grants and equipment from Intel, two Sloan research fellowships, an NSERC Discovery Grant, an IBM faculty partnership award, and NSF grants CCR-0205544, CCR-0509356, and IIS-0133686.

References

[1] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29(12):66–76, Dec. 1996.

[2] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a modern processor: Where does time go? In *The VLDB Journal*, Sep. 1999.

[3] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the 25th International Symposium on Computer Architecture*, June 1998.

[4] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *Proceedings of the Tenth Symposium on High-Performance Computer Architecture*, Feb. 2004.

[5] S. Chen, A. Ailamaki, P. B. Gibbons, and T. C. Mowry. Improving hash join performance through prefetching. In *Proceedings of the 20th International Conference on Data Engineering*, Apr. 2004.

[6] Y. Chou, B. Fahs, and S. Abraham. Microarchitecture optimizations for exploiting memory-level parallelism. In *Proceedings of the 31st International Symposium on Computer Architecture*, June 2004.

[7] Z. Cvetanovic. Performance analysis of the Alpha 21364-based HP GS1280 multiprocessor. In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003.

[8] C. Dubnicki and T. J. LeBlanc. Adjustable block size coherence caches. In *Proceedings of the 19th International Symposium on Computer Architecture*, June 1992.

[9] K. Gharachorloo, A. Gupta, and J. Hennessy. Two techniques to enhance the performance of memory consistency models. In *Proceedings of the 1991 International Conference on Parallel Processing*, Aug. 1991.

[10] C. Gniady, B. Falsafi, and T. N. Vijaykumar. Is SC + ILP = RC? In *Proceedings of the 26th International Symposium on Computer Architecture*, May 1999.

[11] A. Gonzalez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *International Conference on Supercomputing*, July 1995.

[12] D. Gracia Perez, G. Mouchard, and O. Temam. MicroLib: A case for the quantitative comparison of micro-architecture mechanisms. In *Proceedings of the 37th International Symposium on Microarchitecture*, Dec. 2004.

[13] R. Hankins, T. Diep, M. Annavaram, B. Hirano, H. Eri, H. Nueckel, and J. P. Shen. Scaling and characterizing database workloads: Bridging the gap between research and practice. In *Proceedings of the 36th International Symposium on Microarchitecture*, Dec. 2003.

[14] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. SimFlex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, 31(4):31–35, Apr. 2004.

[15] J. Huh, J. Chang, D. Burger, and G. S. Sohi. Coherence decoupling: making use of incoherence. In *Proceedings of the Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[16] T. Johnson, M. Merten, and W.-M. Hwu. Run-time spatial locality detection and optimization. In *Proceedings of the 31st International Symposium on Microarchitecture*, Nov.1998.

[17] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *Proceedings of the 25th International Symposium on Computer Architecture*, June 1998.

[18] A.-C. Lai and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[19] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt. Runahead execution: an effective alternative to large instruction windows. *IEEE Micro*, 23(6):20–25, Nov./ Dec. 2003.

[20] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. In *Proceedings of the Tenth Symposium on High-Performance Computer Architecture*, Feb. 2004.

[21] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso. Performance of database workloads on shared-memory systems with out-of-order processors. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1998.

[22] A. Seznec. Decoupled sectored caches. In *IEEE Transactions on Computers*, 46(2):210–215, 1997.

[23] M. Shao, A. Ailamaki, and B. Falsafi. DBmbench: Fast and accurate database workload representation on modern microarchitecture. In *Proceedings of the 15th IBM Center for Advanced Studies Conference*, Oct. 2005.

[24] T. Sherwood, S. Sair, and B. Calder. Predictor-directed stream buffers. In *Proceedings of the 33rd International Symposium on Microarchitecture,* Dec. 2000.

[25] Y. Solihin, J. Lee, and J. Torrellas. Using a user-level memory thread for correlation prefetching. In *Proceedings of the 29th International Symposium on Computer Architecture*, May 2002.

[26] P. Trancoso, J.-L. Larriba-Pey, Z. Zhang, and J. Torellas. The memory performance of DSS commercial workloads in shared-memory multiprocessors. In *Proceedings of the Third Symposium on High-Performance Computer Architecture*, Feb. 1997.

[27] A. V. Veidenbaum, W. Tang, R. Gupta, A. Nicolau, and X. Ji. Adapting cache line size to application behavior. In *International Conference on Supercomputing*, July 1999.

[28] P. V. Vleet, E. Anderson, L. Brown, J.-L. Bear, and A. Karlin. Pursuing the performance potential of dynamic cache line sizes. In *International Conference on Computer Design*, Oct. 1999.

[29] Z. Wang, D. Burger, K. S. McKinley, S. K. Reinhardt, and C. C. Weems. Guided region prefetching: a cooperative hardware/software approach. In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003.

[30] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. Temporal streaming of shared memory. In *Proceedings of the 32nd International Symposium on Computer Architecture*, June 2005.

[31] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe. Simulation sampling with live-points. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, June 2006.

[32] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation through rigorous statistical sampling. In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003.