MAKING ADDRESS-CORRELATED PREFETCHING PRACTICAL

DESPITE A DECADE OF RESEARCH DEMONSTRATING ITS EFFICACY, ADDRESS-CORRELATED PREFETCHING HAS NEVER BEEN IMPLEMENTED IN A SHIPPING PROCESSOR BECAUSE IT REQUIRES MEGABYTES OF METADATA—TOO LARGE TO STORE PRACTICALLY ON CHIP. NEW STORAGE-, LATENCY-, AND BANDWIDTH-EFFICIENT MECHANISMS FOR STORING METADATA OFF CHIP YIELD A PRACTICAL DESIGN THAT ACHIEVES 90 PERCENT OF THE PERFORMANCE POTENTIAL OF IDEALIZED ON-CHIP METADATA STORAGE.

Thomas F. Wenisch
University of Michigan

Michael Ferdman
Carnegie Mellon University

Anastasia Ailamaki
Babak Falsafi
École Polytechnique Fédérale de Lausanne

Andreas Moshovos
University of Toronto

Memory access latency continues to pose a crucial performance bottleneck for server workloads.[1] Prefetchers bring cache blocks from main memory to on-chip caches prior to explicit processor requests to hide cache miss latency. Prefetching improves throughput and response time by increasing memory-level parallelism[2,3] and remains an essential strategy to address the processor-memory performance gap. Today's systems primarily employ stride-based prefetchers, which require only simple hardware additions and minimal on-chip area. However, these prefetchers are only partially effective in commercial server workloads, such as online transaction processing, which are dominated by pointer-chasing access patterns.[2]

In contrast, address-correlating prefetchers[3-11] are effective for repetitive, yet arbitrarily irregular, access patterns, particularly pointer chasing. Address-correlating prefetchers associate a miss address with a sequence of possible successor misses. More than a decade of research has repeatedly shown that address-correlating prefetchers can eliminate half (or more) of all off-chip misses in pointer-intensive commercial server workloads.[3,4,12] Figure 1 illustrates the potential of address-correlated prefetching across a range of server and scientific applications running on a four-core chip multiprocessor.[12] The figure shows the fraction of off-chip misses that are *covered* (that is, successfully predicted and prefetched) by an idealized address-correlating prefetcher with unbounded zero-latency metadata storage. Nevertheless, stride prefetchers are widely implemented, whereas, to date, no commercial design has implemented an address-correlating prefetcher.

Address-correlating prefetchers have never been deployed in production designs because they require megabytes of address correlation metadata—far too large to store practically on chip. We introduce two innovations for managing metadata that make off-chip storage practical. To address the access-latency challenge, we introduce *hash-based lookup*, in which a hardware-managed hash table maintains an index of previously recorded miss-address sequences within a log of prior misses. Hash-based lookup lets a prefetecher predict a long sequence of prefetch addresses with only two main-memory round-trip latencies (one for the hashed index table lookup, and the second for the address

Published by the IEEE Computer Society

sequence). To address the off-chip memory bandwidth challenge, we introduce *probabilistic update*. This mechanism applies only a randomly selected subset of updates to the hashed index table, reducing metadata memory traffic by a factor of 3.4. Because recurring miss-address sequences tend either to be long or to repeat frequently, stale or missing index table entries don't adversely impact prefetcher effectiveness.

Using these mechanisms, we build on previously proposed address-correlating prefetchers[3,10] to develop sampled temporal memory streaming (STMS),[12] a practical prefetcher with main-memory metadata that achieves 90 percent of the performance potential of an idealized design.

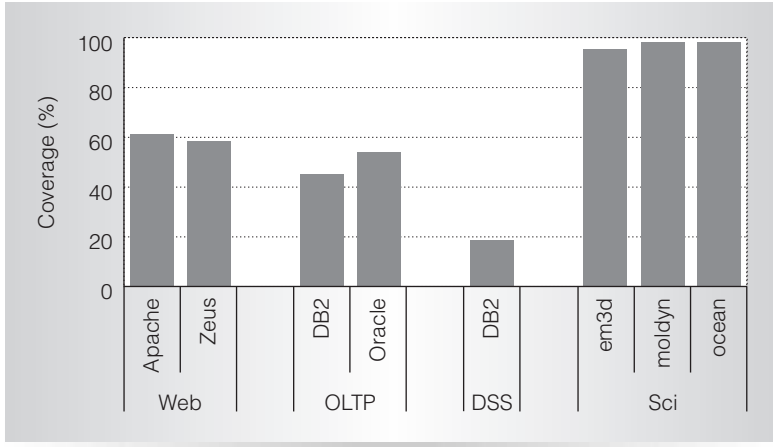

Figure 1. Address-correlated prefetching has the potential to eliminate misses in web, online transaction processing (OLTP), decision support system (DSS), and scientific applications.

Development of address-correlated prefetching

Although address-correlated prefetching can correctly predict most off-chip misses, several impediments have prevented its adoption. First, many processor designers believe that address-correlating prefetchers are impractical because they require massive metadata. These prefetchers require *correlation tables* to capture the relationship between miss addresses and their likely successors. The correlation tables typically require megabytes of storage, and their requirements grow in proportion to the application footprint.

Early address-correlating prefetcher designs[7-9] proposed storing these massive correlation tables on chip—a cost-prohibitive proposition. Later designs shifted correlation metadata to main memory.[3,4,12] However, this development led to new practicality challenges. First, correlation table lookups incur main-memory access latencies, which delay prefetches. Second, accessing and maintaining off-chip correlation metadata can consume substantial off-chip memory bandwidth—an increasingly precious resource, as scaling in cores outstrips improvements in sustainable memory bandwidth. So, doubts persisted about whether address-correlating prefetchers were practical.

Current designs

An address-correlating prefetcher learns temporal relationships between accesses to specific addresses. For instance, if address *B* tends to be accessed shortly after address *A*, an address-correlating prefetcher can learn this relationship and use the occurrence of *A* to trigger a prefetch of *B*. Address-correlating prefetchers succinctly capture pointer-chasing relationships, and thus substantially improve the performance of pointer-intensive commercial workloads.[3]

Pairwise-correlating prefetchers. The Markov prefetcher[7] is the simplest prefetcher design for predicting pairwise correlation between an address and its successor (that is, two addresses that tend to cause consecutive cache misses). The Markov prefetcher hardware comprises a set-associative table that maps an address to several recently observed possibilities for the succeeding miss. On each miss, the prefetcher searches the table for the miss address, and if it finds an entry, it prefetches the likely successors. Several pairwise-correlating prefetchers build on this simple design to optimize correlation table storage[8] or trigger prefetchers earlier to improve lookahead.[9]

Pairwise-correlating prefetchers' key limitation is that they attempt to correctly predict only a single miss per prediction, limiting memory-level parallelism and prefetch lookahead. More recent address-correlating prefetchers use a single correlation to predict a sequence of successor misses.[3-6,10,11]

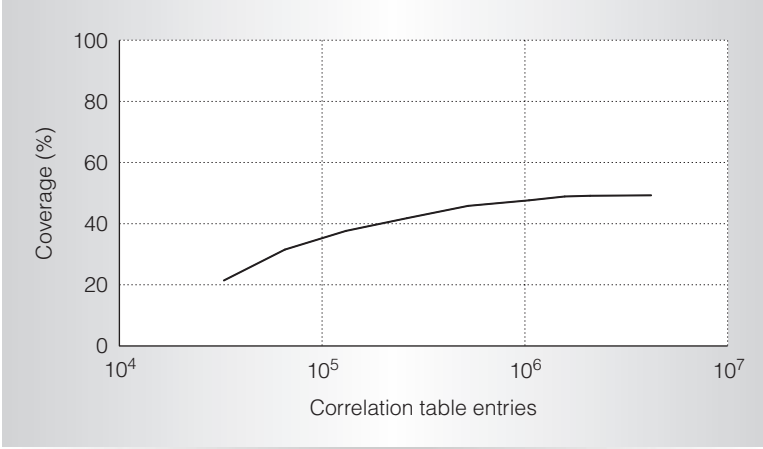

Figure 2. Correlation table entries required to achieve a given prefetch effectiveness. To maximize coverage, correlation tables must store more than 1 million entries, which can require 32 Mbytes of storage.

We call these successor sequences *temporal streams*[3]—extended sequences of memory accesses that recur during program execution (see the "Why does temporal memory streaming work?" sidebar for more information).

Temporal streaming. The observation that extended memory access sequences recur was first quantified in memory trace analysis by Chilimbi and Hirzel.[4] To exploit this observation, researchers initially proposed correlation tables that store a temporal stream (that is, a short sequence of successors) rather than only a single future access in each set-associative correlation table entry.[2,11] This set-associative organization's primary shortcoming is that temporal stream length is fixed to the size of the table entry, typically three to six successor addresses. These table entries are far shorter than typical temporal streams: Offline analyses of miss repetition[3,4] have shown that temporal streams vary drastically in length, from two to hundreds of misses. Fixing stream length in the prefetcher design either leads to inefficient use of correlation table storage (if the entries are too large) or sacrifices lookahead and prefetch coverage (if entries are too small).

To support variable-length temporal streams while maintaining storage efficiency, several designs separate address sequence storage and correlation data.[3,6,10] A *history buffer* records the application's recent miss-address sequence, typically in a circular buffer. An *index table* correlates a particular miss address (or other lookup criteria) to a location in the history buffer. The split-table approach lets a single index-table entry point to a stream of arbitrary length, allowing maximal lookahead and prefetch coverage without substantial storage overhead.

Practicality challenges

Although address correlation research has made impressive strides in designing mechanisms to discover and exploit repetition in memory-access sequences, practicality challenges have persistently precluded mainstream adoption.

On-chip storage requirements. Initial address-correlating prefetcher designs located the correlation tables entirely on chip.[7-10] However, correlation table storage requirements are proportional to the application's footprint.[3] So, for commercial workloads, even the most storage-efficient design[8] requires megabytes of correlation table storage to be effective.[2,3] Figure 2 shows the number of correlation table entries an idealized address-correlating prefetcher requires to eliminate a given fraction of off-chip misses (on average) across a suite of commercial workloads. To achieve maximum coverage, correlation tables must store more than 1 million entries, which can require 32 Mbytes of storage. High storage requirements make on-chip correlation tables impractical.

Recent prefetchers locate correlation metadata in main memory,[2,3,5,11] which can easily accommodate multimegabyte tables. However, off-chip tables lead to two new challenges: high lookup latency and increased memory bandwidth pressure.

High lookup latency. When correlation tables are off chip, each lookup requires at least one main-memory access before prefetching can proceed. So, the prefetching mechanism must be designed to account for long correlation table lookup latency. Epoch-based correlation prefetching (EBCP)[2] and last-touch correlated data streaming

Why does temporal memory streaming work?

To illustrate why memory accesses in commercial applications often occur in temporal streams, we present two motivating examples taken from actual behaviors we've observed in our commercial application suite.

Example one: B+-tree range scans

The B+-tree, a critical data structure in database applications, enables the efficient insertion, removal, and search of database records. A B+-tree maintains a sorted index of records according to a key. Each B+-tree node contains a sorted key list with pointers to children, such that the range of keys within a child's subtree is bounded by two adjacent keys in the parent. A distinguishing feature of the B+-tree is the horizontal pointers that connect the tree's sibling leaves. These horizontal links enable fast in-order tree traversals and are used to implement range scans. In a range scan, the database engine first locates the lower key. It then traverses horizontally along sibling links until it reaches the upper key. Scans for overlapping ranges result in temporal streams following the sibling links. The first range scan records a miss sequence for leaves along the bottom of the tree. The second will access the same leaves in the same order. Because leaves typically aren't contiguous in memory, stride prefetchers can't capture the leaf access sequence.

Example two: Solaris thread scheduler

A key innovation of Solaris 2.3—introduced nearly 15 years ago, but still used in current Solaris releases—is per-processor dispatch queues. In the earliest versions of Solaris, a single queue maintained pointers to all runnable threads waiting to be scheduled on a processor. To improve multiprocessor scalability, this single dispatch queue was split into per-processor dispatch queues, each protected by separate locks, allowing the queues to be modified concurrently. Typically, when the thread currently running on a processor blocks or exhausts its time quantum, the processor simply scans its own dispatch queue to identify which thread to run next. However, if its dispatch queue is empty, the processor tries to steal a runnable thread off another processor's queue. It scans other queues looking for the highest priority thread available, removes the thread from the queue, and finally confirms that no higher priority thread has since become runnable. These operations account for an astounding amount of coherence misses to the locks that protect each dispatch queue and the linked lists that comprise the queues. These miss sequences are highly repetitive because all processors scan the dispatch queues in the same order, advancing through the linked list that connects the dispatch queues. Because the locks remain at fixed addresses, and the queues themselves change little between scans, the misses form temporal streams.

Figure A shows the breakdown of miss sources and their contribution to temporal streams from an offline analysis of a four-core processor running an online transaction processing workload on IBM DB2.[1] Traversals of indexes, buffer-pool pages, and tuples are the largest contributor to temporal streams. Our prior work[1] further details this miss breakdown and reports similar breakdowns for Web serving and decision support workloads.

Reference

1. T.F. Wenisch et al., ''Temporal Streams in Commercial Server Applications,'' *IEEE Int'l Symp. Workload Characterization* (IISWC 2008), IEEE CS Press, 2008, pp. 99-108.

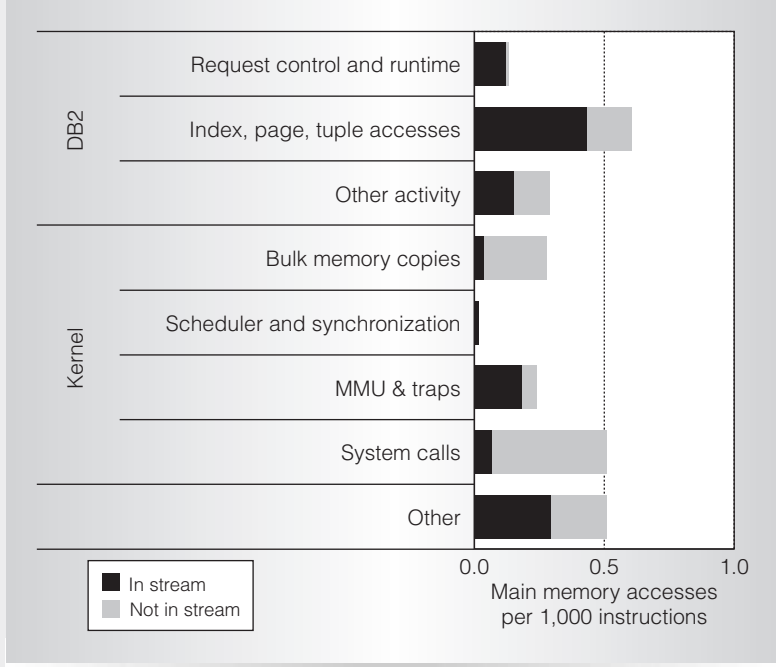

Figure A. Temporal stream origins in an online transaction processing workloads.[1]

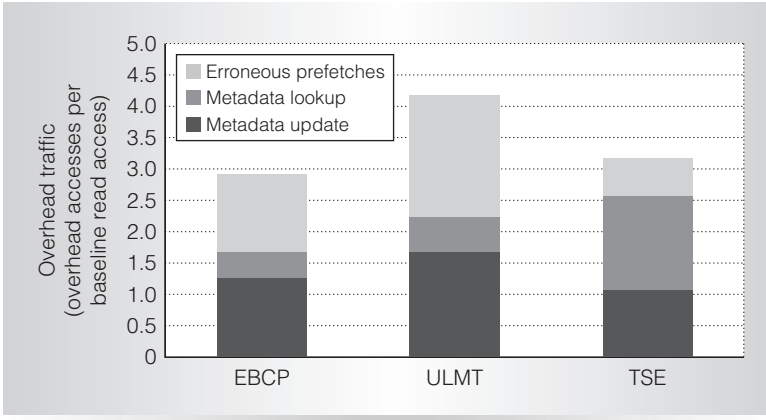

Figure 3. Memory traffic overheads in three off-chip metadata designs: Epoch-based correlation prefetching (EBCP), user-level memory thread (ULMT), and temporal streaming engine (TSE). Overhead traffic includes erroneous prefetches, metadata lookup, and metadata updates.

(LT-Cords)[5] explicitly account for off-chip lookup latency in choosing prefetch addresses. Rather than correlate an address to its immediate successors, these designs skip over successor addresses that will be requested while the correlation table lookup is in progress. However, LT-Cords still requires an on-chip index table, which remains cost-prohibitive for commercial workloads. EBCP employs a single set-associative correlation table, which bounds maximum stream length, limiting memory-level parallelism, lookahead, and bandwidth efficiency relative to designs that separate history and index tables.

Memory bandwidth requirements. Placing metadata off chip substantially increases pressure on memory bandwidth. First, as with any prefetching mechanism, erroneous prefetches consume memory bandwidth, as prefetchers inherently trade increased memory bandwidth requirements to reduce effective access latency. However, off-chip correlation tables make matters worse: both metadata lookups and updates require off-chip memory accesses.

Figure 3 shows the average memory traffic overhead for three existing address-correlating prefetchers that store metadata in main memory: the user-level memory thread (ULMT),[11] EBCP,[2] and the temporal streaming engine (TSE).[3] Overhead traffic is normalized to the number of memory reads without a prefetcher. "Erroneous prefetches" indicates cache blocks that are prefetched from memory, but never accessed. "Metadata lookup" and "metadata update" refer to memory traffic incurred while accessing metadata to find addresses for prefetch and while recording new correlations, respectively.

As the figure shows, overhead traffic is triple the baseline read traffic without a prefetcher. Several factors mitigate the massive traffic overheads' performance impact in the existing designs. All three designs issue correlation table lookups and updates as low-priority traffic, prioritizing processor-initiated requests. ULMT colocates its prefetcher with an off-chip memory controller, so its metadata traffic doesn't cross the processor's pins. TSE's metadata lookups are embedded in existing cache coherence traffic. When unused memory bandwidth is abundant, the memory system can absorb overhead traffic with minimal performance impact. However, available memory bandwidth must be shared among cores in a multicore system, so bandwidth demands are growing rapidly with multicore scaling, whereas improvements in sustainable memory bandwidth are more gradual. So, current main-memory correlation table designs have a high traffic overhead that limits their applicability.

STMS introduces two new mechanisms—hash-based lookup and probabilistic index table update—to enable off-chip metadata storage while addressing latency and memory-bandwidth challenges, to achieve a practical address-correlating prefetcher design.

A practical design: STMS

Figure 4 shows a block diagram of a four-core processor equipped with STMS. STMS comprises on-chip prefetch buffers and queues and off-chip index tables and history buffers.[10] On chip, STMS requires a prefetch buffer and address queue colocated with each core to track pending prefetch addresses and buffer prefetched data. The prefetch buffers and queues orchestrate the streaming process and act as a temporary holding space for a small number of blocks that have been prefetched, but which the core hasn't yet accessed. The off-chip structures are allocated

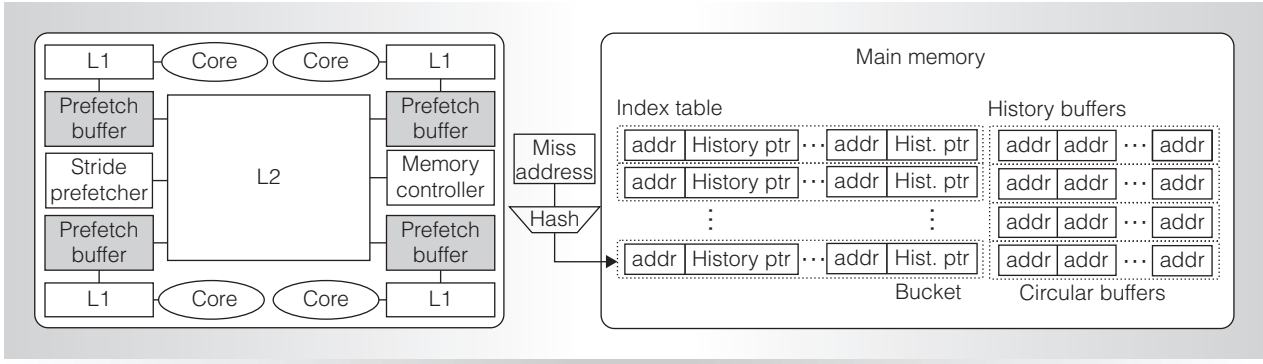

Figure 4. STMS block diagram. STMS adds prefetch buffers on chip and an index table and history buffers in main memory.

in a private region of main memory. Each core receives its own history buffer, but all cores share a unified index table, which contains a mapping from physical addresses to pointers within the history buffers, facilitating lookup.

Operation

STMS operation comprises three steps: recording, locating, and prefetching temporal streams.

Recording temporal streams. STMS records correct-path off-chip misses and prefetched hits in the corresponding core's history buffer. To avoid polluting the history buffer with wrong-path accesses, it marks instructions observing prefetched hits and off-chip loads in the load-store queue. Later, when a marked instruction retires, the effective physical address is appended to the history buffer. To minimize pin-bandwidth overhead, a cache-block-sized buffer accumulates entries, which are then written to main memory as a group. As history buffer entries are created, STMS can update the index table entry for the corresponding address to point to the new history buffer entry.

Lookup. Upon an off-chip read miss, STMS searches for a previously recorded temporal stream that begins with the miss address. STMS performs a pointer lookup in the index table. If it finds a pointer, the address sequence is read from the history buffer, starting at the pointer location.

Streaming cache blocks. A FIFO address queue holds miss addresses read from the history buffer. STMS prefetches addresses from the queue in order. The small, fully associative prefetch buffer stores prefetched data, avoiding cache pollution on erroneous prefetches. On correct prefetches, processor requests are satisfied directly from the prefetch buffer while STMS continues to populate the address queue with addresses from the off-chip history buffer.

Storage efficiency

STMS achieves practical storage overheads by locating correlation tables off chip.

History buffer. The prefetcher's metadata reuse distance drives the history buffer's main-memory storage requirements. The history buffer must be at least large enough to fit all intervening miss addresses between a recorded temporal stream and its previous occurrence. For commercial workloads, the reuse distance depends on the frequency at which data structures are revisited. We find that we need history buffer storage on the order of 32 Mbytes to achieve maximal coverage—at least an order of magnitude greater than can be practically allocated on chip. However, relative to a server's main memory, the history buffer footprint is small.

Index table. An ideal index table can locate the most recent occurrence of any miss address in the history buffers. So, in the worst case, if all addresses in the history

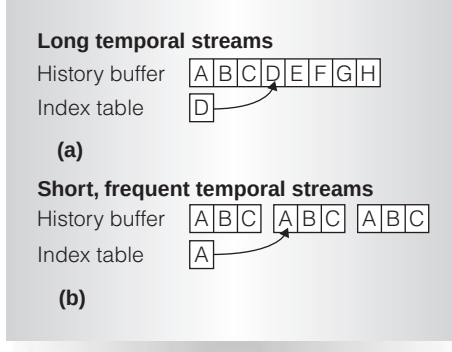

Figure 5. Cases in which probabilistic update is effective. In long temporal streams (a), an index table entry will be created near the first miss in a temporal stream with high probability. Coverage lost on the first few blocks is negligible relative to stream length. In short, frequent temporal streams (b), after several stream occurrences, the probability of adding A to the index table is high. Because temporal streams are stable, old occurrences are still valid.

buffers are distinct, then one index table entry exists per history buffer entry. In practice, we needed far fewer index table entries. Hash-based lookup spreads miss addresses over buckets, applying a least-recently used (LRU) replacement policy in each bucket. The LRU policy brings useful history buffer pointers to the top, naturally aging out unneeded entries. Hash-based lookup achieves maximum coverage with a 16-Mbyte main-memory index table, retaining only a fraction of the index entries of an idealized prefetcher.

Latency efficiency

Temporal streaming prefetchers initiate lookup upon a cache miss. However, STMS can't issue prefetches until it locates an address sequence in the off-chip index table. Cache misses incurred during the lookup process result in lost prefetch opportunity, even if they comprise a predictable temporal stream.[2,5] Effective streaming requires minimal lookup latency to ensure timely prefetching. To achieve low lookup latency, we implement the index table as a bucketized hash table. The hash table maps a trigger miss's physical address to a corresponding temporal stream located in the history buffer. Furthermore, because STMS employs a split-table metadata organization, with a separate history buffer and index table, it can follow temporal streams for tens or hundreds of misses, amortizing the overhead and lost opportunity of the lookup operation over many successful prefetches.

Bandwidth efficiency

Maintaining predictor metadata off chip induces substantial overhead traffic across the memory interface. Spatial locality allows amortization of history buffer updates by storing multiple consecutive addresses with one off-chip write. Under prior temporal streaming designs,[3,6] each miss recorded in the history buffer also incurs an index table update, creating or updating the entry corresponding to the miss address to point to its new history buffer location. Because they're hashed, metadata updates are directed to arbitrary addresses and exhibit neither spatial nor temporal locality. Naïvely adding STMS to a baseline system without addressing index table updates can triple memory traffic.

To reduce index-table-update traffic, we introduced probabilistic update. For every potential index table update, a coin flip, biased to a predetermined sampling probability, determines whether the update will be performed. Index-table-update bandwidth is directly proportional to the sampling probability. For example, a 50 percent sampling probability halves bandwidth requirements. Probabilistic update is highly effective in reducing index-table-update bandwidth, while leading to only a small coverage loss (less than 6 percent with 12.5 percent update probability).

Figure 5 illustrates why probabilistic update is effective. For long temporal streams, probabilistic update likely skips several addresses before creating an index table entry pointing into the body of the stream. However, coverage loss on the first few blocks is negligible relative to the stream's length. For short, frequent temporal streams, the probability of inserting the first address into the index table grows with the number of stream recurrences; a high appearance

frequency results in an index update within a small number of occurrences.

Figure 6 demonstrates probabilistic update's effectiveness. Placing predictor metadata off chip introduces several sources of pin-bandwidth overheads: recording miss addresses in the history buffer, index table updates, lookup operations, and prefetching of erroneously predicted addresses. In an unoptimized system (see the bars labeled 100 percent), the largest bandwidth overhead arises due to index table maintenance. These high traffic overheads demonstrate the need for probabilistic update: with a sampling probability of one-eighth (12.5 percent), we can drastically reduce overhead traffic.

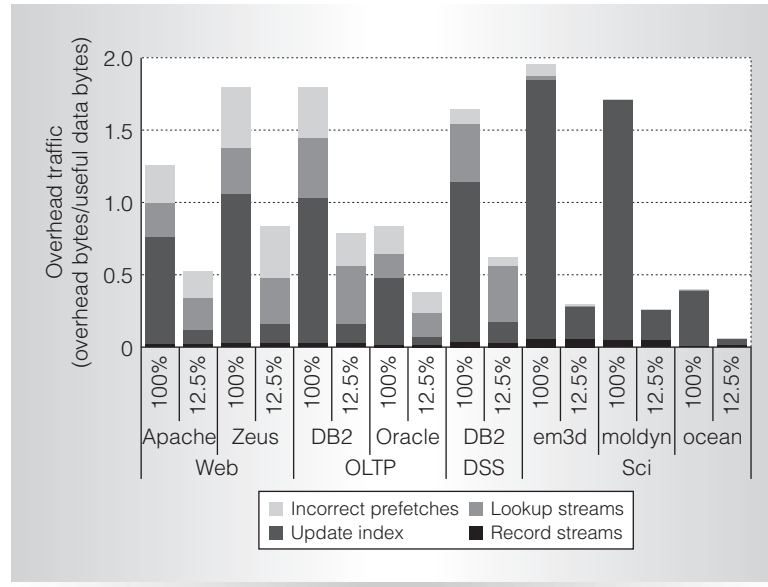

Figure 6. Overhead traffic. The left bar shows memory traffic overheads without probabilistic update (100 percent), while the right bar shows a 12.5 percent update probability.

Impact

Because STMS maintains high coverage, it achieves 90 percent of the performance improvement possible with idealized lookup. Figure 7 compares STMS's performance improvement and a temporal memory streaming design with idealized on-chip lookup (zero-latency on-chip storage). Our proposed mechanisms—hash-based lookup and probabilistic update—enable a bandwidth- and latency-efficient design, matching the effectiveness of an idealized prefetcher while storing metadata in off-chip memory. STMS, for the first time, makes the decade-old promise to accelerate pointer-chasing applications with address-correlated prefetching a practical possibility.

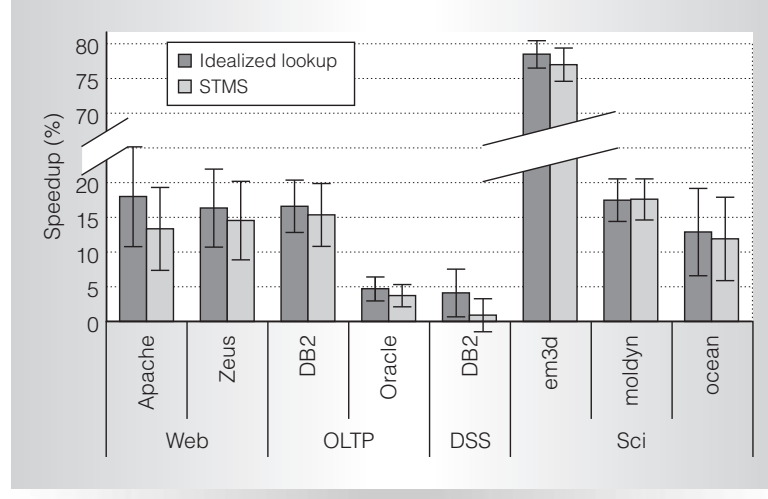

Figure 7. Speed-up of idealized temporal streaming and practical STMS relative to a design with stride prefetching only.

Beyond their specific application to prefetching, the mechanisms we propose (hash-based lookup and probabilistic update) are applicable to a wide range of prediction techniques whose metadata footprints exceed on-chip storage capacity. STMS outlines the general requirements for storing metadata off chip and provides ready-to-use solutions for the bandwidth and latency implementation challenges that would be faced by any prediction mechanism with off-chip metadata. Our work can facilitate research into metadata-intensive prediction techniques (such as address-correlated speculative synchronization, thread scheduling, or power management) that have, until now, had no practical implementation. MICRO

Acknowledgments

We thank the anonymous reviewers for their feedback. This work was supported by software donations from Virtutech, grants from Intel, two Sloan research fellowships, a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant, an IBM faculty partnership award,

and US National Science Foundation grant CCR-0509356.

References

1. N. Hardavellas et al., ''Database Servers on Chip Multiprocessors: Limitations and Opportunities,'' *Proc. 3rd Biennial Conf. Innovative Data Systems Research,* 2007, www.cidrdb.org/cidr2007/papers/cidr07p08.pdf.
2. Y. Chou, ''Low-Cost Epoch-Based Correlation Prefetching for Commercial Applications,'' *Proc. 40th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO 07), IEEE CS Press, 2007, pp. 301-313.
3. T.F. Wenisch et al., ''Temporal Streaming of Shared Memory,'' *Proc. 32nd Ann. Int'l Symp. Computer Architecture,* IEEE CS Press, 2005, pp. 222-233.
4. T.M. Chilimbi and M. Hirzel, ''Dynamic Hot Data Stream Prefetching for General-Purpose Programs,'' *Proc. ACM SIGPLAN 2002 Conf. Programming Language Design and Implementation,* ACM Press, 2002, pp. 199-209.
5. M. Ferdman and B. Falsafi, ''Last-Touch Correlated Data Streaming,'' *IEEE Int'l Symp. Performance Analysis of Systems and Software* (ISPASS 07), IEEE CS Press, 2007, pp. 105-115.
6. M. Ferdman et al., ''Temporal Instruction Fetch Streaming,'' *Proc. 41st Ann. IEEE/ACM Int'l Symp. Microarchitecture* (MICRO 08), IEEE CS Press, 2008, pp. 1-10.
7. D. Joseph and D. Grunwald, ''Prefetching Using Markov Predictors,'' *Proc. 24th Ann. Int'l Symp. Computer Architecture,* ACM Press, 1997, pp. 252-263.
8. Z. Hu, M. Martonosi, and S. Kaxiras, ''TCP: Tag Correlating Prefetchers,'' *Proc. 9th IEEE Symp. High-Performance Computer Architecture* (HPCA 9), IEEE CS Press, 2003, pp. 317-326.
9. A.-C. Lai, C. Fide, and B. Falsafi, ''Dead-Block Prediction and Dead-Block Correlating Prefetchers,'' *Proc. 28th Ann. Int'l Symp. Computer Architecture,* ACM Press, 2001, pp. 144-154.
10. K.J. Nesbit and J.E. Smith, ''Data Cache Prefetching Using a Global History Buffer,'' *Proc. 10th Int'l Symp. High-Performance Computer Architecture,* IEEE CS Press, 2004, p. 96-105.
11. Y. Solihin, J. Lee, and J. Torrellas, ''Using a User-Level Memory Thread for Correlation Prefetching,'' *Proc. 29th Ann. Int'l Symp. Computer Architecture,* IEEE CS Press, 2002, pp. 171-182.
12. T.F. Wenisch et al., ''Practical Off-chip Metadata for Temporal Memory Streaming,'' *Proc. IEEE 15th Int'l Symp. High Performance Computer Architecture,* IEEE CS Press, 2009, pp. 79-90.

Thomas F. Wenisch is an assistant professor of electrical engineering and computer science at the University of Michigan. His research interests include computer architecture, server and data center energy efficiency, multiprocessor systems, and performance evaluation methodology. Wenisch has a PhD in electrical and computer engineering from Carnegie Mellon University. He is a member of the IEEE and ACM.

Michael Ferdman is a PhD candidate in electrical and computer engineering at Carnegie Mellon University, currently studying at École Polytechnique Fédérale de Lausanne. His research interests include computer architecture with an emphasis on proactive memory system design. Ferdman has an MS in electrical and computer engineering from Carnegie Mellon University. He is a student member of the IEEE and ACM.

Anastasia Ailamaki is a professor of computer science at École Polytechnique Fédérale de Lausanne. Her research interests include the broad area of database systems and applications, with emphasis on database system behavior on modern processor hardware and disks. Ailamaki has a PhD in computer science from the University of Wisconsin-Madison. She is a member of the IEEE and ACM.

Babak Falsafi is a professor of computer science at École Polytechnique Fédérale de Lausanne, where he directs the Parallel Systems Architecture Lab. His research interests include architectural support for parallel programming, resilient systems, architectures to break the memory wall, and computer system performance evaluation. Falsafi has a PhD in computer science from

the University of Wisconsin. He is a senior member of the IEEE and the ACM.

Andreas Moshovos is an associate professor of electrical and computer engineering at the University of Toronto. His research interests include microarchitectural optimizations for high-performance processors and systems. Moshovos has a PhD in computer science from the University of Wisconsin-Madison. He is a member of IEEE and the ACM.

Direct questions and comments about this article to Thomas Wenisch, Dept. of Electrical Engineering & Computer Science, Univ. of Michigan, 2260 Hayward St., 4620 CSE, Ann Arbor, MI 48109; twenisch@umich.edu.