Journal of Instruction-Level Parallelism 13 (2011) 1-26 Submitted 3/10; published 1/11

Spatial Memory Streaming

Stephen Somogyi SSOMOGYI@ALUMNI.CMU.EDU
Advanced Micro Devices, Inc.
1 AMD Place
Sunnyvale, CA 94088 USA

Thomas Wenisch TWENISCH@UMICH.EDU
Department of Electrical Engineering and Computer Science
University of Michigan
2260 Hayward Street
Ann Arbor, MI 48109 USA

Michael Ferdman MFERDMAN@ECE.CMU.EDU
Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213 USA

Babak Falsafi BABAK.FALSAFI@EPFL.CH
Computer & Communication Sciences
Ecole Polytechnique Federale de Lausanne
INJ 233, Station 14
CH-1015 Lausanne, Switzerland

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 this 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. We show that memory accesses 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 describe *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.

1. Introduction

As processor clock frequencies have increased drastically during the past decades, memory access latency has improved only marginally, resulting in a performance gap known as the memory wall. Although power constraints have recently limited further growth in the memory wall, technology trends do not forecast the gap to shrink in the future. Instead, memory latency will remain on the order of hundreds of processor clock cycles, causing significant performance loss as processors stall, waiting for memory instead of making forward progress.

Many techniques have been proposed to reduce the performance impact of memory latency. Traditionally, architects used caches to keep recently used data on chip and avoid costly memory accesses. However, caches show diminishing improvements to miss rates as their sizes scale beyond multiple megabytes [14], and with chip multiprocessors (CMPs), the silicon area for larger caches might be better utilized for additional cores. Moreover, caches cannot reduce off-chip coherence misses in multiprocessor systems, nor can they reduce compulsory misses in applications that scan large datasets.

Prefetching is a promising approach for reducing memory stalls. A prefetcher predicts what data the processor will request in the near future and fetches this data into the caches, thus effectively hiding the memory latency. Stride prefetchers [2,21] have been implemented in commercial processors (e.g., [16,29]), predicting regularly strided accesses to accelerate applications dominated by these simple access patterns. However, in many applications such accesses only account for a fraction of total memory behavior, leading researchers to investigate more sophisticated prefetching techniques.

Commercial workloads exhibit more complex and varied memory behaviors [3,25] than desktop, engineering, or scientific workloads, because of their data dependent behavior, large instruction footprint, and interactions with the operating system and I/O devices. Commercial workloads are multi-threaded, with sophisticated fine-grain locking protocols that maximize concurrency, leading to non-determinism and contention in the memory system. Therefore, memory access patterns are complex and irregular, resulting in low prefetch coverage (i.e., few misses predicted correctly) and accuracy (i.e., many incorrect predictions for each correct one).

Nevertheless, commercial workloads compute over massive datasets and have been highly tuned for performance. To achieve this goal, their data is organized in a very structured (if complex) manner, leading to data structures with repetitive layouts and access patterns. As the applications traverse their datasets, recurring patterns emerge in the relative offsets of accessed data. 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), the term *spatial correlation* rather than spatial locality is used to describe the relationship among accesses.

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 [7,23]. While spatial correlation prefetching is effective for desktop/engineering applications [7], the only practical implementation prior to our study achieved less than 20% miss-rate reduction on server workloads [23]. In this work, using a combination of trace-based and cycle-accurate full-system simulation of multiprocessor commercial and scientific applications, we demonstrate:

- **Effective spatial correlation and prediction.** Contrary to previous findings [23], 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 Spatial Memory Streaming (SMS) exploits to predict patterns even for previously-unvisited addresses.
- **Accurate tracking of spatial correlation.** We show that the cache-coupled structures used in previous work ([7,23]) are suboptimal for observing spatial correlation. Accesses to multiple independent patterns are frequently interleaved, inducing 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%.

- **SMS prediction coverage and performance enhancement.** For commercial workloads, we show that SMS predicts on average 55% and at best 78% of cache read misses, providing a mean speedup of 22% and at best 48% over a system without SMS.
- **Spatial correlation of write misses.** We show that off-chip write misses are spatially correlated and predictable with SMS. For commercial workloads, SMS predicts on average 62% of write misses with only 9% overpredictions.
- **Stable spatial repetition through hysteresis.** Over different occurrences of a spatial pattern, some blocks toggle between being accessed and not. By using a 2-bit saturating counter per block for spatial history, SMS overpredicts on average only 8% of cache read misses, compared with 15% when using simple bit vectors.
- **Rotation indexing to reduce redundant storage of spatial patterns.** Because SMS encodes patterns as bit vectors, where each bit maps trivially within a spatial region, it must record different versions of a pattern for each initial offset. With rotation indexing, SMS observes and predicts patterns relative to their initial access, reducing history storage by half for commercial workloads.

2. Overview

Spatial Memory Streaming (SMS) improves the performance of scientific and commercial server applications by exploiting spatial relationships among data beyond a single cache block. Architects have long relied on spatial relationships among memory accesses to improve computer system performance. *Spatial locality* states that items whose addresses are near one another tend to be referenced close together in time [17]. System designers take advantage of spatial locality by moving data at a block granularity—typically 32, 64, or 128 bytes—instead of at word granularity (four or eight bytes). This approach mitigates the overhead of each transfer (e.g., bus acquisition), leading to efficient memory hierarchies that are capable of nearly saturating their peak theoretical bandwidth.

The principle of spatial locality holds true especially for smaller block sizes, with caches in modern systems drastically reducing off-chip memory accesses. However, spatial locality exhibits characteristics that limit its potential effectiveness. Many workloads are not amenable to capturing additional spatial locality by increasing the cache block size, for example, to the order of an operating system page. Commercial workloads in particular exhibit low memory access density: only a small fraction of the data in a multi-kilobyte region is accessed over a reasonable window in time. Using large cache blocks to exploit these spatially related accesses is impractical, leading to bandwidth inefficiency and reducing effective cache capacity.

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.

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/

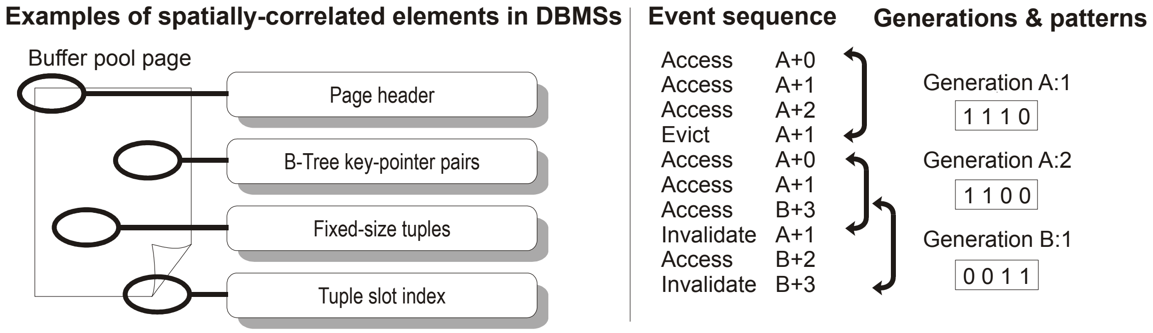

Figure 1: Examples of spatial correlation and spatial region generations. The left figure shows example sources of spatial correlation in databases. The right figure illustrates an event sequence and the corresponding spatial region generations and patterns.

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 run-time 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.

3. Spatial Patterns and Generations

We formalize our notion of spatial correlation similar to prior studies of spatial footprints [7,23]. 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 [23]. 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.

3.1. 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 [7,23]. 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.

PC+address indexing can be approximated by combining the PC with a *spatial region offset* [7,23]. The spatial region offset of a data address is the distance 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. Additionally, 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 [28], contain long repetitive access patterns that visit data only once. In these applications, PC+offset indexing substantially outperforms address-indexed schemes.

4. Hardware Design

SMS comprises two hardware structures. The *active generation table* records spatial patterns as the processor accesses spatial regions. 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.

4.1. Observing Spatial Patterns

SMS 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.

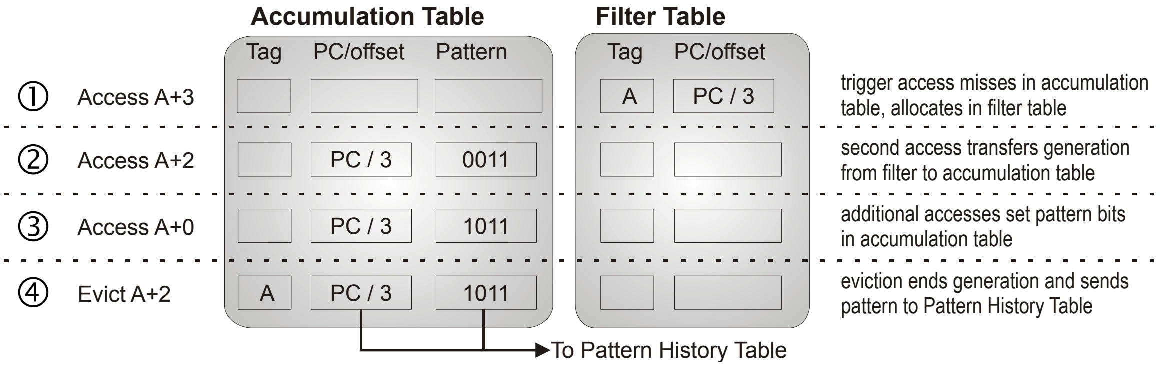
Accumulation Table
Filter Table
Tag
PC/offset
Pattern
Tag
PC/offset
① Access A+3
A
PC / 3
trigger access misses in accumulation table, allocates in filter table
② Access A+2
PC / 3
0011
second access transfers generation from filter to accumulation table
③ Access A+0
PC / 3
1011
additional accesses set pattern bits in accumulation table
④ Evict A+2
A
PC / 3
1011
eviction ends generation and sends pattern to Pattern History Table
To Pattern History Table

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.

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.

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 blocks that 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 with the PC and spatial region offset of the trigger access. 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

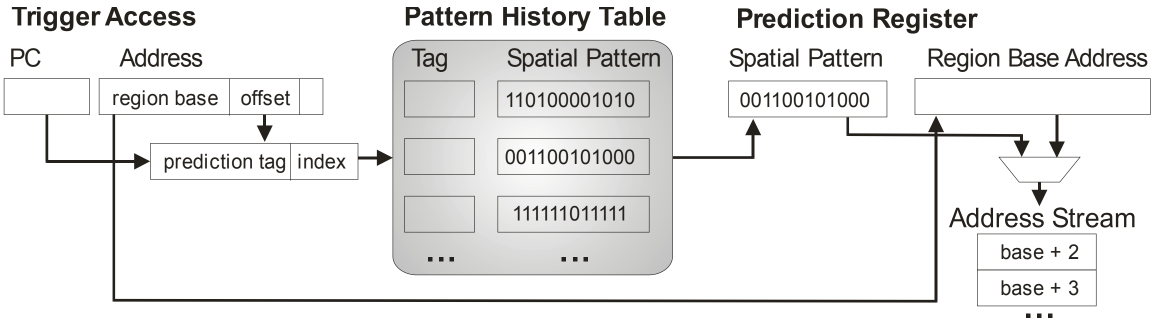

Figure 3: Pattern History Table and prediction process. Upon a trigger access that matches in the PHT, the region base address and spatial pattern are transferred to a prediction register, beginning the streaming process.

transferred from the accumulation table to the pattern history table). In Section 5.5, we observe that small (e.g., 32- or 64-entry) accumulation and filter tables make this occurrence rare.

4.2. 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. 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.

4.3. Predicting Reads and Writes

The primary target for SMS is read stalls because modern out-of-order processors are able to hide and tolerate latency of write operations. Nevertheless, the spatial patterns used by SMS consist of both reads and writes—a consequence of the fact that identifying spatial accesses as reads or writes requires additional bookkeeping.

Over the course of a spatial generation, each block that comprises the resulting pattern undergoes a (possibly unique) progression of reads and writes. Some blocks will only be read, others only written, and still others encounter both reads and writes. With respect to prediction, we are interested in the type of the initial access to a block. The type of any subsequent access is less important because the block has already been fetched and resides in the processor's primary data cache. Thus, the naive approach whereby the predictor sets a bit in the pattern on reads and on writes will misclassify read-write blocks. For example, if we target reads, and the first access to a block is a write, the corresponding bit should not be set. However, any subsequent read will cause the bit to be set.

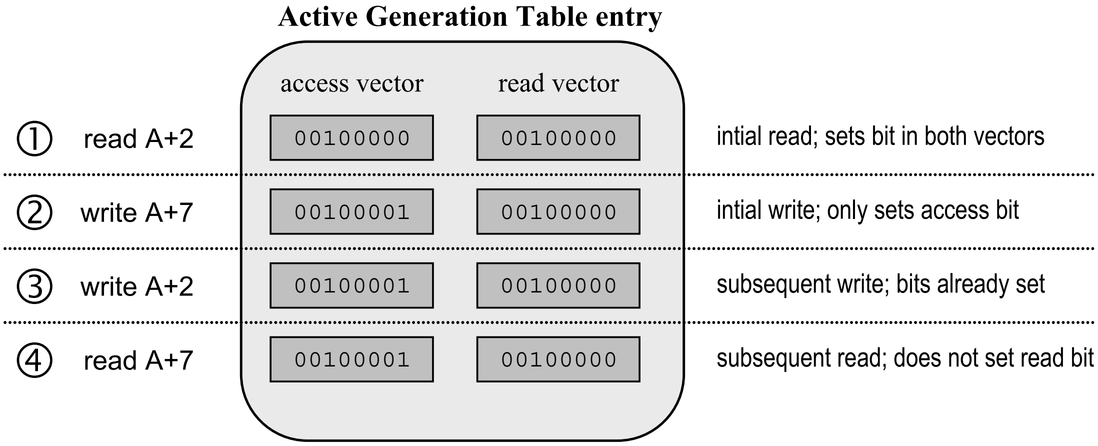
Active Generation Table entry
access vector
read vector
① read A+2
00100000
00100000
intial read; sets bit in both vectors
② write A+7
00100001
00100000
intial write; only sets access bit
③ write A+2
00100001
00100000
subsequent write; bits already set
④ read A+7
00100001
00100000
subsequent read; does not set read bit

Figure 4: Capturing spatial reads and writes. The access and read vectors illustrate their maintenance over a short sequence of reads and writes.

To reduce blocks extraneously recorded as part of spatial patterns, SMS must track the read/write nature of each spatial access. As shown in Figure 4, this operation requires an additional bit vector in the active generation table. To isolate reads or writes, on an access, SMS must first check the *access* spatial pattern: if the block has already been accessed this generation, no action is necessary; if this is the initial access for the block, then its bit in the *read* vector is set accordingly. At the end of a generation, the access vector represents all accesses, the read vector represents spatial reads, and the write vector can be constructed through the difference of the access and read vectors.

4.4. Expanding Spatial History

Spatial correlation—while effective—is imperfect: access patterns that change over time (e.g., because of varying control flow) can lead to overpredictions, which cause bandwidth overhead and pollution. In the basic SMS design, spatial history entries (i.e., stored in the PHT) correspond directly to observed spatial patterns. As subsequent patterns occur with the same prediction index, the old pattern is replaced with the newer one. To reduce overheads due to overprediction, we explore more sophisticated variations of spatial history. Much like branch predictors can improve their accuracy by incorporating history of more branch outcomes [37], we improve the accuracy of spatial prediction by incorporating additional spatial history.

We consider three approaches that all share the same storage cost. Under the first two approaches, we modify the PHT to maintain the two most recently observed patterns for each prediction index, instead of the single most recent. SMS performs one of two set operations on these patterns to determine a predicted pattern for prefetching: *union*, which seeks to maximize coverage at the expense of accuracy; and *intersection*, which minimizes overpredictions but is also likely to reduce coverage.

The third approach is more sophisticated. Instead of maintaining bit vectors where each bit corresponds to a block in a spatial region, PHT entries contain a vector of 2-bit saturating counters, each counter corresponding to a particular spatial region block. When SMS creates a new history entry (i.e., for a prediction index that does not exist in the PHT), per-block counters are initialized to *2* for blocks that were part of the observed pattern and *1* for blocks that were not. To update

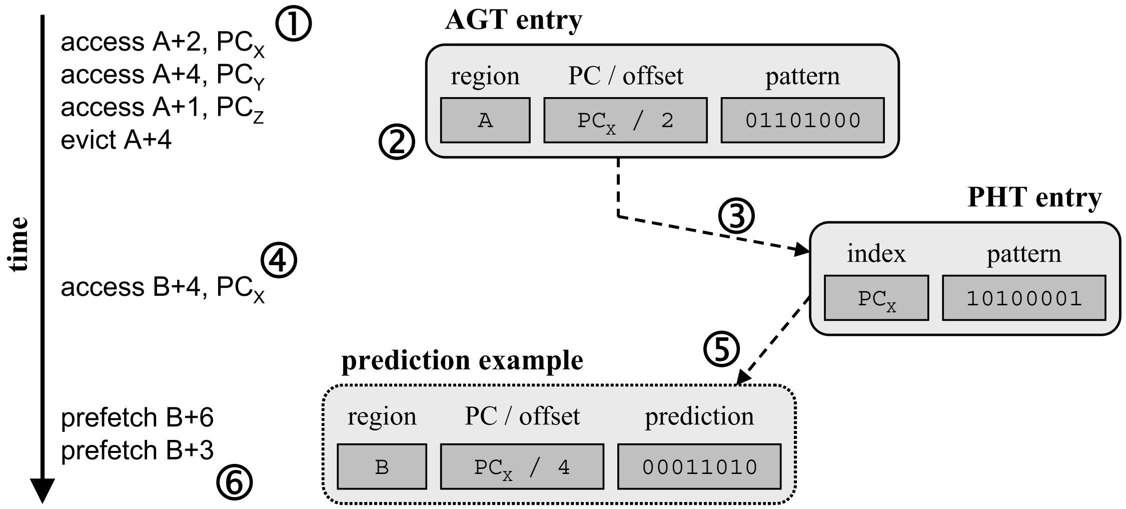

Figure 5: Rotated spatial patterns. The figure walks through an example: (1) operations belonging to one spatial generation, (2) corresponding AGT entry at generation end, (3) rotated pattern stored in PHT, (4) trigger access of new generation, (5) information required for prediction, and (6) prefetches issued.

spatial history (i.e., when the prediction index of a completed pattern matches an existing PHT entry), SMS increments counter values for blocks that were accessed, and decrements the other counters. During prediction, SMS prefetches all blocks for which the corresponding counter has value *2* or *3*.

4.5. Rotated Patterns

SMS employs PC+offset indexing to achieve good coverage with relatively low mispredictions (Section 5.2). SMS trains quickly and prefetches compulsory misses because its prediction index is PC-based. The offset portion of the index accounts for different alignments of spatial layouts relative to the fixed region boundaries imposed by SMS, ensuring that a prediction index maps accurately to a spatial pattern.

For every trigger PC, SMS learns a different pattern for each initial offset. However, these patterns are unlikely to be truly unique—rather, each consists of a similar pattern of offsets relative to its trigger. Conceptually, PHT entries are wasted for patterns that recur for different spatial offsets. Practically, these wasted entries are necessary because the bits in a spatial pattern directly correspond to blocks in a spatial region. A PC-only index (i.e., without the spatial offset) cannot distinguish between different instances of a pattern, and essentially prefetches random blocks if the offset of a generation being predicted does not match the offset of the generation when it was recorded.

We propose *rotation indexing* to eliminate redundant patterns caused by different offsets of the same trigger PC. The training process (observed patterns in the AGT) proceeds as before. However, prior to storing a pattern in the PHT, it is rotated so that the trigger offset is always the first bit. We illustrate the rotation process in Figure 5. SMS places the rotated pattern in the PHT according to a PC-only prediction index. When predicting a spatial generation, SMS looks up a pattern in the PHT according to the trigger PC and rotates it according to the offset of the trigger access in its spatial region.

Table 1: System and application parameters for SMS.

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 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, 64 clients, 2.0 GB buffer pool
Decision Support (TPC-H on DB2)	
Qry 1	Scan-dominated, 2.0 GB buffer pool
Qry 2	Join-dominated, 2.0 GB buffer pool
Qry 16	Join-dominated, 2.0 GB buffer pool
Qry 17	Balanced scan-join, 2.0 GB 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 relaxs., 20K res., err tol 1e-07
sparse	4096x4096 matrix

5. Evaluation

We evaluate SMS using a combination of trace-driven and cycle-accurate full-system simulation of a shared-memory multiprocessor using Flexus [15]. 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 [1,11]. We perform speculative load and store prefetching [10], and speculatively relax memory ordering constraints at memory barrier and atomic read-modify-write memory operations [11]. We list other relevant parameters of our system model in Table 1 (left).

We evaluate three classes of commercial workloads [3,14]. Online transaction processing (OLTP) executes many short database queries, each on a single processor. Its memory behavior is dominated by migratory sharing and chains of dependent misses. Decision support systems (DSS) consist of individual, long running queries that the database parallelizes across all processors. Its memory accesses are predominantly compulsory misses. Web servers handle many tiny requests, and much like OLTP, parallelism emerges across requests. The most common memory behavior in web serving is sharing of common OS structures. We also consider several multi-threaded scientific applications, which exhibit extremely repetitive memory behavior and mainly serve as a point of comparison against the commercial workloads.

We evaluate a distributed shared memory (DSM) machine consisting of 16 nodes with one processor core per node. Obviously, chip multiprocessors (CMPs) have recently become mainstream. Compared with DSMs, CMP-only systems exhibit different memory behaviors because coherence traffic is confined on chip and capacity pressure tends to increase. However, even with CMPs, servers in our target market segment will continue to use multiple chips, thus producing off-chip memory behavior similar to a pure DSM (i.e., with one core per chip). In this study, we focus on a pure DSM model because it directly exposes off-chip memory access patterns, enabling us to better understand spatial and temporal correlation than through a hybrid multi-chip, multicore architecture.

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 [28]: 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 5.10 use this trace-based methodology.

For cycle-accurate simulations, we use a sampling approach developed in accordance with SMARTS [36]. 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 [35]. 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 cycles) as our performance metric, which is proportional to overall system throughput [34].

5.1. 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 6. 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

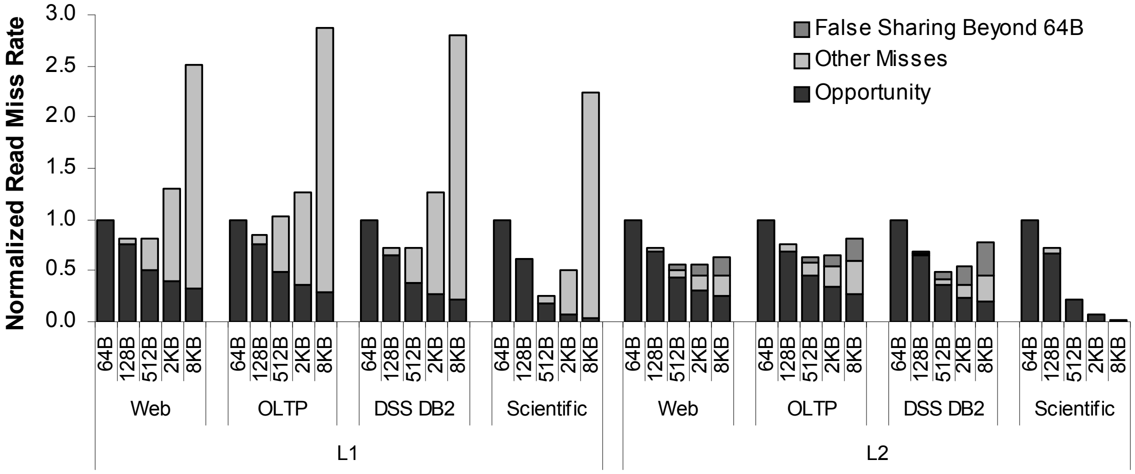

Figure 6: L1 and L2 (off-chip) miss rates versus block/region size. Opportunity represents an oracle spatial predictor that incurs one miss per spatial region generation.

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 L2 capacity reduces the prevalence of conflict effects as compared to the L1 cache. 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 et al. demonstrate that the latency penalty of false sharing can be eliminated through coherence decoupling—speculative use of incoherent data [18]. 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.

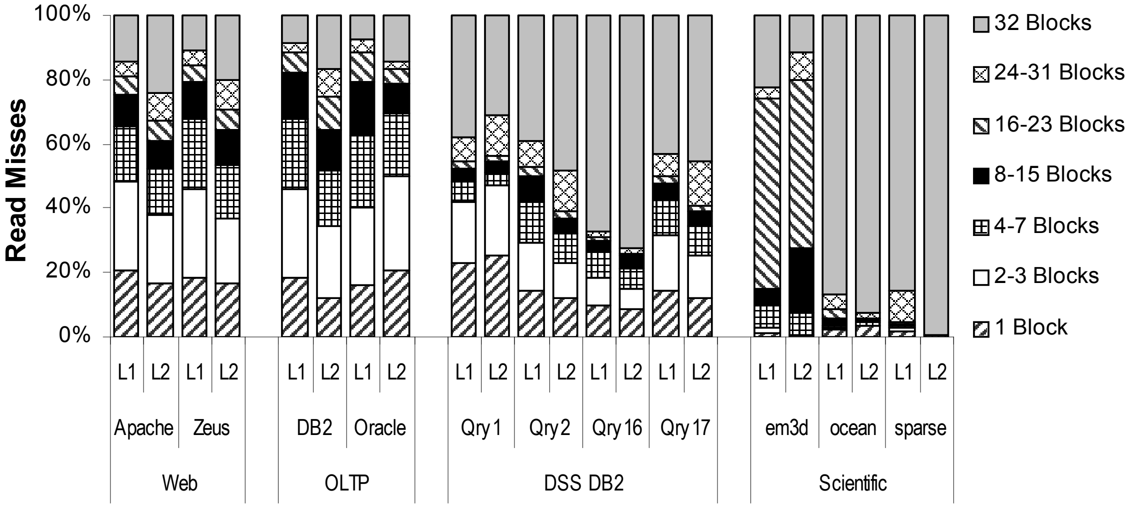

Figure 7: Memory access density. Each segment represents the percentage of L1 or L2 misses from generations of the indicated density (i.e., the number of blocks in the 2KB spatial region that incur misses during the generation).

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 7 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 5.4). 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 the L1 and L2 caches. Thus, no single block size can simultaneously exploit the available spatial correlation while using bandwidth and storage efficiently.

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 the L1 cache, despite its small capacity.

5.2. Indexing

Prior studies of spatial predictors [7,23] 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.

We compare the Address, PC+address, PC, and PC+offset indexing schemes in Figure 8, 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.

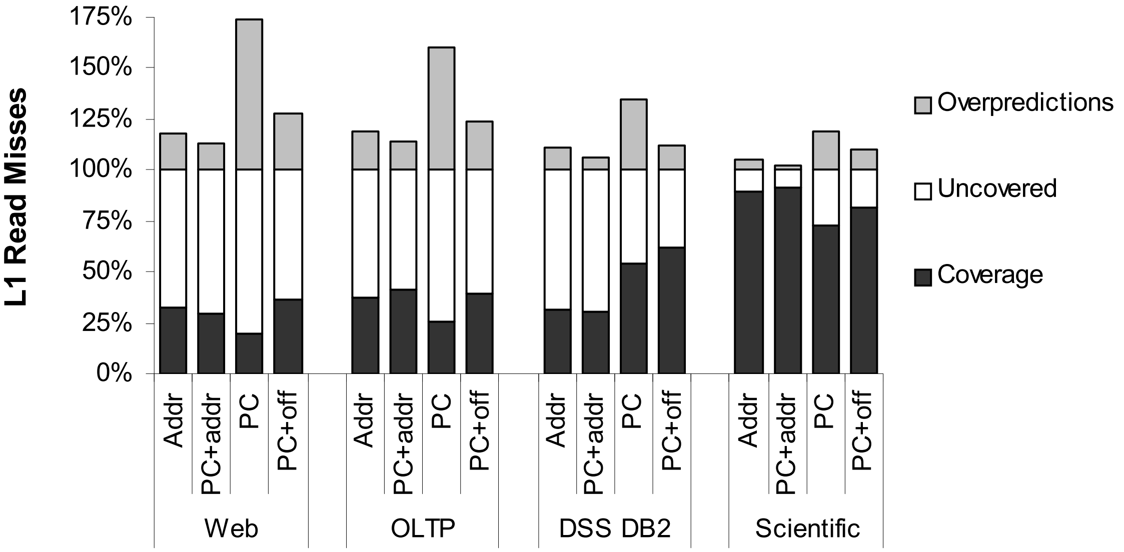

Figure 8: Index comparison. The index type is given below each bar. PHT size is unbounded.

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., 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 [23], 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 blocks 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 9 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

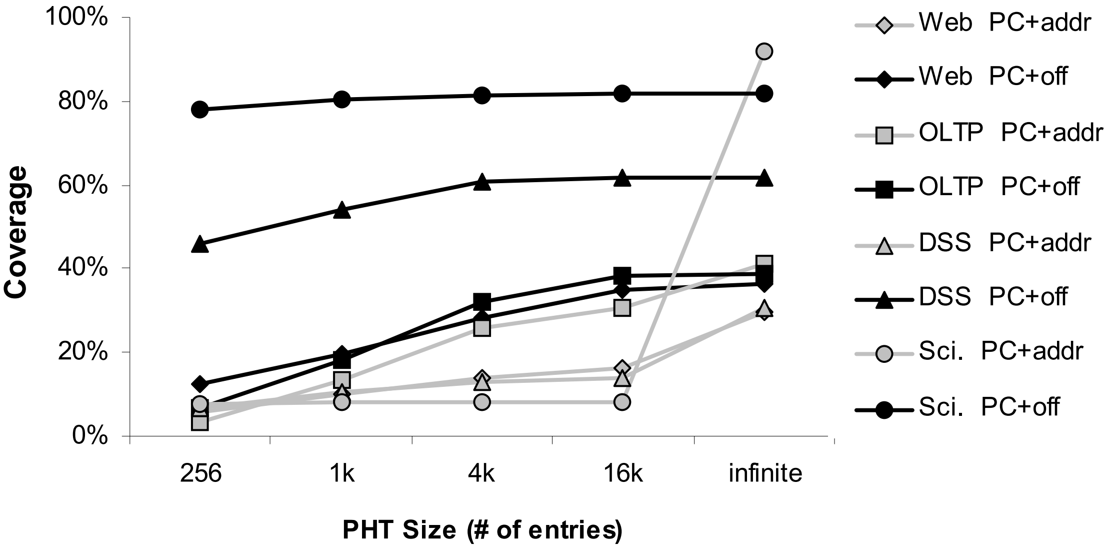

Figure 9: PHT storage sensitivity for PC+address and PC+offset indexing. The finite PHTs are 16-way set-associative.

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.

5.3. 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 [7,23] 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 [23] employs a decoupled sectored cache [27], whereas the spatial pattern predictor [7] provides 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 10. 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, as

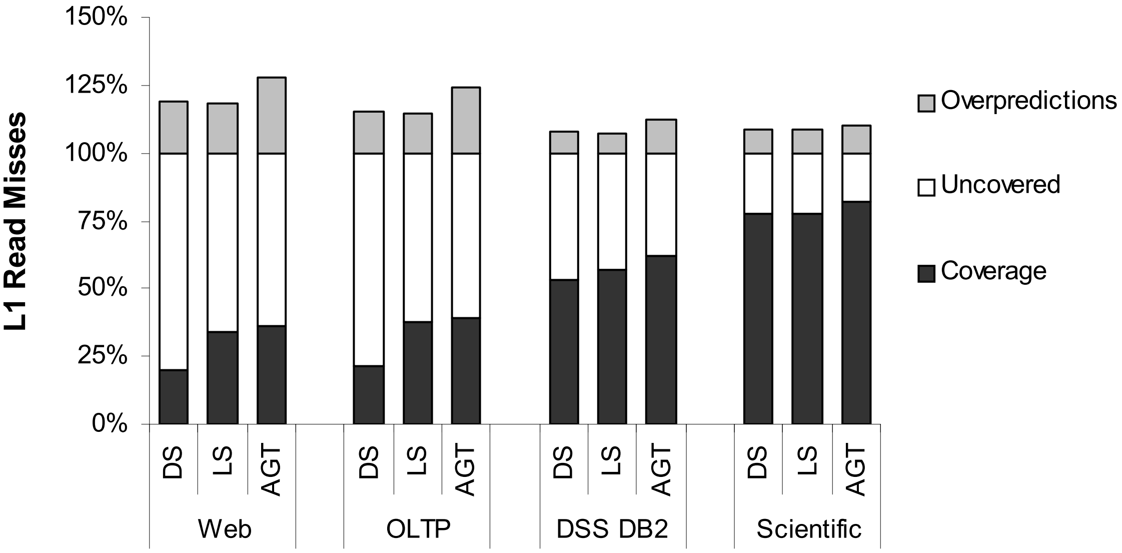

Figure 10: Comparison of training structures. DS=Decoupled Sectored. LS=Logical Sectored. AGT=Active Generation Table. PHT size is unbounded.

we demonstrated in Figure 7). In the scientific applications, blocks in the same 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 11 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.

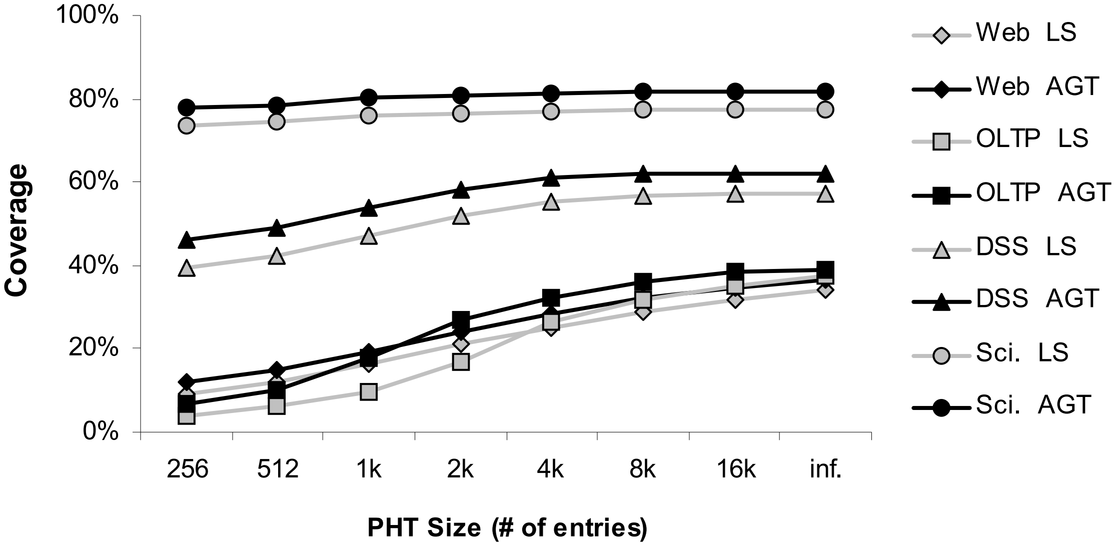

Figure 11: PHT storage sensitivity. LS=Logical Sectored. AGT=Active Generation Table.

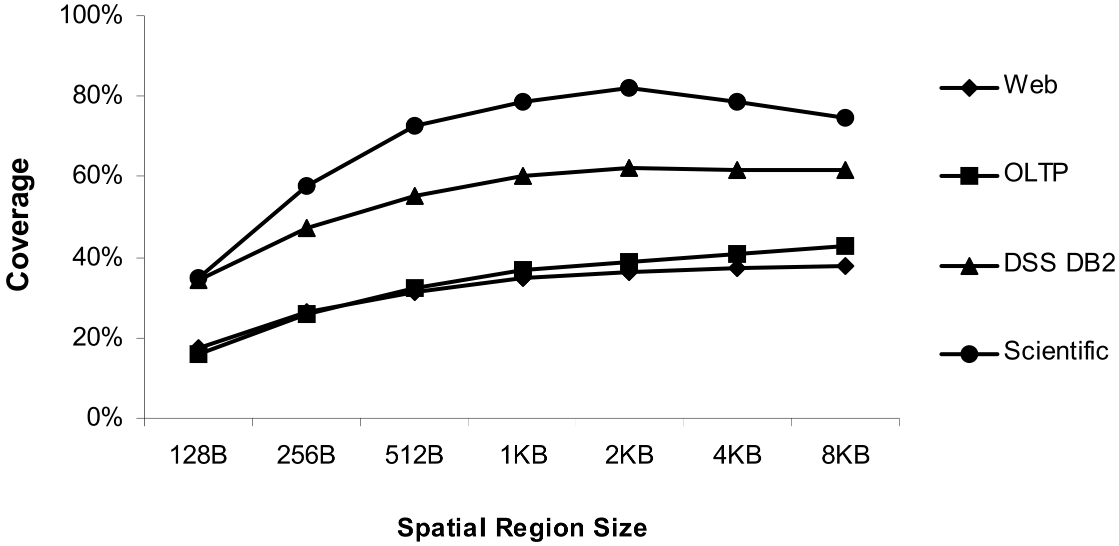

Figure 12: Spatial region size. SMS with PC+offset indexing and AGT training. PHT size is unbounded.

5.4. Spatial Region Size

Our oracle study (Figure 6) 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 sensitivity to region size in Figure 12, 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.

5.5. 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

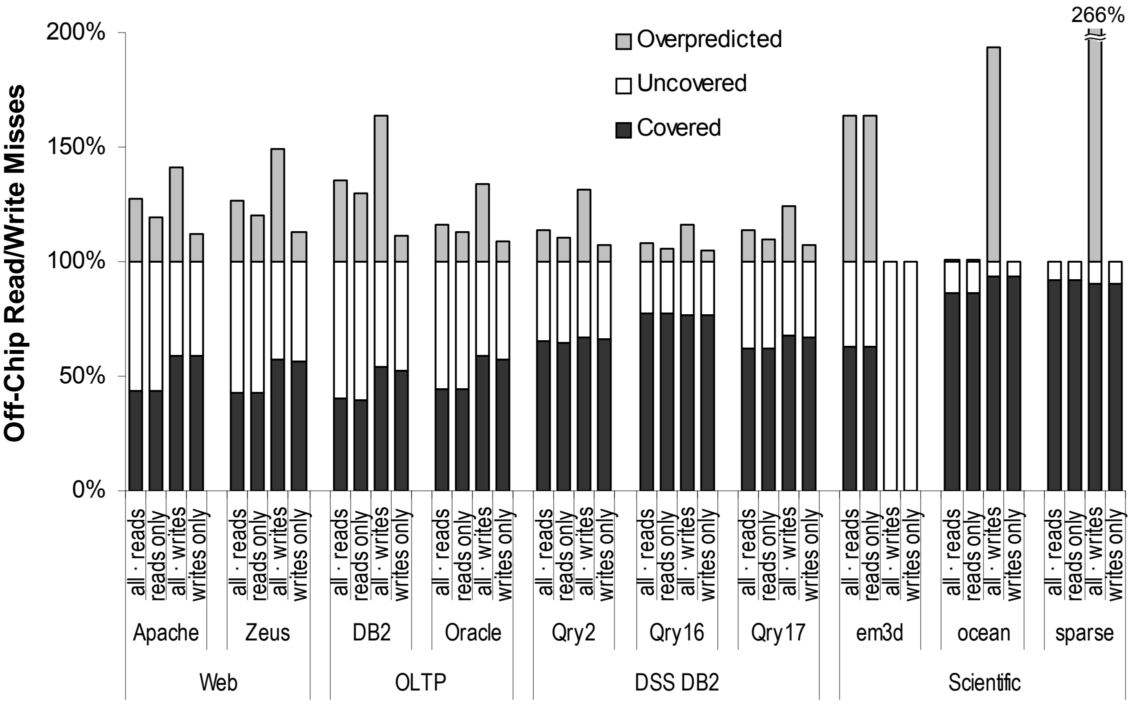

Figure 13: Targeting reads and writes. *All-reads* and *all-writes* record and predict patterns consisting of all accesses, but are evaluated with respect to read or write misses. *Reads-only* and *writes-only* represent optimizations that predict patterns of only read or write accesses.

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.

5.6. Differentiating Reads from Writes

Poor prediction accuracy is the downside of using general access patterns (as opposed to read patterns or write patterns) to predict only read misses or only writes misses. Targeting only reads, predictor accuracy can be improved. Although the primary goal of differentiating reads from writes is to reduce mispredictions for read misses, we explore the effectiveness of spatial correlation at predicting write misses. Predicting writes can be useful for in-order, sequentially consistent machines that cannot overlap write latency, or for predicting write sets and acquiring early write permission in advanced memory speculation techniques [6,33].

Results for read and write isolation are shown in Figure 13. The left-most bar for each application, *all-reads*, corresponds to the base SMS design. Observing read patterns and storing these in the PHT for later prediction (*reads only*), SMS achieves identical coverage as the base design. However, by eliminating the noise that writes cause for read prediction, we reduce overpredictions by 24% on average.

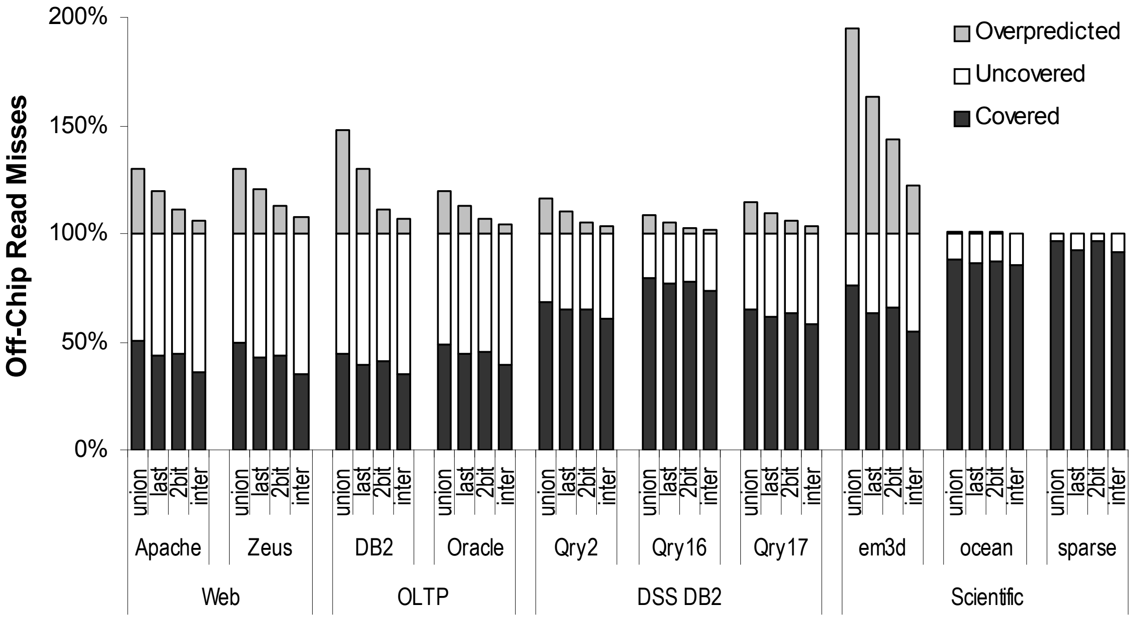

Figure 14: Improving spatial history. *last* uses the most recently observed pattern for spatial history, as in basic SMS. *union* and *inter* each maintain the two most recent patterns, and predict using union or intersection, respectively. *2bit* uses a 2-bit saturating counter per block, updated as patterns are observed.

The right-most two bars present the corresponding results for writes. With *all-writes*, SMS maintains and predicts patterns of all accesses, but we evaluate its predictions against off-chip write misses. As above, by isolating write vectors, SMS reduces overpredictions without impacting coverage. The difference in overpredictions is more pronounced with writes than with reads—writes are less frequent than reads, so the noise that reads add to write vectors is more significant than the noise caused by inserting writes into read vectors. Results for the scientific applications are noteworthy: in em3d, there are no write misses at all; in ocean and sparse, write misses are few and perfectly repetitive.

5.7. Expanding Spatial History

Figure 14 presents prediction results for SMS with expanded spatial history. We evaluate the effectiveness of SMS at predicting off-chip read misses and include the read-prediction optimization from Section 4.4. We arrange the results in order of decreasing overpredictions, from left to right: *union*, *last* pattern (i.e., the base SMS design), *2bit* saturating counters, and *inter*(section).

Spatial history has a direct influence on overpredictions. On average, each successively more conservative approach reduces overpredictions by 36%, 47%, and 43%, respectively. Union predicts everything; its overpredictions are highest. Conversely, intersection yields the lowest overpredictions. 2bit falls between simple last pattern and intersection because the hysteresis effect of the saturating counters prevents spurious blocks from being prefetched, but is not as restrictive as intersection.

The 2bit strategy captures stable spatial correlation, attaining essentially the same coverage as the single most recent pattern, but without mispredicting blocks that toggle into and out of patterns.

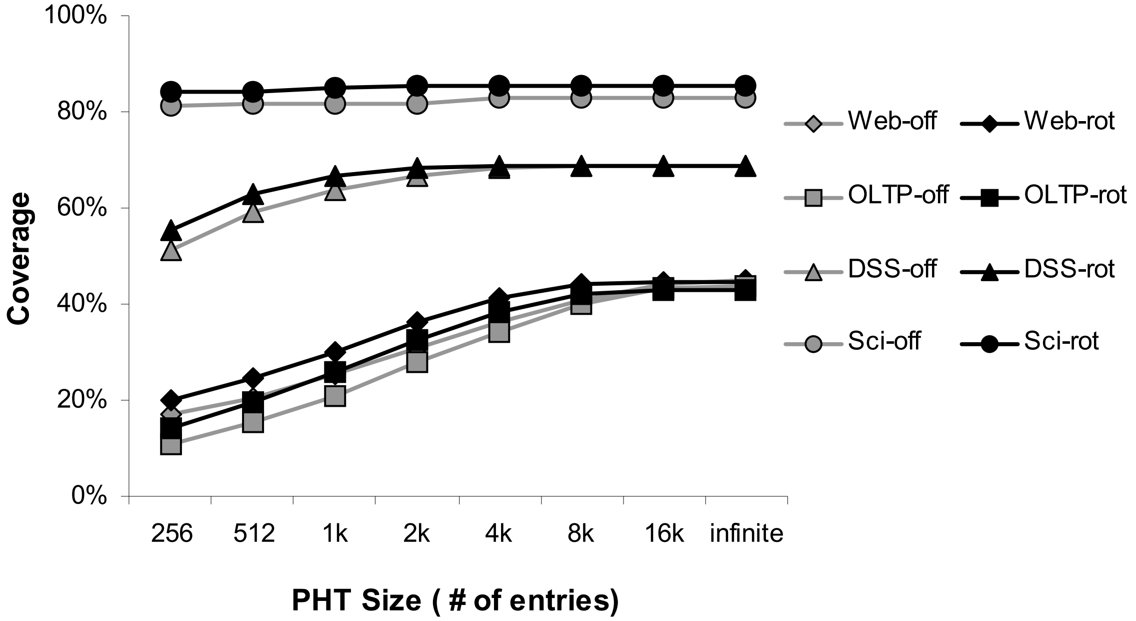

Figure 15: PHT storage with rotated patterns. *off* represents PC+offset indexing. *rot* represents PC indexing with rotation.

Furthermore, by reducing mispredictions compared with last pattern prediction, 2bit minimizes the effects of pollution, achieving a slight increase in coverage for some applications.

5.8. Rotated Patterns

The objective of rotation indexing is to reduce the size of the PHT without lowering coverage. We evaluate rotation indexing with the read-prediction optimization from Section 4.3 and 2-bit spatial history from Section 4.4. We show predictor coverage in Figure 15 as a function of PHT size, which ranges from 256 entries to infinite. We summarize the results by workload category, comparing PC+offset against PC-rotation indexing. For clarity, we omit overpredictions from the graph; across all applications, overpredictions scale with coverage and there is no difference between the two indexing methods.

With an infinite PHT, prediction coverage is similar for PC+offset and PC-rotation indexing. This result verifies that different patterns with the same trigger PC tend to be rotated versions of each other. If the patterns were unrelated, then coverage would drop (and overpredictions increase). em3d is solely responsible for the apparent increase in coverage of the scientific applications—its pseudo-random spatial patterns are more predictable with rotation than through static offsets.

Analyzing the entire range of PHT sizes, PC-rotation indexing requires roughly half the number of PHT entries as PC+offset indexing to achieve the same coverage. This trend holds for all the commercial workloads, although DSS exhibits no sensitivity above 2k entries. To attain peak coverage, SMS requires 16k PHT entries with PC+offset indexing, but just 8k entries using PC-only indexing with rotation.

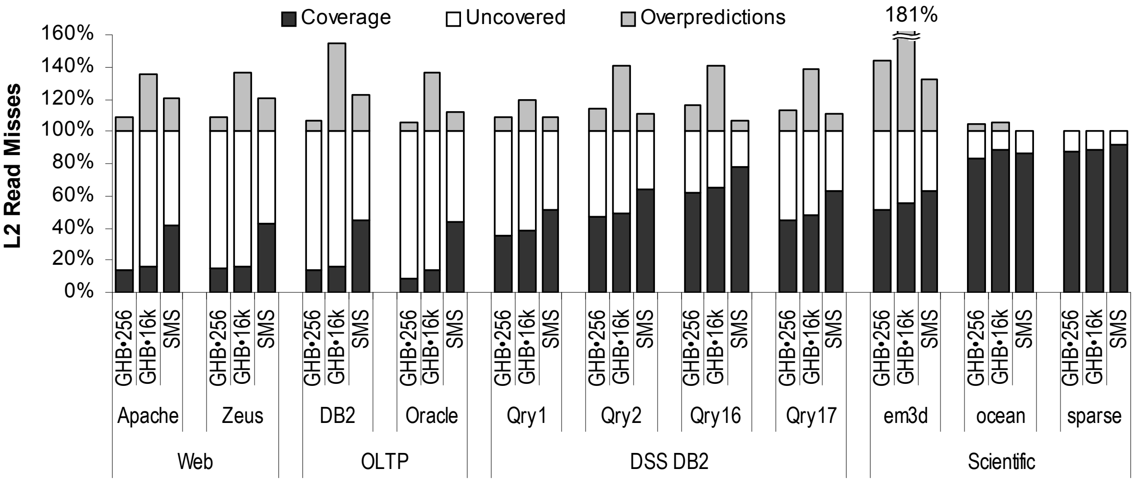

Figure 16: Practical SMS configuration and comparison to GHB. GHB•256 and GHB•16k refer to PC/DC Global History Buffer with 256- and 16k-entry history buffers, respectively. SMS uses AGT training (with a 64-entry accumulation and 32-entry filter table), 2KB spatial regions, and a 16k-entry, 16-way set-associative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

5.9. 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) [26], whose PC/DC (program counter / delta correlation) variant was shown to be the most effective prefetching technique for desktop/engineering applications [13]. 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 CPU applications [13,26]) 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 16.

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 coverage. As expected, in the scientific applications, both predictors capture the repetitive access sequences.

5.10. Performance Results

We evaluate the performance impact of SMS on scientific and commercial applications with respect to a baseline system without SMS. Figure 17 shows the performance improvement for each

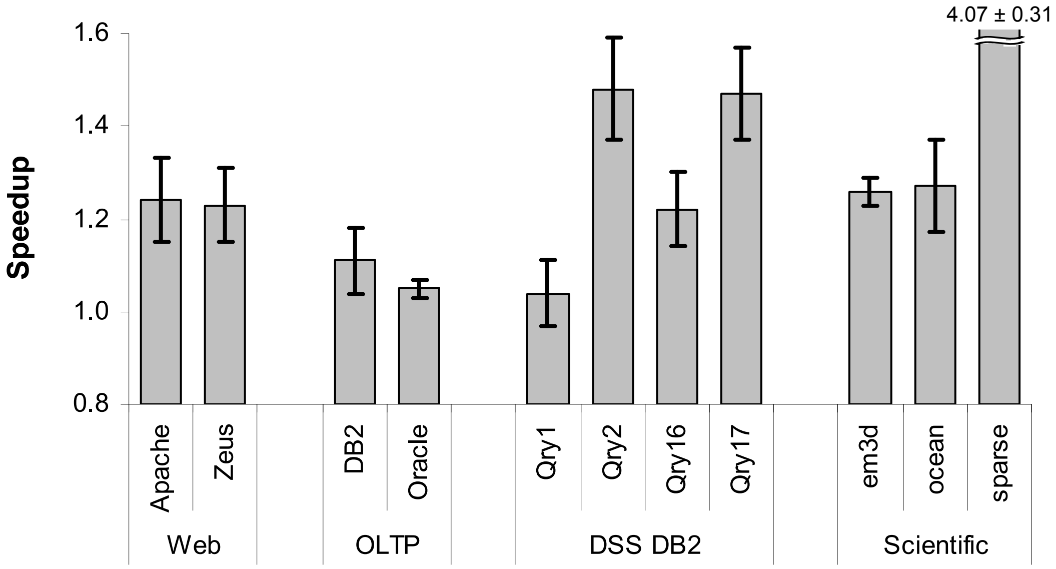

Figure 17: Speedup with 95% confidence intervals. Geometric mean speedup is 1.37.

application with 95% confidence intervals given by our sampling methodology. Figure 18 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).

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.

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 [8], 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.

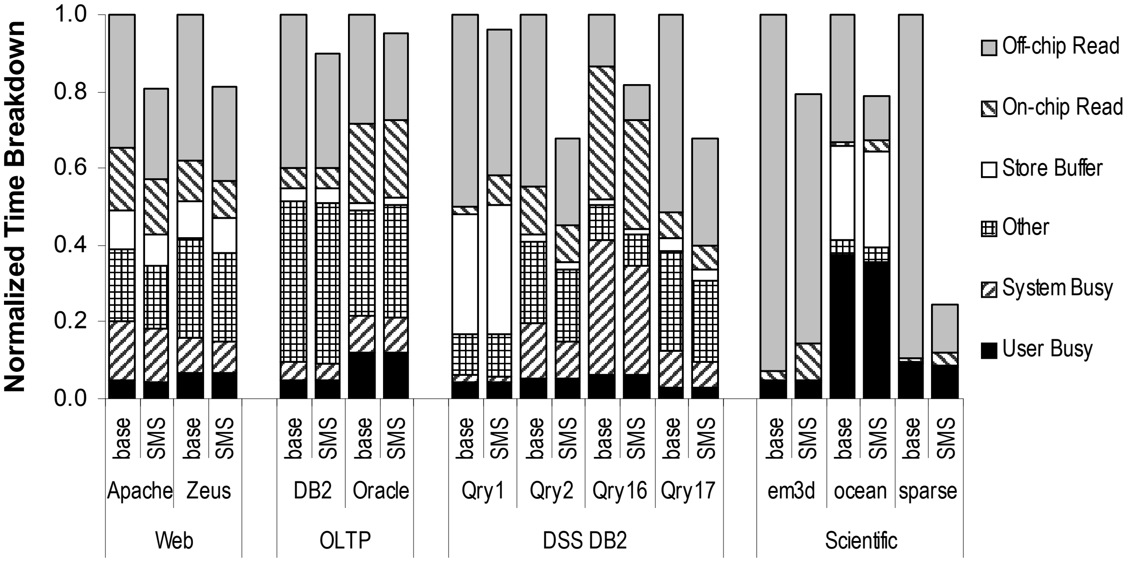

Figure 18: Time breakdown comparison. The base and SMS bars for each application are normalized to represent the same amount of completed work.

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

6. Related Work

Several prior proposals use offline analysis to exploit variations in spatial locality. Vleet et al. [31] propose offline profiling to select fetch size upon a miss. Guided region prefetching [32] uses compiler hints to direct both spatial and non-spatial (e.g., pointer-chasing) prefetches. However, the complex access patterns and rapid changes in datasets 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 [12], the spatial locality detection table [20], and caches that dynamically adjust block size [9,30]. 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.

Other proposals directly target spatial correlation. We discuss spatial footprints [23] and the spatial pattern predictor [7] in Section 3. Compared with SMS, these techniques utilize different training mechanisms that are not as proficient as the AGT at identifying repetitive spatial layouts. Neighborhood prefetching [22] captures spatial correlation, but predicts with PC-only indexing, which results in low accuracy. Density vectors (a.k.a., spatial patterns) are used in [24] to reject prefetches that are likely to be mispredictions. The authors start with an aggressive, inaccurate prefetcher and employ spatial correlation to minimize the impact of its inaccuracy. We suggest that spatial correlation is better utilized on its own, instead of in a tandem approach where spatial prediction does not directly yield coverage. Adaptive stream detection [19] dynamically adjusts prefetch aggressiveness to improve timeliness for regions with less dense spatial patterns.

Stealth prefetching [5] exploits coherence tracking at the spatial region granularity to fetch entire regions, if a region is not shared by other processors. This brute force approach is the antithesis of SMS and other intelligent spatial-correlating prefetchers in that it makes no attempt to predict access patterns within spatial regions. Predictor virtualization [4] proposes mechanisms to store predictor metadata in existing on-chip caches. The authors used the SMS prefetcher in their evaluation and found that predictor virtualization nearly obviates the need for dedicated pattern history storage (i.e., the PHT).

7. Conclusions

We have shown 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 established that code-based correlation is fundamentally superior to address-based correlation because it can predict previously-unvisited addresses. We demonstrated that decoupled tracking of spatial generations is superior to implementations that couple spatial observation to cache structures.

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. Even with 64KB of pattern storage per processor, SMS can be effectively implemented entirely on chip. Moreover, predictor virtualization [4] can reduce the dedicated storage to less than 1KB per processor by maintaining pattern history in the traditional memory hierarchy. 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 37% and at best 307%.

References

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

[2] J.-L. Baer and T.-F. Chen. An effective on-chip preloading scheme to reduce data access penalty. In *Proceedings of Supercomputing '91*, November 1991.

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

[4] I. Burcea, S. Somogyi, A. Moshovos, and B. Falsafi. Predictor virtualization. In *Proceedings of the Thirteenth International Conference on Architectural Support for Programming Languages and Operations Systems*, March 2008.

[5] J. F. Cantin, M. H. Lipasti, and J. E. Smith. Stealth prefetching. In *Proceedings of the Twelfth Conference on Architectural Support for Programming Languages and Operating Systems*, October 2006.

[6] L. Ceze, J. Tuck, P. Montesinos, and J. Torrellas. Bulk enforcement of sequential consistency. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, June 2007.

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

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

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

[10] 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 (Vol. I Architecture)*, August 1991.

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

[12] A. Gonzalez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *Proceedings of the SC95 Conference on Supercomputing*, July 1995.

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

[14] N. Hardavellas, I. Pandis, R. Johnson, N. G. Mancheril, A. Ailamaki, and B. Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research*, January 2007.

[15] 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, April 2004.

[16] R. Hedge. Optimizing application performance on Intel Core microarchitecture using hardware-implemented prefetchers. *http://www.intel.com/cd/ids/developer/asmo-na/eng/298229.htm*.

[17] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. Morgan Kaufmann, September 2006.

[18] J. Huh, J. Chang, D. Burger, and G. S. Sohi. Coherence decoupling: making use of incoherence. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2004.

[19] I. Hur and C. Lin. Memory prefetching using adaptive stream detection. In *Proceedings of the 39th Annual International Symposium on Microarchitecture*, December 2006.

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

[21] N. P. Jouppi. 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*, May 1990.

[22] D. M. Koppelman. Neighborhood prefetching on multiprocessors using instruction history. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, October 2000.

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

[24] W.-F. Lin, S. K. Reinhardt, D. Burger, , and T. R. Puzak. Filtering superfluous prefetchs using density vectors. In *Proceedings of the nternational Conference on Computer Design*, September 2001.

[25] A. M. G. Maynard, C. M. Donnelly, and B. R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1994.

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

[27] A. Seznec. Decoupled sectored caches. In *IEEE Transactions on Computers*, 46(2):210–215, February1997.

[28] 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*, October 2005.

[29] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. POWER5 system microarchitecture. *IBM Journal of Research and Development*, 49(4/5):505–521, 2005.

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

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

[32] 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 Annual International Symposium on Computer Architecture*, June 2003.

[33] T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Mechanisms for store-wait-free multiprocessors. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, June 2007.

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

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

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

[37] T.-Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24th Annual International Symposium on Microarchitecture*, December 1991.