Spatio-Temporal Memory Streaming

Stephen Somogyi*, Thomas F. Wenisch†, Anastasia Ailamaki‡,* and Babak Falsafi‡
*Carnegie Mellon University †University of Michigan
‡Ecole Polytechnique Fédérale de Lausanne
http://www.ece.cmu.edu/~stems

ABSTRACT

Recent research advocates memory streaming techniques to alleviate the performance bottleneck caused by the high latencies of off-chip memory accesses. Temporal memory streaming replays previously observed miss sequences to eliminate long chains of dependent misses. Spatial memory streaming predicts repetitive data layout patterns within fixed-size memory regions. Because each technique targets a different subset of misses, their effectiveness varies across workloads and each leaves a significant fraction of misses unpredicted.

In this paper, we propose Spatio-Temporal Memory Streaming (STeMS) to exploit the synergy between spatial and temporal streaming. We observe that the order of spatial accesses repeats both within and across regions. STeMS records and replays the temporal sequence of region accesses and uses spatial relationships within each region to dynamically reconstruct a predicted total miss order. Using trace-driven and cycle-accurate simulation across a suite of commercial workloads, we demonstrate that with similar implementation complexity as temporal streaming, STeMS achieves equal or higher coverage than spatial or temporal memory streaming alone, and improves performance by 31%, 3%, and 18% over stride, spatial, and temporal prediction, respectively.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design styles—*cache memories*

General Terms

Design, experimentation, performance

Keywords

Prefetching, spatial correlation, temporal correlation

1. INTRODUCTION

The memory system remains a bottleneck in modern computer systems. Although processor clock frequencies are increasing less rapidly than in the past, high memory/interconnect latencies cause execution to stall for hundreds of cycles during an off-chip miss, accounting for one-half to two-thirds of execution time [1,10,11,22]. Traditionally, designers have used larger caches to mitigate the impact of off-chip accesses, but this approach provides diminishing returns in today's multi-megabyte caches and is less appealing in chip multiprocessors (CMPs) where the silicon area can be used instead for additional cores.

CMPs themselves do not solve the memory bottlenecks of traditional multi-chip multiprocessors. While some communication occurs on chip with low latency, inter-chip communication misses remain costly, as do off-chip capacity misses to DRAM. Multithreading [16] can potentially overlap off-chip memory stalls; however, multithreading is only effective when additional threads are available (which may not be the case in commercial server applications [11]) and does not improve response time.

One approach for reducing the performance impact of off-chip accesses is to prefetch the data. While simpler techniques such as stride prefetching [14,19] have been shown to be effective for scientific, desktop and engineering applications, they are largely ineffective for commercial workloads. However, recent research with more sophisticated prefetchers has shown promise for these workloads. Temporal address-correlating prefetchers [6,13,17,20,26] predict recurring sequences of misses, which arise as applications iterate over data structures, even arbitrarily irregular ones common in commercial workloads. These prefetchers exploit the observed order between misses, but cannot prefetch deeply because large-scale data traversals do not repeat perfectly. In contrast, spatial-correlating prefetchers [4,15,21] predict repetitive spatial layouts over contiguous regions of memory, which arise when applications organize data at a page granularity. These prefetchers are more accurate and can predict compulsory misses, but do not establish order among predictions and are limited in prefetch depth.

Temporal and spatial prefetchers each target different memory system behaviors; many of the temporally predicted accesses are not predicted spatially, and vice versa. In this paper, we show significant opportunity to exploit temporal and spatial correlation concurrently. Rather than simply placing existing spatial and temporal predictors side by side, an intelligent spatio-temporal hybrid design can overcome some of the underlying techniques' limitations. In particular, spatial prefetching suffers from its inability to predict the first miss to each region, and regions are restricted to a fixed size. Temporal prefetching suffers from low accuracy because it does not know where streams terminate and it cannot predict compulsory misses.

In this paper, we propose Spatio-Temporal Memory Streaming (STeMS) as one approach for exploiting both temporal and spatial correlation. STeMS temporally captures the sequence of accesses to different regions, and spatially captures accesses within each region. The key innovation in this paper is to combine temporal and spatial predictions into a single total predicted sequence. This unified approach prevents the predictors from interfering with each

ISCA'09, June 20–24, 2009, Austin, Texas, USA.
Copyright 2009 ACM 978-1-60558-526-0/09/06...$5.00.

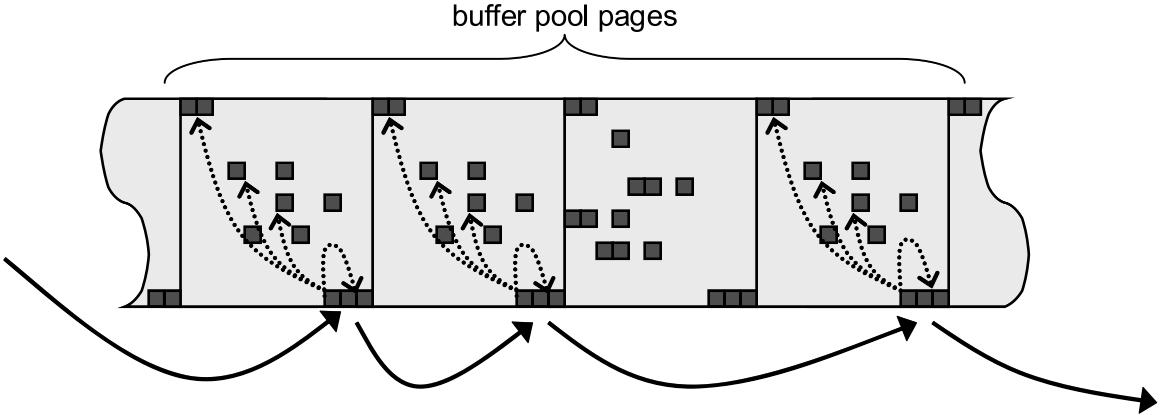


FIGURE 1. Example of temporal and spatial correlation in a buffer pool. Dark squares represent memory accesses. The solid arrows show a temporal sequence across pages; the dotted arrows show spatial accesses within pages.

other, enhances lookahead for spatial accesses and facilitates simple streaming mechanisms. To reconstruct the total miss sequence accurately, STeMS leverages new observations about the temporal characteristics of spatial correlation: that the sequence of accesses within and across spatial regions is repetitive.

STeMS achieves equal or higher prediction coverage and speedup than either underlying prefetcher alone, and compared with a naive combination of the two, STeMS drastically improves prefetch accuracy. STeMS is also effective for our entire suite of commercial workloads, including decision support systems (DSS, for which temporal streaming does not work) and online transaction processing (OLTP, for which spatial prediction does little). We evaluate STeMS with memory traces and cycle-accurate simulation to demonstrate:

- **Opportunity for synergy between spatial and temporal correlation.** We show that each correlation predicts different memory accesses: on average, 32% (temporal), 54% (spatial) and 70% (joint spatio-temporal).
- **Spatio-temporal streaming.** We show that 47% of misses at a spatial-region granularity recur in repetitive sequences, similar to repetition in the sequence of all misses (45%). We show that the access sequence within spatial regions is also repetitive; over 86% of accesses recur within a reordering window of two, and 92% within a window of four.
- **Hardware design.** We describe a practical hardware design for STeMS in a multiprocessor server system. STeMS predicts on average 62% of off-chip read misses but mispredicts an additional 29%. In cycle-accurate, full-system timing simulation, STeMS yields mean speedups of 31% over a baseline system, or 18% and 3% over systems with temporal and spatial memory streaming, respectively.

The remainder of this paper is organized as follows. In Section 2, we discuss temporal and spatial correlation in detail. We discuss the synergy behind spatio-temporal streaming in Section 3 and present a hardware STeMS design in Section 4. We evaluate our design in Section 5. In Section 6, we discuss related work and then conclude in Section 7.

2. BACKGROUND

We first review the fundamental concepts of temporal and spatial correlation and provide an overview of hardware implementations to exploit them, namely Temporal Memory Streaming (TMS, [26]) and Spatial Memory Streaming (SMS, [21]).

2.1 Temporal Correlation

We use *temporal correlation* to refer to a pair of phenomena: that sequences of misses are likely to repeat and that recent sequences are more likely to repeat than older sequences. The intuition behind these ideas is that applications tend to access data in a repetitive manner, so the access sequence is likely to repeat. As data structures are modified, a record of the most recent traversal is most likely to accurately reflect a path through a particular structure. Figure 1 shows an example of a simple temporal traversal among pages in a database buffer pool.

Temporal correlation exhibits several advantageous properties. Because the recorded miss sequence includes all misses from a thread, a processor only follows a single sequence when prefetching, as opposed to interleaving misses from multiple sequences. Temporal correlation is ideal for accelerating chains of dependent misses (i.e., pointer chasing), because sequences contain the miss addresses themselves, allowing a predictor to fetch the elements of a dependence chain in parallel rather than sequentially. Finally, temporal sequences are frequently long [24] (hundreds of misses), thus amortizing the startup cost associated with locating/following a new sequence.

Temporal correlation also has limitations. Because it relies on address repetition, it cannot predict previously unobserved addresses (i.e., compulsory misses) that are common in applications that scan large data sets. Memory addresses can exist as part of many different traversals, and temporal correlation may not be able to identify the best sequence to follow. Finally, training can be slow because a particular code path over a particular data structure must recur before it is predictable.

2.2 Temporal Memory Streaming

TMS is a hardware design that exploits temporal correlation. It records the observed miss sequence in a large circular buffer that must be stored in main memory because of its size (~2MB per processor). On an unpredicted off-chip miss, TMS locates the most recent occurrence of the miss address in the buffer and proceeds to prefetch several cache blocks whose addresses follow. As prefetched blocks are consumed, TMS reads more addresses from the buffer and streams more data. By maintaining a constant number of fetched blocks, TMS throttles its streaming to match application demand.

A key challenge for TMS is locating a particular miss address in the circular buffer, so that it can commence streaming. TMS therefore requires a mechanism to map an address to its most recent occurrence in the buffer, and update the corresponding mapping after every append. These mappings can be maintained by extending directory entries in directory-based multiprocessors [26], or in a main-memory hash table [25] for other systems.

TMS has been shown to be effective for OLTP and web serving. One strength of TMS is its ability to parallelize dependent misses that are prevalent in these pointer-chasing workloads, thereby increasing memory-level parallelism. In contrast, TMS is mostly ineffective for DSS workloads, which are dominated by scans of previously untouched data.

2.3 Spatial Correlation

We use *spatial correlation* to refer to the phenomenon that memory accesses occur in repetitive spatial patterns—that the same offsets, relative to some base address, are accessed. Spatial correlation arises because applications use data structures composed of many objects with a fixed layout, and a traversal is likely to touch the same elements within each object as it walks the structure. See Figure 1 for an example.

A key advantage of spatial correlation comes from its use of relative offsets instead of complete addresses. Rather than requiring a traversal to repeat over a particular set of addresses, spatial correlation can apply a pattern observed in one part of memory to a similar object allocated anywhere else in the memory space. This property allows spatial correlation to predict compulsory misses. Additionally, offsets reduce storage requirements because they are more compact than complete addresses.

An important weakness of spatial correlation is its inability to capture pointer-based dependence: because dynamic objects can be allocated anywhere in the memory space, pointers between two such objects have no inherent spatial relationship. Another shortcoming of spatial correlation stems from its relatively high startup costs. Whereas temporal correlation amortizes the startup cost over potentially unbounded sequences, spatial correlation can amortize only over the limited number of accesses that comprise a pattern (restricted, for example, by the OS page size). In workloads with less dense spatial patterns, a single unpredictable block at the beginning of each pattern can account for a significant fraction of all misses, thereby reducing opportunity for spatial correlation.

2.4 Spatial Memory Streaming

SMS is a hardware prefetcher that exploits spatial correlation. SMS observes spatial patterns at the L1 data cache and stores them in an on-chip history table for later prediction. When a region is first accessed, SMS uses information about the miss (e.g., address, PC) to look up a previously observed pattern in the table. If a pattern is found, SMS calculates the set of addresses for the current region that would yield the same pattern, and fetches these blocks into the L1 cache.

SMS logically partitions the memory space into fixed-size *spatial regions* of 2KB (32 cache blocks), encoding patterns as 4-byte bit vectors. Because a bit vector represents all 32 blocks in a region, computing prefetch addresses is straightforward, given the base address of a region and which bits are set in the vector.

A primary challenge in spatial prediction is delineating spatial patterns in both space and time. In space, SMS use the fixed-size regions discussed above. In time, SMS observes accesses over epochs called *spatial generation*s, which begin with the first (a.k.a., trigger) access to an inactive region and end when one of the accessed blocks is evicted or invalidated from the L1 cache.

The second key design choice for spatial prediction is how to associate a spatial pattern with its trigger access. SMS uses the PC (program counter) of the trigger instruction, thus correlating a pattern with the program code responsible for generating that pattern. Code correlation allows SMS to apply a pattern learned in one spatial region to many others. Consequently, SMS trains quickly and can predict compulsory misses. Furthermore, storage requirements scale with the size of the program code footprint rather than the size of the data set, allowing predictor data (~64KB per processor) to reside on chip.

SMS works well for DSS and web serving workloads. DSS queries perform scans over large amounts of data, contained in database pages that all share the same layout. Because these pages are traversed by the same code, SMS rapidly learns the spatial access patterns. In contrast, SMS is less effective for OLTP because many accesses that are spatially predictable are already issued in parallel by out-of-order processing.

3. SPATIO-TEMPORAL STREAMING

Temporal and spatial correlation are each effective for exploiting different aspects of program behavior. Temporal correlation captures dependence chains and pointers or other sequences that cover large portions of the memory space. In contrast, spatial correlation captures distinct access patterns for different program behaviors over restricted regions of memory. Independently, neither is capable of capturing the diverse set of memory behaviors in programs.

The goal with spatio-temporal streaming is to enable efficient streaming of both temporally and spatially correlated memory behaviors. Consider, for example, a non-clustered index scan through a database table (Figure 2). Logically, the scan proceeds sequentially through database pages that comprise the table. In actuality, these pages may be scattered throughout the buffer pool, as each was allocated to the next free location when read from disk. Within each page, a repetitive access sequence emerges—page ID, lock bits, slot indices, and finally data. When the database is finished with one page, it moves to the next. Therefore, the sequence of page accesses repeats for any scan through this table, and the accesses within each page repeat for a particular scan.

To predict access patterns as in the scan described above, a predictor should integrate knowledge of large-scale behavior (e.g., algorithms, page allocations) with small-scale details (e.g., fields within a structure, elements within a page). In this section, we introduce one approach to designing such a predictor. We then discuss details of our hardware implementation in Section 4.

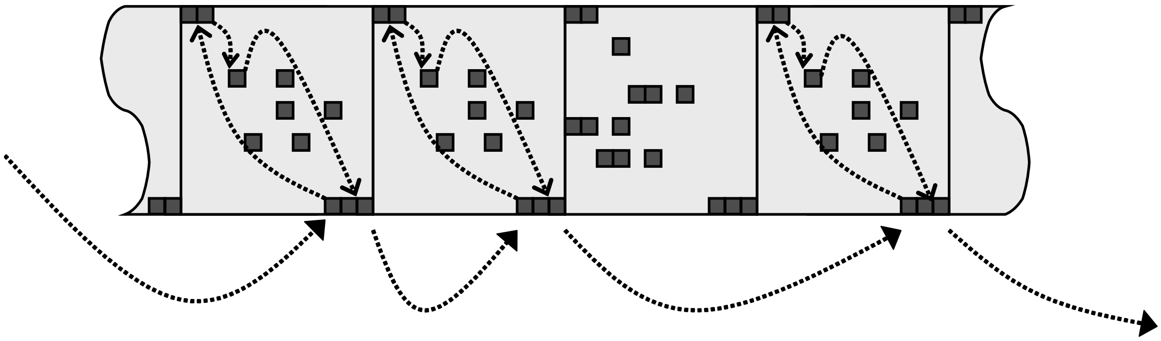


FIGURE 2. Motivating example for spatio-temporal streaming: a database index scan. The order of page accesses is arbitrary but repetitive (temporal behavior). Accesses within pages repeat (spatial behavior). Overall, the scan consists of a global access sequence that is predictable using temporal and spatial correlation together.

3.1 Hybrid Spatio-Temporal Prediction

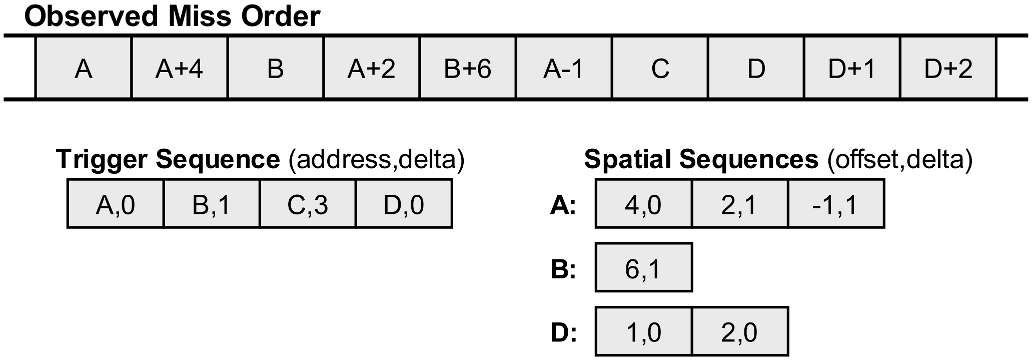
We first discuss the most straightforward hybrid implementation. Instead of recording the sequence of all misses as in TMS, this hybrid design records the sequence of only spatial triggers (i.e., the first access within a spatial region during each generation). The spatial predictor trains as in SMS. Then on an off-chip miss, the hybrid looks up the address in the trigger sequence and proceeds to fetch the blocks whose addresses follow. As each block is fetched, it triggers a spatial lookup and elements of the predicted spatial pattern are also fetched.

While this approach successfully predicts both temporal and spatial relationships, it overwhelms the memory system because the spatial patterns predicted in rapid succession are prefetched simultaneously. The resulting burst in bandwidth demand causes contention in the memory system, delays prefetches, and pollutes the cache or overflows the prefetch buffer. SMS does not encounter this problem because it observes generations as the program executes, naturally spreading spatial predictions out in time. Prior work [21] has shown that many spatial generations are active in parallel, with accesses interleaved across generations. A naive hybrid design has no notion of priority among generations and thus would fetch spatially predicted blocks in a different order than the processor requests them.

The problem with the above implementation is that the predictor lacks knowledge of which particular block the processor will request next, out of the pool of spatially and temporally predicted addresses that will be requested “soon.” To enable effective spatio-temporal prediction, we must order all predictions into a single sequence that the processor will follow. We can construct such an ordering by exploiting temporal characteristics of spatial accesses: that accesses repeat both within and across spatial generations.

Our fundamental innovation in STeMS is to reconstruct the total miss order by interleaving predictions from different spatial regions into a single predicted sequence. To reconstruct the total ordering, STeMS maintains the order of spatial triggers (the same idea as temporal streaming), the order within spatial regions (new in this design) and the relative interleaving of individual spatial and temporal accesses across concurrently traversed regions.

In Figure 3, we decompose a miss sequence into its relative interleavings. The miss order consists of triggers (i.e., the initial misses in every region, denoted by *X*) and spatial accesses (i.e., subsequent accesses within generations, denoted by *X*±*n*). The temporal sequence then consists only of the triggers. Additionally, for each trigger we record its temporal *delta*—the number of entries skipped in the global miss order. For example, *D* immediately follows *C*, so its delta is zero; *B* skips one element after *A*, its delta is one. Within each spatial region, we extract the offsets of the blocks that are accessed, along with the delta for each that preserves its relative position in the global miss order. Clearly, given this collection of trigger and spatial sequences, we can reconstruct the original miss order. This reconstruction is the fundamental mechanism behind STeMS.



Observed Miss Order

A	A+4	B	A+2	B+6	A-1	C	D	D+1	D+2

Trigger Sequence (address,delta)

A,0	B,1	C,3	D,0

Spatial Sequences (offset,delta)

A:	4,0	2,1	-1,1
B:	6,1		
D:	1,0	2,0	

FIGURE 3. Decomposition of total miss order into temporal and spatial elements. The temporal component is the subsequence of spatial triggers; the spatial component comprises the access sequence for each spatial region. Deltas record the relative interleaving among sequences.

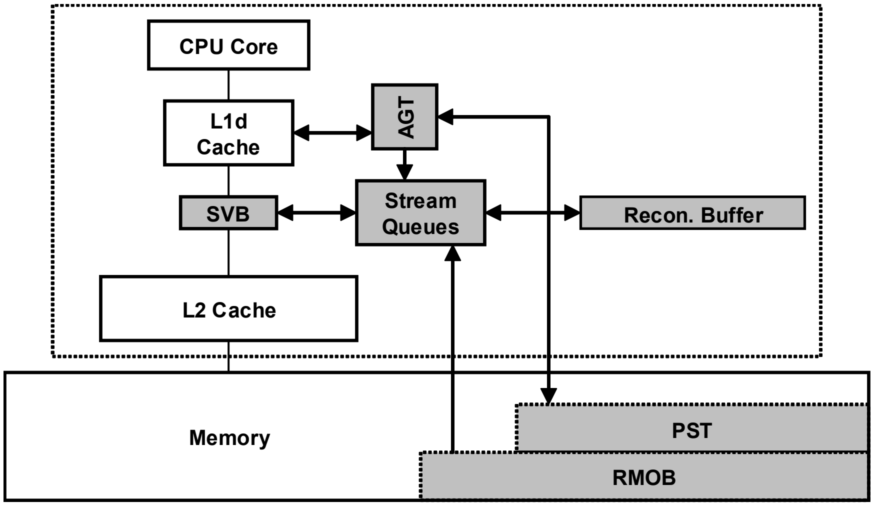


FIGURE 4. System overview. Shaded components are new or have been modified for STeMS.

Compared with temporal streaming, STeMS additionally predicts spatial accesses, and reduces storage requirements for the temporal sequence because only spatial triggers are recorded. Compared with spatial streaming, STeMS predicts trigger accesses, but increases history storage requirements because the sequence of spatial accesses must be recorded instead of merely the pattern.

4. HARDWARE IMPLEMENTATION

In this section, we present a hardware design for Spatio-Temporal Memory Streaming (STeMS). Figure 4 illustrates the location of STeMS components in one node of a multiprocessor system. As with other predictors, STeMS comprises two operations that go on concurrently: *training*, to observe and store memory behaviors; and *prediction*, to generate addresses for prefetch. We first discuss training, then prediction, and finally the hardware costs associated with each component.

4.1 High-level Operation: Training

The spatial and temporal prediction mechanisms each train in a similar fashion as the standalone SMS and TMS predictors.

Spatial predictor trains independently. The predictor observes all L1 accesses, the *active generation table* (AGT) accumulates accesses to each spatial region over the course of a generation (from first access until a block is replaced from the cache), and upon generation termination, the *pattern sequence table* (PST) stores the observed spatial sequence. STeMS differs from SMS in that it records sequences rather than bit vectors and the PST must reside off chip because sequence information inflates its size.

Region misses recorded in circular buffer. Much like TMS, STeMS records the temporal miss sequence of (spatial) triggers in an off-chip circular buffer—the *region miss order buffer* (RMOB). In STeMS, however, misses that are spatially predictable are omitted from the temporal sequence. Hence, prior to appending a miss to the RMOB, STeMS queries the spatial predictor, and only performs the append for trigger accesses and spatial misses. Each RMOB entry contains the block address, the PC of the miss instruction, and the reconstruction delta.

4.2 High-level Operation: Streaming

The streaming mechanisms operate very similarly to TMS: off-chip misses can initiate new streams and the prefetcher throttles streaming to match application demand. The key difference is that TMS reads the address sequence directly from the circular buffer, while STeMS must reconstruct its prediction sequence.

Lookup on off-chip miss. Upon every off-chip miss, STeMS identifies the most recent occurrence of the address in an RMOB and sends subsequent RMOB entries to the requesting processor. As entries are consumed by the reconstruction process (see below), STeMS fetches additional RMOB entries so that reconstruction can resume and the stream can continue as long as possible.

Reconstruction. STeMS constructs a total predicted miss sequence using both temporal and spatial predictions. We illustrate the reconstruction process in Figure 5. First, STeMS places the initial miss address at the start of a reconstruction buffer. Second, the predictor inserts the addresses from subsequent RMOB entries into the buffer, leaving as many empty spaces as their deltas indicate. Third, STeMS calculates the spatial lookup index for each RMOB entry using the address and PC, and looks up this index in the pattern sequence table. If found, for each element in the spatial sequence, STeMS calculates the address (using the recorded offset and the region address of the trigger) and inserts it into the reconstruction buffer according to its delta.

Streaming. After reconstruction, STeMS moves the sequence of addresses to a stream queue and fetches predicted cache blocks to the requesting processor in order, placing them in a *streamed value buffer* (SVB). When a block is consumed from the SVB (i.e., a prefetch hit), STeMS fetches the next block according to the stream queue. To reduce erroneously fetched blocks due to invalid streams, only a single block is fetched at the beginning of a new stream. If the block is consumed, the stream is likely to be useful, and further blocks are fetched. When the number of available prefetch addresses in a queue drops below a threshold, STeMS resumes reconstruction from where it left off previously, adding more addresses to the end of the stream queue.

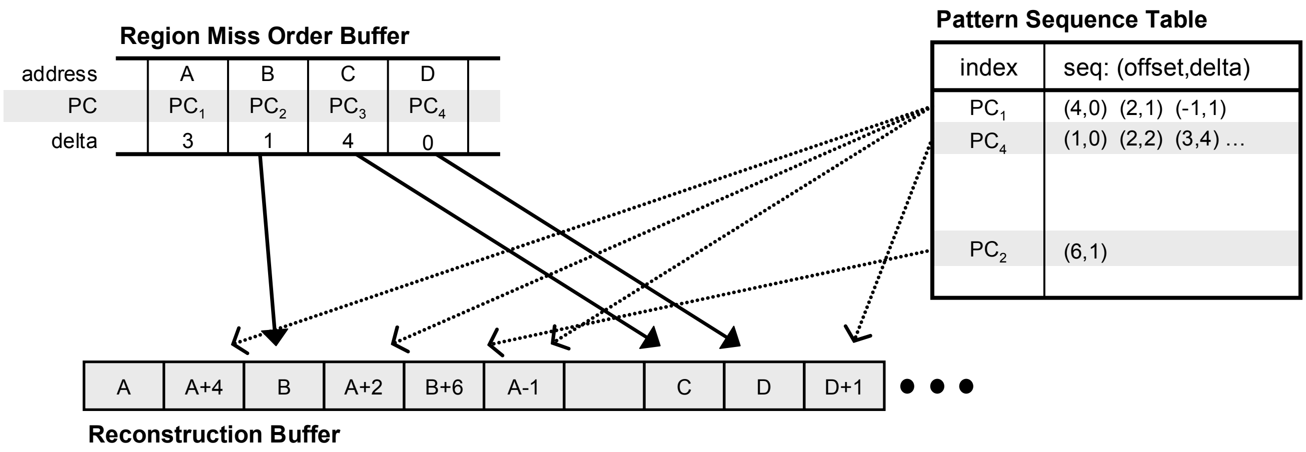


FIGURE 5. Example of reconstruction. Temporal elements from the RMOB are placed in the reconstruction buffer. Each RMOB entry triggers a spatial lookup, and the resulting spatially predicted addresses are interleaved into the overall sequence according to their reconstruction deltas.

Spatial-only streams. To achieve any coverage on compulsory-miss regions (e.g., pages that have never been seen before), STeMS must support spatial-only streams. During reconstruction, when STeMS queries the spatial predictor for each RMOB entry, the AGT remembers the lookup index for each region. During program execution, as spatial generations begin, the lookup index of the trigger access for each generation is computed and compared to the reconstruction index. If they differ, or if the region was not predicted during reconstruction, STeMS initiates a spatial-only stream using the spatial sequence contained in the PST for the correct index. STeMS treats these spatial-only streams like reconstructed streams, except it ignores the delta information.

4.3 Hardware Cost

Spatial prediction. We make one significant change to the SMS design of [21]: instead of simple bit vectors, the history table stores vectors of 2-bit saturating counters, one per block. Many patterns (as identified by their prediction index) contain both stable and unstable accesses. The stable accesses repeat with high probability, leading to good predictions, but the unstable ones do not, generating mispredictions. Hysteresis afforded by the saturating counters allows spatial history to learn the stable parts of each pattern. Compared with bit vectors, 2-bit counters attain the same coverage while roughly halving overpredictions. All SMS/STeMS results in this paper assume saturating counters in the spatial history table.

Ignoring storage per entry, STeMS requires the same number of entries in the AGT and PST as SMS. However, both structures must maintain the sequence of accesses rather than a simple bit vector. Each block can only appear once in a sequence, corresponding to the order in which it was first accessed. A spatial sequence requires 32*10 bits = 40 bytes: for each of 32 blocks, 2 bits for the saturating counter value and 8 bits for the reconstruction delta. Thus, an AGT (64 entries) requires 2.5KB of SRAM. With 16K entries, the PST requires 640KB per processor, so this data must be stored in main memory. In contrast, standalone SMS requires 0.4KB for the AGT and 64KB for the PHT.

Temporal prediction. The RMOB records off-chip miss addresses. In addition to the 5-byte physical address, in STeMS each entry contains 16 bits for the PC and 8 bits for the delta, totalling of 8B per entry. Because some misses are filtered (i.e., spatial hits), the overall size of the buffer is reduced from 384K entries (2MB) for TMS to 128K entries (1MB) for STeMS (all sizes per processor). For scientific applications that exhibit very specific RMOB requirements (i.e, where the RMOB must capture the miss sequence of an entire iteration to provide any coverage) and dense spatial patterns, the reduction can be even more significant.

Reconstruction. The reconstruction process requires a finite state machine (capable of bit shifts and 10-bit addition) and temporary storage (the reconstruction buffer, 256 entries). To accommodate minor reordering during reconstruction, if STeMS tries to place an address in an entry that has already been populated, it searches for an adjacent free space. We find that searching at most two elements forward or backward allows 99% of addresses to be placed (92% in their original location).

Streaming. Even though only one stream is typically productive at any time, several stream queues are necessary to prevent thrashing when new streams are initiated on misses. STeMS orders the streams based on activity (i.e., prefetches and hits) and chooses the LRU stream as a victim when a new stream must be allocated. We use eight stream queues in our evaluation. Stream lookahead is the number of blocks per stream that STeMS tries to maintain in the SVB, and is important because it controls timeliness and mispredictions (particularly at the end of streams). We use a lookahead of eight for commercial workloads [26], but 12 for our scientific applications, which exhibit higher BW requirements. Finally, we use a 64-entry SVB for all evaluations in this paper.

5. EVALUATION

In the following subsections, we evaluate and analyze spatio-temporal memory streaming and compare against the underlying temporal and spatial prediction techniques.

5.1 Methodology

We evaluate STeMS using cycle-accurate full-system simulation of a shared-memory multiprocessor using FLEXUS [27]. FLEXUS models the SPARC v9 ISA and can execute unmodified commer-

TABLE 1. System and application parameters.

Processing Nodes	UltraSPARC III ISA 4 GHz 8-stage pipeline; out-of-order 4-wide dispatch / retirement 96-entry ROB, LSQ ASOtso memory model
L1 Caches	Split I/D, 64KB 2-way, 64B blocks 2-cycle load-to-use, 3 ports, 32 MSHRs
L2 Cache	Unified, 8MB 8-way, 64B blocks 25-cycle hit latency, 1 port
Main Memory	3 GB total memory, 40 ns access latency 64 banks per node 64B 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
Stride Prefetcher	32-entry buffer, max 16 distinct strides

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

cial 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 configure our processor model to approximate the hardware resources of the Intel® Core 2 microarchitecture. We use a store-wait-free memory model [23] to minimize the penalty of stores and memory ordering instructions, thus exposing more off-chip read stalls compared with a conventional TSO system. 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 tuned database parameters, (e.g., number of client processes), to maximize performance for a conventional system under TSO. We select three queries from the TPC-H DSS workload based on the categorization in [18]. All three 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 assure that the server system is fully saturated (client activity is not included in timing results). Finally, we include three scientific applications to provide a frame of reference for our commercial application results.

We analyze memory traces for our workload characterization and initial predictor results. We collect traces in Flexus with in-order execution and no memory stalls. For the OLTP and web workloads, we warm main memory for over 5000 transactions or requests, then collect traces of five billion instructions. For DSS, we analyze queries 2 and 17 in their entirety, and trace 5B instructions of query 16. For scientific applications, we trace one entire iteration after warming the system for three iterations.

We obtain performance results using the SimFlex multiprocessor sampling methodology [27]. The SimFlex methodology extends the Smarts [28] statistical sampling framework to multiprocessor simulation. Our samples are drawn over intervals of 10s to 30s of simulated time (as observed by the OS in functional simulation) for OLTP and web server applications, over the complete query execution for DSS, and over a single iteration for scientific applications. 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 for a period of 100,000 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 [27].

5.2 Comparing Temporal and Spatial Correlation

We first examine the similarities and differences between spatial and temporal correlation through their corresponding prediction mechanisms. We evaluate the breakdown of coverage in Figure 6, classifying each off-chip read miss as predictable by both techniques, only one, or neither.

In OLTP and web, each of the four categories accounts for a significant fraction of misses, corroborating prior results ([21,26]) that both predictors can benefit these workloads, with OLTP slightly biased towards TMS and web serving towards SMS. The large fraction of misses that are predictable by only one technique is the key motivation behind STeMS: a hybrid predictor should be able to capture most of this joint coverage. In contrast, 34–38% of misses remain unpredictable by either technique—these are outside the scope of what STeMS attempts to predict.

In DSS, temporal streaming is largely ineffective, while spatial streaming predicts over 60% of misses. The majority of these misses are compulsory, so TMS is fundamentally unable to predict them. Because these results demonstrate little opportunity for a hybrid design to improve over SMS, the goal for STeMS is to achieve the same performance as with SMS, while maintaining benefits for other workloads.

Both predictors achieve high coverage for the scientific applications, but TMS is essentially perfect, whereas SMS cannot disam-

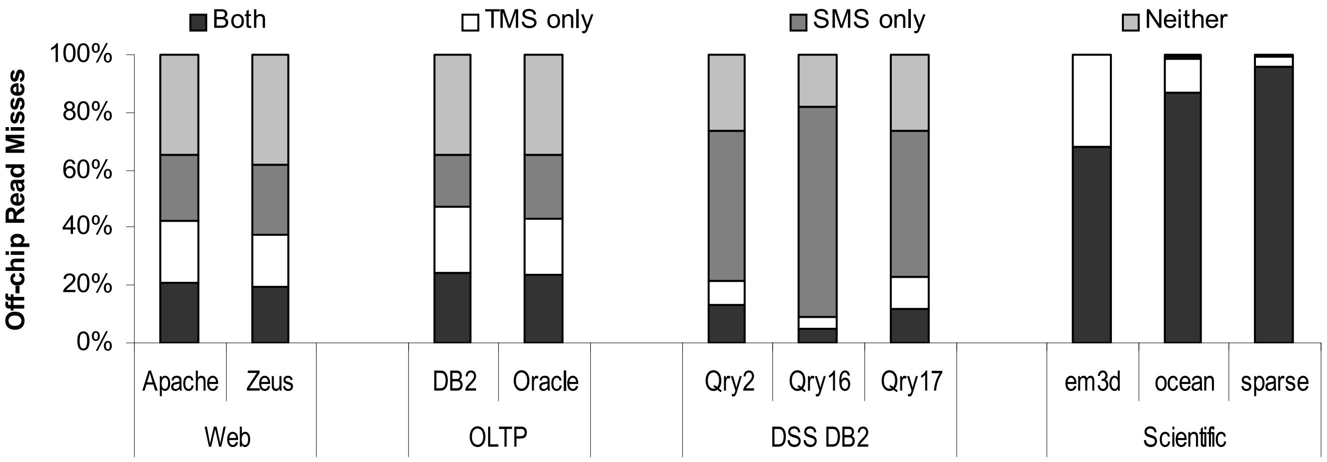


FIGURE 6. Joint analysis of temporal and spatial memory streaming. Each read miss is classified as predicted by both predictors, only one, or neither.

biguate spatial patterns for certain behaviors (it could with an address-based prediction index, however). Again, the goal for STeMS is attain the same performance as the better of the underlying predictors, because there is no opportunity for further improvement.

5.3 Temporal Correlation Across Spatial Regions

STeMS relies on repetition in the sequence of spatial triggers as its basis for reconstruction. Prior work [26] has demonstrated temporal repetition in sequences of miss addresses. Because spatial triggers are a subset of all misses, and given knowledge of application behaviors, we expect triggers to exhibit a similar level of temporal correlation.

Like prior studies of repetition in L1 data streams [5,24], we use the Sequitur hierarchical data compression algorithm [9] to identify temporal repetition in address sequences. Sequitur constructs a grammar whose production rules correspond to repetitions in its input. Each production rule maps a label to a sequence of symbols and other rule labels. Sequitur operates by incrementally extending the grammar's root production rule by one symbol at a time. As each symbol is appended, the grammar is modified to create new production rules that capture any new repetition the appended symbol creates. Sequitur maintains two invariants as the grammar grows. First, no pair of symbols are adjacent more than once in the grammar. Second, every production rule in the grammar (except the root rule) is used more than once. As a result of these invariants, the grammar's rules correspond to distinct repetitive sequences.

We present results of our Sequitur analysis in Figure 7, analyzing the sequence of all misses (as in TMS) and the subset of misses that are also spatial triggers. We use the following categorization: *non-repetitive*: addresses that do not recur; *new*: the first occurrence of a repetitive sequence; *head*: the first element in subsequent occurrences; and *opportunity*: non-head elements in repetitive occurrences of a sequence.

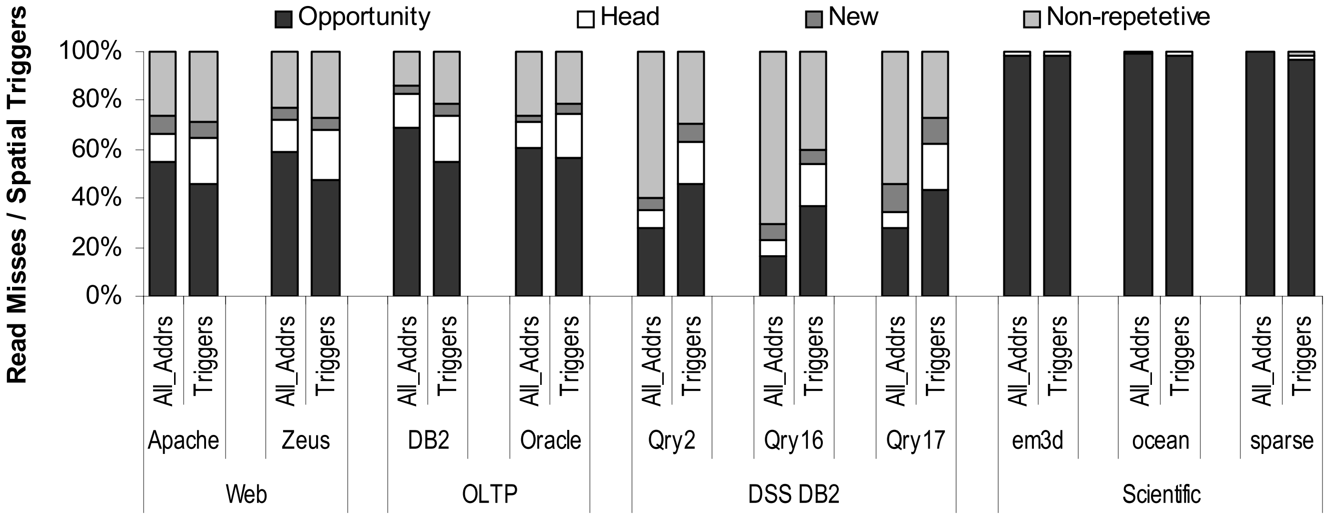


FIGURE 7. Temporal repetition of addresses and spatial triggers. All_Addrs evaluates the total sequence of read misses; Triggers only evaluates the subset that are spatial trigger events.

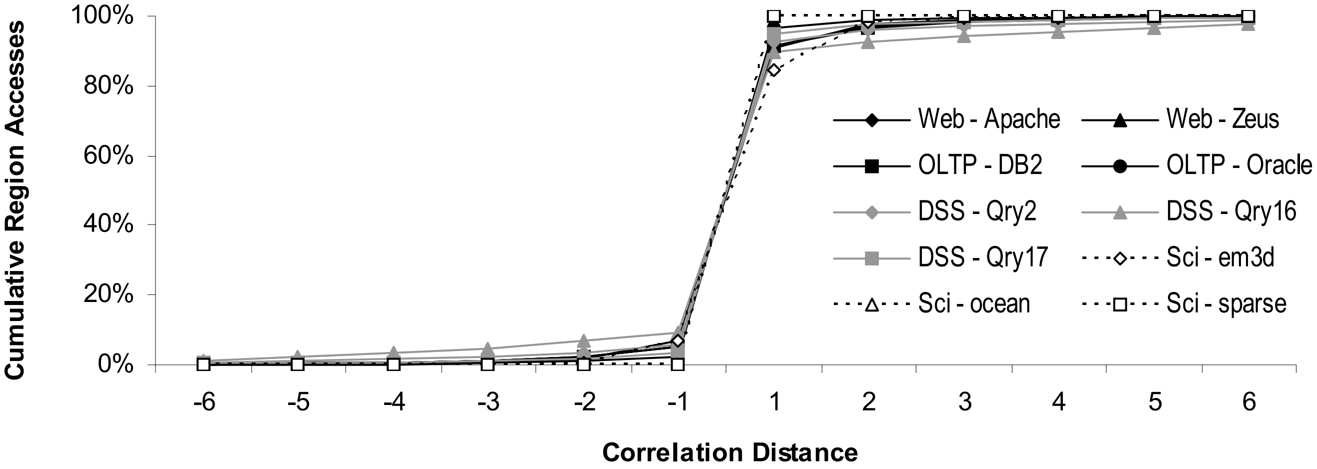


FIGURE 8. Temporal repetition within spatial generations. Correlation distance compares a sequence with the prior occurrence—a distance of +1 is perfect repetition; other distances represent a jump within the prior sequence for two accesses that are consecutive in the sequence under evaluation.

In OLTP and web, temporal correlation for spatial triggers is 5–15% lower than for all misses. Many of the spatial hits are also temporally correlated (the "both" category in Figure 6), so when they are removed from consideration, the resulting misses exhibit less repetition. We observe the opposite effect in DSS: because spatial hits are not temporally correlated but comprise a large fraction of misses, the leftover misses contain nearly all the temporal repetition. For all the commercial workloads, heads account for a larger fraction of triggers than of all misses, indicating that repetitive sequences are shorter. Overall, although spatial triggers exhibit slightly lower temporal correlation, they still form long repetitive sequences that can stand as the foundation for hybrid reconstruction.

5.4 Temporal Correlation Within Spatial Regions

Given a repetitive sequence of spatial triggers, we now show that the access sequence within spatial patterns is repetitive; this is the second property necessary for reconstructing a total miss sequence from temporal and spatial components.

We use correlation distance [26] in Figure 8 to quantify intra-generation sequence repetition (Sequitur is not suitable for this analysis because the access sequences within each spatial region must be evaluated separately, and Sequitur requires a single global order). We record the sequence of accesses within each generation, and at termination, compare against the prior occurrence of that generation (identified by the spatial lookup index). For every pair of subsequent accesses in the new sequence, we calculate the distance between those same two offsets in the prior sequence, and report this as the correlation distance. Thus a distance of +1 is perfect repetition and all other distances represent reordering. Although reorderings of up to 32 are possible, we show up to ±6 because this range accounts for 96% of spatial accesses in all of our applications.

Temporal repetition within spatial generations is nearly perfect. Over 92% of spatially predictable accesses recur with a reordering distance of 4 or less, and over 86% within a reordering window of only 2 (excluding DSS Qry16, these numbers rise to 96% and 92%, respectively). Temporal correlation is much stronger at this small scale than at the macro level (e.g., the opportunity portion of "all_addrs" in Figure 7). Because spatial patterns are code correlated, the same program code will execute during different occurrences of the same generation; with temporal correlation over all misses, there is no guarantee that the same code executes on multiple occurrences of sequences that begin with the same head address.

5.5 Memory Streaming Comparison

We present prediction results for STeMS in Figure 9, as well as compare against the prior temporal and spatial streaming techniques. Covered misses are successfully eliminated by prefetching (i.e., predicted correctly and still reside in the SVB at the time of the processor request). Overpredictions are erroneously fetched blocks, which cause bandwidth overhead and potentially pollute the SVB. These incorrect prefetches are normalized against the number of off-chip read misses in the baseline system.

In OLTP and web, STeMS predicts on average 8% more off-chip misses than the best of the underlying predictors, for coverage between 50% and 56%. In DSS, STeMS achieves essentially the same coverage as SMS. Across the commercial workloads, STeMS cannot capture all the opportunity suggested by the joint TMS-SMS results (Figure 6). This discrepancy is caused by imperfect reconstruction, which in turn is caused by imperfections in the temporal information of predicted sequences. Both temporal and spatial sequences repeat reliably at a high level, but glitches—reorderings, insertions, deletions—do exist. Moreover, the temporal deltas used in reconstruction are also imperfect (the deltas are roughly as likely to repeat perfectly as the sequences themselves). Together, these errors compound, leading to loss of coverage and increased overpredictions. Nevertheless, STeMS achieves at least the same coverage as the better of TMS or SMS across our entire suite of commercial workloads.

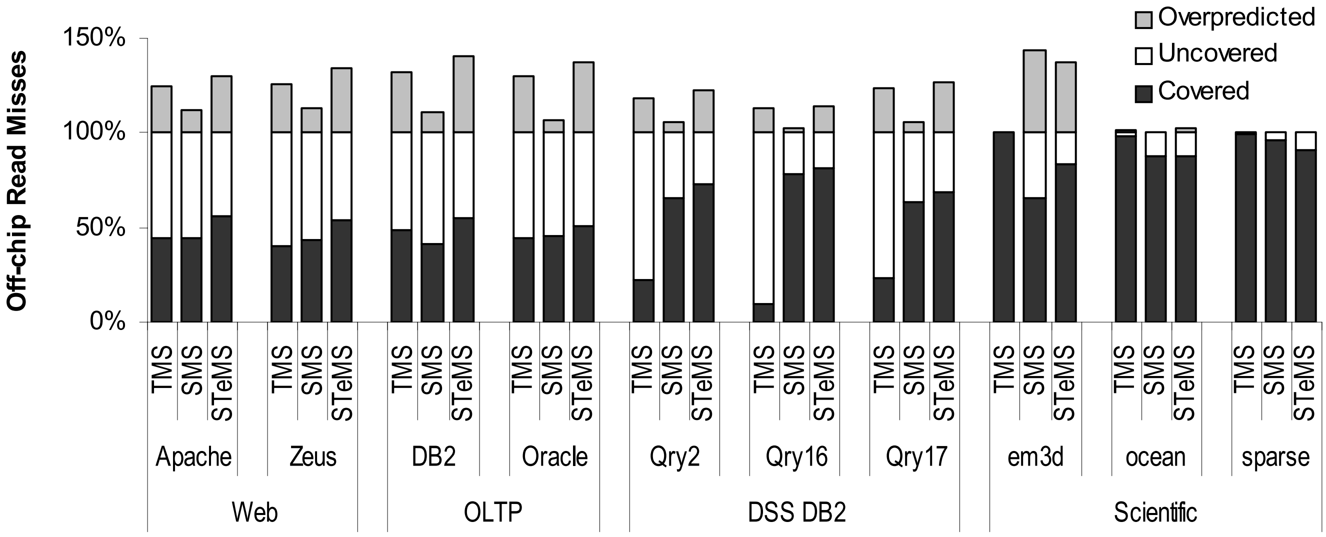


FIGURE 9. Comparison of temporal, spatial, and spatio-temporal memory streaming.

In the scientific workloads, it is difficult for STeMS to match TMS, which is effectively perfect. em3d is a good example of how hybrid reconstruction does not always recreate the original miss sequence. In em3d, the overall temporal sequence is perfectly repetitive, but jumps randomly over memory. Thus, with spatial prediction, the same trigger PC leads to many different spatial patterns. In STeMS, the temporal sequence contains only triggers, and reconstruction is unable to choose the "best" pattern to use for each trigger, so coverage falls between that of TMS and SMS. In sparse, both temporal and spatial sequences are repetitive, but several common spatial patterns toggle between two different delta sequences. Because incorrect deltas are used for some patterns during reconstruction, STeMS achieves lower coverage.

TMS and SMS overpredictions are nearly entirely disjoint, so we do not expect overpredictions with STeMS to be significantly lower than their sum. The one possibility for STeMS to improve overpredictions arises because the temporal sequence recorded in the RMOB is more accurate than in TMS, and thus generates fewer temporal overpredictions. However, many of the remaining overpredictions are spatial triggers, which result in mispredictions of entire spatial regions. The resulting spatial overpredictions negate the reduction in temporal overpredictions, leaving STeMS with the same or only slightly lower overpredictions than the sum of TMS and SMS.

We evaluated a system with TMS and SMS operating independently but concurrently. We found that although coverage approaches the combined coverage suggested in Figure 6, the predictors interfere with each other and generate roughly 2-3x the overpredictions of STeMS in OLTP and web. Thus, we did not pursue this approach further.

5.6 Performance Improvement

We show in Figure 10 the performance improvement of STeMS, along with TMS and SMS, over a baseline system with only a stride prefetcher. Our statistical sampling methodology produces the 95% confidence intervals shown in the graph. In general for the commercial workloads, STeMS matches or outperforms the underlying prefetchers, as expected given the coverage results from Section 5.5.

Each type of application shows different characteristics. In web serving, STeMS achieves a slight speedup advantage over the other predictors, corresponding to its higher coverage. Because Apache incurs more off-chip read stalls than Zeus, it benefits more from prefetching.

In OLTP, we corroborate prior results [21] showing that SMS offers little performance improvement despite its high coverage. Through hybrid reconstruction, STeMS achieves similar speedups as TMS despite the ineffectiveness of spatial prediction. Speedups are low in Oracle because the baseline system spends only one-quarter of time on off-chip memory accesses.

In DSS, STeMS matches the speedup of SMS. As seen from these results, temporal predictions have virtually no impact on performance, so the increased coverage that STeMS offers does not translate into improved performance over SMS.

For the scientific workloads, TMS achieves nearly perfect coverage, accelerating em3d and sparse by a factor of four or more. In em3d, STeMS outperforms SMS because of its higher coverage, but cannot match TMS—since memory-level parallelism increases drastically with perfect prediction, imperfections in the hybrid predicted sequence have an immediate negative impact on speedup. In ocean and sparse, STeMS outperforms SMS despite similar or lower coverage, demonstrating increased prefetch timeliness of the single predicted sequence over numerous independent spatial predictions.

6. RELATED WORK

Prefetching is an active research area for computer architects. STeMS clearly extends the work from [26] and [21] to overcome the limitations inherent with temporal and spatial correlation. Concurrent with our work, stream chaining [7] is a general approach for combining predictions that does not necessarily rely on spatial and temporal correlation. The authors note that although many

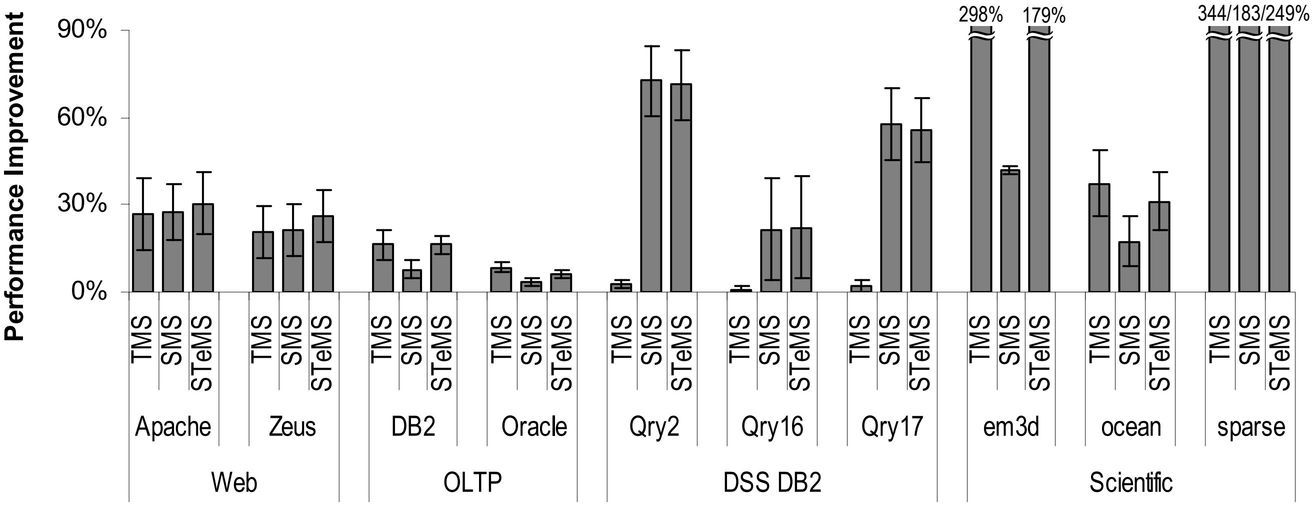


FIGURE 10. Performance improvement of STeMS compared with TMS and SMS. The error bars represent 95% confidence intervals.

prefetching techniques use localization to generate more predictable streams, the interaction between streams is unknown, making it difficult to issue prefetches at the optimal time. Stream chaining uses miss graphs to link together localized streams into predictable chains of memory accesses, thereby improving prefetch accuracy and lookahead. In contrast to STeMS, which attempts to directly construct the global miss sequence, stream chaining instead predicts which miss stream will be activated next in program order.

The earlier temporal streaming studies [5,20,26] have been extended directly by other proposals. Epoch-based prefetching [6] divides temporal sequences into epochs of parallelizable misses, and predicts only epochs for which the prefetches will be timely. The authors of [25] explored temporal streaming in the context of chip multiprocessors and proposed mechanisms to address the issues they encountered, mostly caused by limited off-chip bandwidth. Both the epoch and CMP-aware ideas are orthogonal and could be applied to the STeMS implementation in this paper. Instruction streaming [8] uses temporal correlation of instruction miss sequences to eliminate fetch stalls that bottleneck performance in applications with large code footprints.

Recent work in spatial prefetching includes adaptive stream detection [12], which dynamically adjusts prefetch aggressiveness to improve timeliness for regions with less dense spatial patterns. Stealth prefetching [3] exploits coherence tracking at the spatial region granularity to fetch entire regions, if a region is not shared by other processors. Predictor virtualization [2] proposed mechanisms to store predictor metadata in existing on-chip caches, nearly obviating the need for dedicated storage. This technique can be applied directly to the history structures used by STeMS.

Although the focus of our work is on hardware prefetching, software prefetching benefits from similar observations. Precomputation threads have been enhanced to prefetch irregular data structures [30], similar to spatial prefetching. Self-repairing prefetchers [29] track loads that frequently miss and dynamically adjust lookahead to ensure prefetches arrive just in time. Hot data streams [5] use off-line Sequitur analysis to prefetch temporal streams. In general, software prefetching suffers from instruction overheads, use of significant computational resources, or off-line/static analysis that cannot adapt to changes in program behavior.

7. CONCLUSION

In this paper, we propose Spatio-Temporal Memory Streaming (STeMS), a hardware prefetching technique for capturing both temporal and spatial relationships in memory accesses. STeMS is motivated by the observation that temporally and spatially predicted accesses do not entirely overlap—instead, there is opportunity to exploit both concurrently. For the first time, we show temporal repetition in spatial access patterns. STeMS leverages this property to combine temporal and spatial predictions into a single predicted miss sequence that matches the application's memory behavior more closely than the underlying prediction techniques. Compared with temporal and spatial streaming alone, and with a similar implementation cost as temporal streaming, we demonstrate that STeMS achieves equal or greater prediction coverage and application speedup across our suite of commercial workloads.

Acknowledgements

The authors would like to thank the anonymous reviewers for their feedback on this paper and members of the SimFlex team at Carnegie Mellon for contributions to our simulation infrastructure. This work was partially supported by equipment donations from Intel, a Sloan research fellowship, an ESF EurYI grant, an IBM faculty partnership award, and NSF grants CCF-0702658, CCR-0509356, CCF-0845157, CCR-0205544, IIS-0133686 and IIS-0713409.

References

[1] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and David A. Wood. DBMSs on a modern processor: Where does time go? In *The VLDB Journal*, pages 266–277, Sep. 1999.

[2] Ioana Burcea, Stephen Somogyi, Andreas Moshovos, and Babak Falsafi. Predictor virtualization. In *Proceedings of the 13th Conference on Architectural Support for Programming Languages and Operations Systems*, Mar. 2008.

[3] Jason F. Cantin, Mikko H. Lipasti, and James E. Smith. Stealth prefetching. In *Proceedings of the 12th Conference on Architectural Support for Programming Languages and Operations Systems*, Oct. 2006.

[4] Chi F. Chen, Se-Hyun Yang, Babak Falsafi, and Andreas Moshovos. Accurate and complexity-effective spatial pattern prediction. In *Proceedings of the 10th Symposium on High-Performance Computer Architecture*, Feb. 2004.

[5] Trishul M. Chilimbi and Martin Hirzel. Dynamic hot data stream prefetching for general-purpose programs. In *Proceedings of the Conference on Programming Language Design and Implementation (PLDI)*, June 2002.

[6] Yuan Chou. Low-cost epoch-based correlation prefetching for commercial applications. In *Proceedings of the 40th International Symposium on Microarchitecture*, Dec. 2007.

[7] Pedro Diaz and Marcelo Cintra. Stream chaining: Exploiting multiple levels of correlation in data prefetching. In *Proceedings of the 36th International Symposium on Computer Architecture*, June 2009.

[8] Michael Ferdman, Thomas F. Wenisch, Anastasia Ailamaki, Babak Falsafi, and Andreas Moshovos. Temporal instruction fetch streaming. In *Proceedings of the 41st International Symposium on Microarchitecture*, Nov. 2008.

[9] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7, 1997.

[10] Richard Hankins, Trung Diep, Murali Annavaram, Brian Hirano, Harald Eri, Hubert Nueckel, and John 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.

[11] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju G. Mancheril, Anastassia Ailamaki, and Babak Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *Proceedings of the 3rd Conference on Innovative Data Systems Research*, Jan. 2007.

[12] Ibrahim Hur and Calvin Lin. Memory prefetching using adaptive stream detection. In *Proceedings of the 39th International Symposium on Microarchitecture*, Dec. 2006.

[13] Doug Joseph and Dirk Grunwald. Prefetching using Markov Predictors. In *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997.

[14] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th International Symposium on Computer Architecture*, May 1990.

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

[16] Jack L. Lo, Luiz Andre Barroso, Susan J. Eggers, Kourosh Gharachorloo, Henry M. Levy, and Sujay S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Proceedings of the 25th International Symposium on Computer Architecture*, June 1998.

[17] Kyle J. Nesbit and James E. Smith. Data cache prefetching using a global history buffer. In *Proceedings of the 10th Symposium on High-Performance Computer Architecture*, Feb. 2004.

[18] Minglong Shao, Anastassia Ailamaki, and Babak Falsafi. DBmbench: Fast and accurate database workload representation on modern microarchitecture. In *Proceedings of the IBM Center for Advanced Studies Conference*, Oct. 2005.

[19] Timothy Sherwood, Suleyman Sair, and Brad Calder. Predictor-directed stream buffers. In *Proceedings of the 33rd International Symposium on Microarchitectur*e, Dec. 2000.

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

[21] Stephen Somogyi, Thomas F. Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. Spatial memory streaming. In *Proceedings of the 33rd International Symposium on Computer Architecture*, June 2006.

[22] Pedro Trancoso, Josep-L. Larriba-Pey, Zheng Zhang, and Josep Torellas. The memory performance of DSS commercial workloads in shared-memory multiprocessors. In *Proceedings of the 3rd Symposium on High-Performance Computer Architecture*, Feb. 1997.

[23] Thomas F. Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. Mechanisms for store-wait-free multiprocessors. In *Proceedings of the 34th International Symposium on Computer Architecture*, Junr 2007.

[24] Thomas F. Wenisch, Michael Ferdman, Anastasia Ailamaki, Babak Falsafi, and Andreas Moshovos. Temporal streams in commercial server applications. In *Proceedings of the International Symposium on Workload Characterization*, Sep. 2008.

[25] Thomas F. Wenisch, Michael Ferdman, Anastasia Ailamaki, Babak Falsafi, and Andreas Moshovos. Practical off-chip meta-data for address-correlated prefetching. In *Proceedings of the 15th Symposium on High-Performance Computer Architecture*, Feb. 2009.

[26] Thomas F. Wenisch, Stephen Somogyi, Nikolaos Hardavellas, Jangwoo Kim, Anastassia Ailamaki, and Babak Falsafi. Temporal streaming of shared memory. In *Proceedings of the 32nd International Symposium on Computer Architecture*, June 2005.

[27] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. SimFlex: statistical sampling of computer system simulation. *IEEE Micro*, 26(4):18–31, July-Aug. 2006.

[28] Roland Wunderlich, Thomas Wenisch, Babak Falsafi, and James Hoe. SMARTS: Accelerating microarchitecture simulation through rigorous statistical sampling. In *Proceedings of the 30th International Symposium on Computer Architecture*, June 2003.

[29] Weifeng Zhang, Brad Calder, and Dean M. Tullsen. A self-repairing prefetcher in an event-driven dynamic optimization framework. In *Proceedings of the 4th International Symposium on Code Generation and Optimization*, Mar. 2006.

[30] Weifeng Zhang, Dean M. Tullsen, and Brad Calder. Accelerating and adapting precomputation threads for effcient prefetching. In *Proceedings of the 13th Symposium on High-Performance Computer Architecture*, Feb. 2007.