Efficient Metadata Management for Irregular Data Prefetching

Hao Wu
Arm Research and
The University of Texas at Austin
Austin, Texas
haowu@cs.utexas.edu

Krishnendra Nathella
Arm Research
Austin, Texas
Krishnendra.Nathella@arm.com

Dam Sunwoo
Arm Research
Austin, Texas
Dam.Sunwoo@arm.com

Akanksha Jain
The University of Texas at Austin
Austin, Texas
akanksha@cs.utexas.edu

Calvin Lin
The University of Texas at Austin
Austin, Texas
lin@cs.utexas.edu

ABSTRACT

Temporal prefetchers have the potential to prefetch arbitrary memory access patterns, but they require large amounts of metadata that must typically be stored in DRAM. In 2013, the Irregular Stream Buffer (ISB), showed how this metadata could be cached on chip and managed implicitly by synchronizing its contents with that of the TLB. This paper reveals the inefficiency of that approach and presents a new metadata management scheme that uses a simple metadata prefetcher to feed the metadata cache. The result is the Managed ISB (MISB), a temporal prefetcher that significantly advances the state-of-the-art in terms of both traffic overhead and IPC.

Using a highly accurate proprietary simulator for single-core workloads, and using the ChampSim simulator for multi-core workloads, we evaluate MISB on programs from the SPEC CPU 2006 and CloudSuite benchmarks suites. Our results show that for single-core workloads, MISB improves performance by 22.7%, compared to 10.6% for an idealized STMS and 4.5% for a realistic ISB. MISB also significantly reduces off-chip traffic; for SPEC, MISB's traffic overhead of 70% is roughly one fifth of STMS's (342%) and one sixth of ISB's (411%). On 4-core multi-programmed workloads, MISB improves performance by 19.9%, compared to 7.5% for idealized STMS. For CloudSuite, MISB improves performance by 7.2% (vs. 4.0% for idealized STMS), while achieving a traffic reduction of 11× (96.2% for MISB vs. 1082.7% for STMS).

KEYWORDS

Data prefetching, irregular temporal prefetching, CPUs

ACM Reference Format:
Hao Wu, Krishnendra Nathella, Dam Sunwoo, Akanksha Jain, and Calvin Lin. 2019. Efficient Metadata Management for Irregular Data Prefetching. In *ISCA '19: ACM International Symposium on Computer Architecture, June 22–26, 2019, Phoenix, Arizona, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/1122445.1122456

ISCA '19, June 22–26, 2019, Phoenix, Arizona, USA

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Data prefetchers are critical components of modern memory systems, and today's microprocessors typically contain multiple prefetchers, primarily for regular or strided memory accesses. In terms of commercial impact, the greatest potential for growth resides in the area of temporal prefetchers. Because these prefetchers learn pairs of correlated addresses, they can learn arbitrary memory access patterns, including those from highly irregular memory access streams. Unfortunately, the need to memorize pairs of correlated addresses leads to enormous amounts of metadata that must be stored off chip, so a critical issue is the management of metadata to minimize traffic overhead and access latency [1–3].

For GHB-based prefetchers [4], Wenisch, et al. [2] address this problem by probabilistically updating metadata and by amortizing off-chip lookup over many prefetch requests. Unfortunately, these prefetchers still access significant amounts of off-chip metadata (see Section 5.2). Moreover, while caches are a common method of reducing both latency and traffic, these prefetchers do not lend themselves to metadata caching because of the poor temporal locality of GHB metadata [2, 5].

Jain and Lin's Irregular Stream Buffer (ISB) introduces a metadata representation that allows ISB to cache portions of its metadata on chip.[1] ISB's metadata is cacheable because the metadata maps each physical memory address to a new *structural address*, such that two temporally correlated physical addresses are assigned consecutive structural addresses (see Figure 1). Of course, address mappings can be easily cached, just as they are cached in a TLB.

ISB's metadata cache is managed implicitly by the TLB: When a TLB entry is brought in from DRAM, its corresponding ISB metadata is also brought in, and when a TLB entry is evicted, the corresponding metadata is written to DRAM. Thus, in theory, the latency of metadata access is hidden by the high cost of a TLB eviction.

Unfortunately, ISB's metadata management scheme suffers from three deficiencies:

- It fetches large amounts of useless metadata (90% of its loaded metadata is never used), which leads to poor metadata cache efficiency (30% hit rate) and high metadata traffic overhead (411% traffic overhead for irregular SPEC 2006 workloads).

[1] As discussed in Section 3, this new metadata representation also enables ISB to use a technique known as *PC localization* [3], which significantly improves both coverage and accuracy.

- It does not scale to large page sizes. Because the size of the on-chip cache grows with the page size, ISB is infeasible for huge pages (2MB to 1GB in size), which are orders of magnitude larger than the standard 4K page, and which are important for many programs that have large memory footprints.
- It does not work for modern two-level TLBs. If the metadata cache is synchronized with the L1 TLB, the latency of L2 TLB hits is too short to hide the latency of off-chip metadata requests. If the metadata cache is synchronized with the L2 TLB, the metadata cache would be impractically large—on the order of 200-400KB.

In this paper, we introduce a new metadata management scheme for ISB that addresses all three deficiencies. Instead of piggybacking off of the TLB, our new scheme uses a simple metadata prefetcher to load the metadata cache, and it uses an LRU replacement policy to evict lines from the metadata cache.

There are three key insights behind our metadata management scheme. First, TLB-based cache management is wasteful because metadata for a physical page, which includes 64 consecutive physical-to-structural mappings, typically exhibits poor spatial locality, yielding metadata cache utilization of about 10%. Thus, metadata should be cached at a finer granularity. Second, because structural addresses are temporally ordered, the structural address space has precisely the information that is needed to fill the metadata cache with useful entries ahead of time. Thus, we prefetch metadata entries by using next-line prefetching for structural-to-physical mappings. Third, many off-chip metadata requests are to addresses that have not been seen before, so no metadata exists for these addresses. Thus, we use a Bloom filter to record the physical addresses that have associated metadata; by checking the Bloom filter before issuing metadata prefetches, we dramatically reduce the number of metadata requests for unmapped physical addresses.

This paper makes the following contributions:

- We introduce the Managed ISB (MISB), which represents the next step in the evolution of prefetcher metadata management, as it includes a metadata cache that is filled by a simple metadata prefetcher.
  - Contrary to previous reports [1, 2, 5], we show that metadata for temporal prefetchers *can* be effectively cached. For a 32KB on-chip budget, MISB's fine-grained metadata cache achieves hit rates of 66%, twice that of ISB's metadata cache.
  - We find that for temporal prefetchers, metadata caching and metadata prefetching work synergistically to provide high coverage and low overhead.
- We show that our new metadata management scheme allows MISB to be commercially viable.
  - MISB's off-chip traffic overhead is reduced to 70%, compared with ISB's overhead of 411%, a reduction of 5.9×.
  - MISB's metadata management scheme scales to huge pages and 2-level TLBs. For example, on a system with 2MB pages, where ISB gets no performance benefits, MISB sees a 25.5% speedup.
- We evaluate MISB using a highly accurate proprietary simulator for single-core simulations and using the ChampSim simulator for multi-core simulations. We show that MISB significantly advances the state-of-the-art for a variety of workloads.
  - For irregular SPEC2006 benchmarks running on a single core, MISB improves IPC by 22.7% over a baseline that performs no prefetching, compared to 4.5% for ISB, 10.6% for an idealized STMS[2] and 6.3% for the Best Offset (BO) prefetcher [6].
  - For irregular SPEC2006 benchmarks running on a 4-core multi-programmed system, MISB improves IPC by 19.9%, whereas idealized STMS and Domino improve IPC by 8.8% and 9.6% respectively.
  - For CloudSuite server workloads running on multiple cores, MISB sees IPC improvements of 7.2%, compared with 4.0% for idealized STMS and 3.9% for idealized Domino. Thus, contrary to prior reports [5], we show that PC localization *is* beneficial for server workloads.
  - We find that MISB works well as a hybrid along with BO and SMS, since its benefits are largely orthogonal to those of BO and SMS.

This paper follows a standard organization. We first describe related work (Section 2) and background material (Section 3), before presenting our solution (Section 4), our empirical evaluation (Section 5), and conclusions (Section 6).

2 RELATED WORK

We now summarize related work in irregular prefetching and then compare our metadata management scheme with those of other temporal prefetchers.

2.1 Irregular Prefetchers

Irregular memory access patterns, such as those produced by pointer-based code and indirect array accesses, are difficult to prefetch because they do not follow simple stride-like patterns. One class of solutions employs the compiler to insert prefetch instructions [7, 8] for linked data structures, but these solutions suffer from poor timeliness. Hardware solutions that detect and prefetch pointers [9–12] are timely but suffer from poor accuracy. Other hardware prefetchers build specialized solutions for a limited class of irregular accesses. For example, the Spatial Memory Stream (SMS) prefetcher [13] targets irregular spatial footprints that repeat across memory regions, and the Indirect Memory Prefetcher (IMP) targets indirect array accesses [14].

Temporal prefetchers—the subject of this paper—memorize pairs of memory addresses that are correlated with each other. Joseph and Grunwald first exploited temporal correlation by using a Markov table [15] to record a list of possible successors for each memory reference. In 2001, Chilimbi et al., observed that correlated addresses can form long sequences, whose length can vary from two to several hundred [16, 17]. To exploit variable length *temporal streams*, Wenisch et al.'s STMS prefetcher [2] replaces the Markov table with the GHB, which records past memory accesses in a large FIFO buffer that is accessed using an Index Table, which points to the last occurrence of a given memory address. Domino improves the

[2]Our idealized STMS prefetcher incurs no latency and consumes no bandwidth for metadata accesses; a similarly idealized ISB outperforms STMS.

predictability of STMS by indexing the GHB with the past two addresses instead of one [5]. The Irregular Stream Buffer [3] replaces the GHB with a new organization that enables it to find correlated address pairs within PC-localized streams instead of the global miss stream, providing better coverage and accuracy. Finally, Spatial Temporal Memory Streaming (STeMS) [18] integrates temporal prefetchers with spatial prefetchers (SMS) to prefetch both spatial and temporal access patterns; the ideas in STeMS and MISB are largely orthogonal.

2.2 Metadata Management

Regardless of their organization, all temporal prefetches have megabytes of metadata that must be maintained off-chip. Wenisch et al., identify three key goals in managing this metadata [2]: (1) low lookup latency, (2) bandwidth-efficient lookups, and (3) bandwidth-efficient updates to off-chip metadata. Many solutions have been proposed to hide the latency of off-chip metadata accesses, but with a few notable exceptions [2], surprisingly little has been done to address the large traffic overhead of accessing off-chip metadata.

Traditional table-based methods, such as Markov tables [15], suffer from long lookup latency and high metadata read/write traffic because they must access off-chip metadata for each individual prefetch request. To hide latency, some tables store in each entry the entire stream instead of a single neighbor [19, 20], but these schemes are complicated by the high variance of temporal stream lengths [16, 21].

GHB-based methods [2, 5], including STMS and Domino, hide latency for variable length temporal streams by leveraging the temporal nature of the *history buffer*. In particular, STMS [2] amortizes the cost of accessing off-chip metadata by bringing in long prefetch sequences on every lookup. It further reduces metadata traffic by probabilistically dropping metadata writes at the cost of lower prefetch coverage. In spite of these optimizations, GHB-based solutions incur significant metadata traffic (see Section 5).

Finally, the Irregular Stream Buffer (ISB) [3] hides metadata lookup latency by caching a portion of the metadata in a small on-chip cache, which is synchronized with the contents of the TLB. Unfortunately, ISB also suffers from high traffic because most of the metadata that it fetches on a TLB miss is not utilized while it resides in the metadata cache. Furthermore, ISB's TLB synchronization scheme does not scale to 2-level TLBs and large pages, as it would require an infeasible amount of on-chip storage (200-400 KB for 2-level TLBs and several MBs for large pages) to store metadata for all TLB-resident data.

MISB differs from prior metadata management schemes in three ways. First, it dramatically reduces traffic by improving the utilization of ISB's metadata cache and by decoupling metadata cache management from the TLB. Significantly, the GHB is difficult to cache because the history buffer is a large FIFO buffer whose entries are not reused for long periods of time [2, 5]. Thus, no prior temporal prefetcher, including ISB, has been able to use metadata caching as an effective tool for reducing metadata traffic. Second, MISB uses metadata prefetching to hide the latency of off-chip metadata accesses. STMS [2] also fetches its metadata ahead of time by packing multiple entries within a cache lines, but MISB extends the notion of metadata prefetching to multiple cache lines. Finally, MISB is the first to filter useless metadata traffic using a Bloom filter.

Finally, Burcea et al.'s Predictor Virtualization [22] includes an on-chip metadata cache for the spatial SMS prefetcher [13]. MISB instead addresses metadata management for temporal prefetchers, which is a harder problem, because temporal prefetchers have orders of magnitude more metadata than SMS. Moreover, we find that for temporal prefetchers, metadata caching alone is insufficient, as accurate metadata prefetching is needed to hide latency.

3 BACKGROUND

We now describe in more detail the two prominent methods of organizing metadata for temporal prefetchers, and we discuss their implications for metadata cacheability.

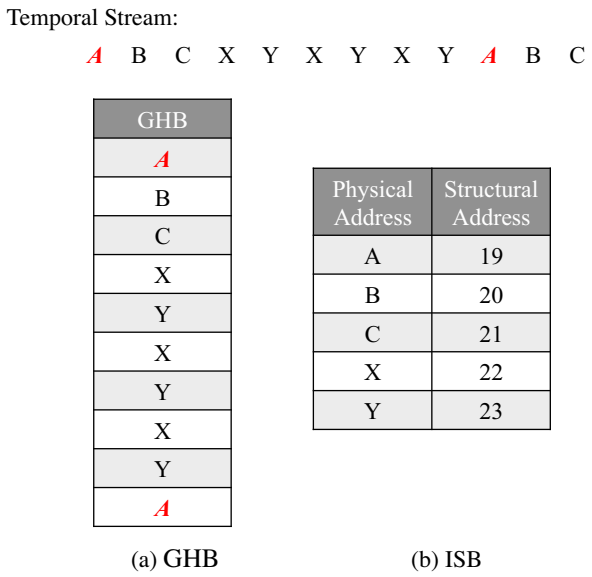

Figure 1: Metadata for GHB (left) and ISB (right).

STMS: Global History Buffer. The Global History Buffer (GHB) records all memory references in a circular FIFO buffer. Figure 1 shows how the GHB stores repeated occurrence of a temporal stream. On the second access to *A*, STMS retrieves the portion of the GHB from the last occurrence of *A* (entry 0 in our example) and issues prefetches for *B*. The GHB is then updated by appending *A* at the end. Thus, GHB records temporal information, and its size is proportional to the number of dynamic memory accesses.

To locate the last instance of *A*, the GHB is accompanied by an Index Table. For each memory address, the Index Table contains the GHB pointer corresponding to its last access. Thus, the size of the Index Table is proportional to the number of unique memory accesses.

ISB: Structural Address Space. The ISB maps correlated physical addresses to consecutive addresses in a new address space called the *structural address space*. Thus, for ISB, a temporal stream is a sequence of consecutive structural addresses that can be prefetched using a next line prefetcher (see Figure 1). ISB's metadata thus consists

of mappings from physical to structural addresses (PS mappings) and mappings from structural to physical addresses (SP mappings).

Figure 1 illustrates ISB's operation: When *A* is accessed the second time, ISB finds the PS mapping for *A* (19), predicts structural address 20 for prefetching, and finds the SP mapping for structural address 20 to generate the actual prefetch.

ISB's organization has two benefits. First, it can combine address correlation with PC-localization, which helps improve coverage and accuracy; PC-localization is a technique that segregates the global stream into sub-streams for each PC, such that each sub-stream is more predictable. Thus, addresses are considered to be correlated only if they appear consecutively in the PC-localized sub-stream. Second, ISB's metadata can be cached in on-chip metadata caches. ISB manages its *PS cache* and *SP cache* by synchronizing them with the contents of the TLB.

Metadata Caching. It is commonly believed that metadata for temporal prefetchers cannot be cached because the long reuse distance of metadata entries leads to poor temporal locality [2, 5]. We argue that metadata for temporal prefetchers *can* be cached if organized appropriately.

The GHB is difficult to cache for two reasons. First, it is organized as a circular time-ordered buffer, where the meaning of each buffer entry depends on its position relative to the other entries of the GHB, so it is difficult to cache a portion of the GHB without losing this positional information. Second, even if the GHB could be modified to support piecewise caching, it stores many redundant entries, which increase the reuse distance of individual GHB entries. For example, in Figure 1, the first GHB entry for *A* at index 1 will be accessed after several accesses to *X* and *Y*, each of which create a new entry in the GHB. Thus, to cache *A*'s entry, we would need a GHB cache with 12 entries (and an Index Table with 5 entries). We find that for SPEC and CloudSuite workloads, the GHB reuse distance is typically in the hundreds of thousands.

By contrast, ISB's mappings are not redundant: Each physical address maps to one structural address, and each structural address maps to one physical address. For the example in Figure 1, ISB assigns X and Y structural addresses 100 and 101, so it will have a total of 5 PS entries and 5 SP entries. For our evaluated workloads, we find that the number of entries in the SP cache is 6× smaller than those in the GHB, which translates to shorter reuse distances in the SP cache.

4 OUR SOLUTION

An effective metadata management scheme has two key goals: (1) hide the latency of off-chip metadata accesses so that prefetch decisions are not delayed waiting for off-chip metadata to arrive; (2) minimize the traffic overhead of temporal prefetching, considering both reads and writes.

To address these goals, MISB's metadata management scheme has three components—(1) a metadata cache, (2) a metadata prefetcher, and (3) a metadata filter that avoids issuing spurious metadata requests—that together allow MISB to judiciously manage metadata. We now describe each component in more detail.

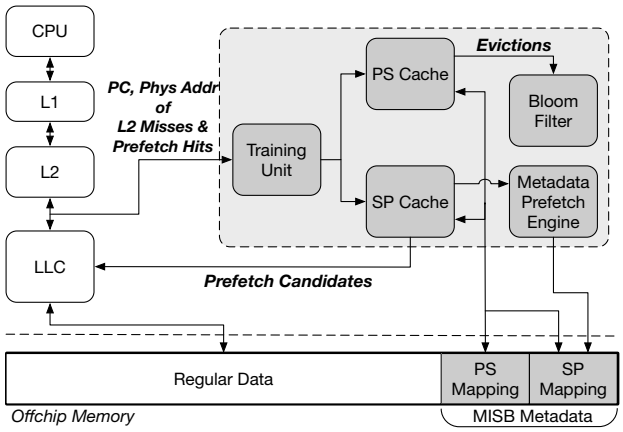

Figure 2: Overview of System with MISB Prefetcher

Metadata Caching. Metadata caching is important because on-chip metadata accesses have minimal latency and do not incur off-chip traffic. Thus, like ISB, MISB has two on-chip caches that store a small portion of its overall off-chip metadata. In particular, the *PS cache* stores mappings from physical addresses to structural addresses, while the *SP cache* stores structural-to-physical address mappings. However, unlike ISB, MISB manages its PS and SP caches at a fine granularity to ensure that the metadata cache is utilized efficiently. In particular, each logical metadata cache line in MISB's PS and SP caches holds one mapping, and on an eviction, the least recently used mapping is evicted. This fine granularity ensures that only useful mappings are retained. As explained in Section 3, such fine-grained caching is not possible with GHB-based prefetchers [2, 5] and was not used with ISB [3].

Metadata Prefetching. While caches reduce the latency and traffic for many metadata requests, caches alone cannot hide the latency of all metadata accesses, so to further hide latency, MISB includes a metadata prefetcher. The key insight behind this prefetcher is that because temporal streams are laid out sequentially in the structural address space, metadata can be accurately prefetched by deploying a simple next-line prefetcher on the SP cache. Significantly, on PS and SP metadata cache misses, MISB gets prefetching benefits from fetching a metadata cache line with 8 mappings. Because our metadata prefetching relies on the highly accurate temporal information in the structural address space, our metadata prefetching is also highly accurate.

While caches handle the latency and traffic concerns for many metadata requests, caches alone cannot hide the latency of all metadata accesses, so to further hide latency, MISB includes a metadata prefetcher. The key insight behind this prefetcher is that because temporal streams are laid out sequentially in the structural address space, metadata can be accurately prefetched by deploying a simple next-line prefetcher on the SP cache. Significantly, on PS and SP metadata cache misses, MISB gets prefetching benefits from fetching a metadata cache line with 8 mappings. Because our metadata prefetching relies on the highly accurate temporal information in the structural address space, our metadata prefetching is also highly accurate.

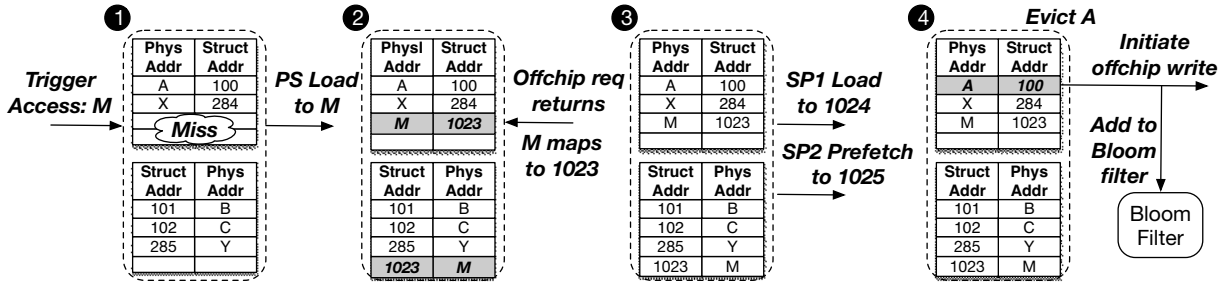

Figure 3: MISB: PS and SP Transactions.

Metadata Filtering. Finally, we observe that misses in the MISB's PS cache generate significant amounts of spurious metadata traffic because many PS load requests are to physical addresses for which MISB has no mapping since they have never been seen before. In particular, PS loads account for 65% of overall metadata traffic, and half of these requests are found to be invalid. Such requests increase traffic while providing no benefit. To filter these requests, MISB uses a Bloom Filter [23], which is a storage-efficient probabilistic data structure for tracking set membership. In particular, when a new PS mapping is written to off-chip storage, the mapping is added to a Bloom filter. Future PS misses are sent to memory only if the physical address is found in the Bloom filter. Because Bloom filters do not have false negatives, MISB does not filter any useful PS requests, but because Bloom filters can have false positives, they do admit useless PS requests. In Section 5.4, we analyze the impact of of the Bloom filter's inaccuracy on MISB.

4.1 Overall Operation

Figure 2 shows the overall design of MISB. The Training Unit finds correlated addresses within a PC-localized stream and assigns them consecutive structural addresses. On-chip mappings are stored in the PS and the SP caches, and on eviction, they are written to the corresponding off-chip PS and SP tables. The Bloom filter tracks valid PS mappings and filters invalid off-chip requests for PS mappings.

Conceptually, MISB's overall training and prediction algorithms are similar to ISB's, but they differ significantly in the way that MISB manages the movement of metadata between the on-chip metadata caches and off-chip metadata storage. We now describe MISB's metadata management scheme and its interactions with ISB's training and prediction algorithms.

Prediction. On a prediction, MISB first queries the on-chip PS cache with the trigger physical address. If the PS request hits, MISB predicts prefetch requests for the next few structural addresses. If the PS request misses, MISB issues an off-chip *PS load* request, delaying the prediction until the request completes. For example, in Figure 3-❶, the trigger address M misses in the PS cache and initiates a *PS load* for M. When the request completes, the new mappings are inserted into both the PS and SP caches, as indicated by the shaded entries in Figure 3-❷.

Regardless of whether the *PS load* hits or misses in the cache, when we find its structural address s, we issue a data prefetch request for structural address $s+1$, which causes MISB to query the on-chip SP mapping for $s+1$ (structural address 1024 in Figure 3-❸). If found, a data prefetch for the corresponding physical address is issued. If not found, the predicted structural address $s+1$ is placed in a small (32-entry) buffer and an *SP1 load* request is issued to off-chip memory. At the same time, future requests to structural addresses $s+2$, $s+3$ and so on are anticipated with the issuance of *SP2 prefetch* requests. For example, in Figure 3-❸, an *SP2 prefetch* request is issued for structural address 1025, assuming a metadata prefetch degree of 1. The degree of metadata prefetching can be tuned, and like PS requests, each SP request carries mappings for 8 consecutive structural addresses. SP requests also fill both the PS and SP caches. In Section 5.4, we show that our metadata prefetching scheme improves hit rates for both PS and SP caches.

Training. MISB's training is similar to ISB's training algorithm. The Training Unit keeps track of consecutive memory references for a given PC and assigns PC-localized correlated addresses consecutive structural addresses. The on-chip PS and SP caches are updated with newly assigned mappings, and if mappings change, they are marked dirty so that they can be written to off-chip memory.

Metadata Organization. As shown in Figure 3, MISB's on-chip metadata caches are organized at a fine granularity, where each cache entry holds one mapping (8 bytes). A fine-grained organization ensures better metadata cache efficiency because individual mappings can be retained and evicted based on their usefulness. For example, if one portion of the stream is more likely to be reused than another, then our metadata cache can selectively retain mappings for the first portion and discard mappings for the second portion. To maximize off-chip bandwidth utilization, off-chip requests are issued at the granularity of 64 bytes (or 8 mappings). Unless specified otherwise, our metadata caches are 8-way set-associative.

Unlike ISB, both the PS caches and SP caches are managed using an LRU policy, which allows the PS and SP caches to retain the mappings that see the most utility in the respective caches. For example, in Figure 3, for the stream A, B, C, the PS Cache has the physical to structural mapping for A, and the SP cache has structural to physical mappings for B and C, but not A. Both our PS and SP caches are writeback caches, so dirty evictions in these caches result in an off-chip store request.

Finally, our off-chip storage includes two tables: The PS Table and the SP Table. *PS loads* are served by the PS Table, while *SP1 loads* and *SP2 prefetches* are served by the SP Table. By contrast, since ISB does not need the SP Table for prefetching, ISB maintains

just the off-chip PS Table and uses the PS entries to construct the on-chip SP table.

Bloom Filter. As shown in Figure 3-❹, on an off-chip store request from the PS cache, the corresponding store address is added to the Bloom filter to indicate that an off-chip mapping exists for this physical address. The Bloom filter is then probed on future *PS loads*, and the load is issued only if the Bloom filter confirms that a mapping will exist in off-chip memory.

An ideal Bloom filter with infinite resources can eliminate all false positives, but with limited resources, the Bloom filter produces false positives. To reduce false positives, we provision 17KB for the Bloom filter and use the h3 hash [24], which provides a good tradeoff between space efficiency and traffic. For more bandwidth-constrained environments, the Bloom filter budget can be increased to further reduce traffic.

5 EVALUATION

5.1 Methodology

For single-core configurations, we evaluate MISB using a proprietary cycle-level simulator that is correlated against the RTL of commercially-available CPU designs. This highly accurate and flexible simulator is developed and maintained by a team of engineers. Our generic CPU model implements the ARMv8 AArch64 ISA and uses the configuration shown in Table 1. The simulator employs a simple fixed-latency memory model, but it models bandwidth constraints accurately and stalls the execution accordingly. Our small page configuration uses pages sizes of 4KB, and our large page configuration uses a page size of 2MB.

Core	Out-of-order, 2GHz, 4-wide fetch, decode, and dispatch 128 ROB entries
TLB	48-entry fully-associative L1 I/D-TLB 1024-entry 4-way L2 TLB
L1I	64KB private, 4-way, 3-cycle latency
L1D	64KB private, 4-way, 3-cycle latency Stride prefetcher
L2	512KB private, 8-way, 7-cycle latency
L3	2MB per core, shared, 16-way 12-cycle latency
DRAM	Single-Core: 85ns latency, 32GB/s bandwidth Multi-Core: 8B channel width, 800MHz, tCAS=20, tRP=20, tRCD=20 2 channels, 8 ranks, 8 banks, 32K rows 32GB/s bandwidth total

Table 1: Machine Configuration

For multi-core configurations, we use ChampSim [25, 26], a trace-based simulator that includes an out-of-order core model with a detailed memory system. ChampSim's cache subsystem includes FIFO read and prefetch queues, with demand requests having higher priority than prefetch and metadata requests. The main memory model simulates data bus contention, bank contention, and bus turnaround delays; bus contention increases memory latency. The main memory read queue is processed out of order and uses a modified Open Row FR-FCFS policy. Our ChampSim simulations use the configuration in Table 1 and replicate single-core performance trends from our proprietary simulator[3].

Benchmarks. We present single-core results for all memory-bound workloads from SPEC2006 [27]. For detailed analyses, we choose a subset of benchmarks that are known to have irregular access patterns [3]. For SPEC benchmarks we use the reference input set. For all single-core benchmarks, we use SimPoints [28] to find representative regions. Each SimPoint has 30 million instructions, and we generate at most 30 SimPoints for each SPEC benchmark..

We present multi-core results for CloudSuite [29] and multi-programmed SPEC benchmarks. For CloudSuite, we use the traces provided with the 2^{nd} Cache Replacement Championship. The traces were generated by running CloudSuite in a full-system simulator to intercept both application and OS instructions. Each CloudSuite benchmark includes 6 samples, where each sample has 100 million instructions. We warm up for 50 million instructions and measure performance for the next 50 million instructions. For multi-programmed SPEC simulations, we simulate 4 benchmarks chosen uniformly randomly from all memory-bound benchmarks, and for 8-core results, we choose 8 benchmarks chosen uniformly randomly. Overall, we simulate 80 4-core mixes and 35 8-core mixes. For each mix, we simulate the simultaneous execution of SimPoints of the constituent benchmarks until each benchmark has executed at least 500 million instructions. To ensure that slow-running applications always observe contention, we restart benchmarks that finish early so that all benchmarks in the mix run simultaneously throughout the execution. We warm the cache for 30 million instructions and measure the behavior of the next 100 million instructions.

Prefetchers. We compare MISB against four irregular prefetchers, namely, Spatial Memory Streaming (SMS) [13], Sampled Temporal Memory Streaming (STMS) [2], Irregular Stream Buffer (ISB) [3], and Domino [5]. SMS captures irregular patterns by applying irregular spatial footprints across memory regions. STMS, ISB, and Domino represent the state-of-the-art in temporal prefetching. For simplicity, we model idealized versions of STMS and Domino, such that their off-chip metadata transactions complete instantly with no latency or traffic penalty. Thus, our performance results for STMS and Domino represent the upper bound of performance for these prefetchers. To estimate their traffic overhead, we count the number of metadata requests, but the requests are never issued to the memory system. **Throughout our evaluation, references to STMS and Domino refer to these idealized implementations.** We also try variants of STMS and Domino that cache the index table in a 32 KB on-chip cache and probabilistically update the off-chip index table [2]. These implementations also do not incur any latency and traffic penalty and are meant to evaluate the impact of probabilistic metadata update on traffic and performance.

For ISB and MISB, we faithfully model the latency and traffic of all metadata requests. For MISB, we use 49KB of on-chip storage,

[3]Absolute quantities such as IPC, MPKI, and traffic in GB/s do not match exactly between the two simulators, but the relative differences in these quantities are similar.

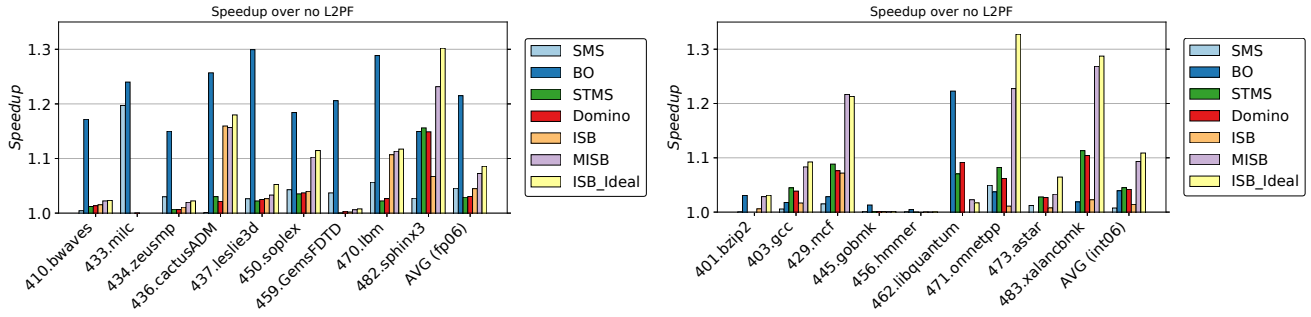

Figure 4: Comparison of Prefetchers on SPECfp 2006 (left) and SPECint 2006 (right).

which contains 32KB for the on-chip metadata cache and 17KB for the Bloom filter. We also compare against an idealized version of ISB which has access to all the metadata instantaneously, thereby representing an upper bound of performance for ISB and MISB.

Unless otherwise specified, all prefetchers train on the L2 access stream, and prefetches are inserted into the L2 cache. Unless otherwise specified, all prefetchers use a prefetch degree of 1, which means that they issue at most one prefetch on every trigger access.

We also evaluate MISB as the irregular component of a hybrid prefetcher that uses the Best Offset Prefetcher (BO) [6] as the regular prefetcher. We choose BO because it won the Second Data Prefetching Championship [30].

5.2 Single-Core Results

Figure 4 compares all the prefetchers on memory-intensive benchmarks from SPECfp (left) and SPECint (right). On SPECint, which mostly consists of challenging irregular benchmarks, MISB outperforms all prefetchers with a speedup of 9.3% vs. 4.5% for STMS, the second best prefetcher. On SPECfp, which mostly consists of regular benchmarks, BO outperforms all temporal prefetchers with an overall speedup of 21.5%. Temporal prefetchers do not perform well on regular benchmarks because they cannot prefetch compulsory misses, but we show later in this section that temporal prefetchers combine well with regular prefetchers. Because the benefit of temporal prefetching is most pronounced for irregular benchmarks, the rest of this section focuses on a subset of 7 irregular benchmarks (5 from SPECint and 2 from SPECfp).

Irregular SPEC2006. The top graph of Figure 5 shows that for the irregular SPEC2006 benchmarks, MISB outperforms all other prefetchers. Its 22.7% speedup comes close to the 26.9% speedup of an idealized ISB that incurs no metadata overhead. By contrast, realistic ISB achieves a 4.5% speedup, which illustrates the severe limitations of ISB's metadata management scheme on a modern system with a 2-level TLB. MISB also outperforms idealized versions of STMS (10.6% speedup) and Domino (9.5% speedup), which illustrates its benefits over unrealistically optimistic versions of GHB-based temporal prefetchers. Regular prefetchers, such as BO and SMS, do not perform well on irregular benchmarks, achieving only 6.3% and 2.3% speedup, respectively.

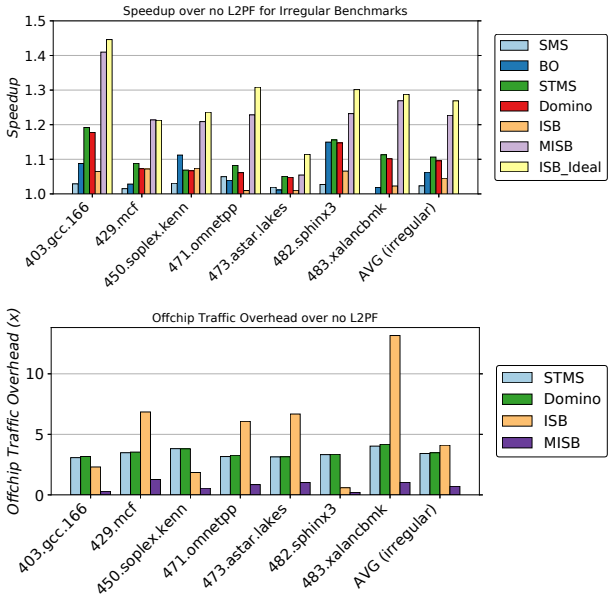

Figure 5: Irregular SPEC2006 Results

The bottom graph of Figure 5 shows that MISB's traffic overhead is significantly lower than that of the other prefetchers. In particular, MISB's traffic overhead over a baseline with no prefetching is 70%, while STMS, Domino, and ISB incur five to six times more traffic (342% for STMS, 348% for Domino, and 411% for ISB). The overhead includes traffic due to metadata requests and useless prefetches, but as we will see, ISB and MISB issue very few useless prefetches, so the vast majority of their traffic overhead can be attributed to metadata requests. We expect these traffic savings to translate directly to both energy and power savings.

MISB's traffic overhead can be reduced from 70% to 45% by using it at the L3 cache (train on L3 accesses and prefetch into the L3), but this reduction in traffic comes at the cost of performance, as speedup is reduced from 22.7% to 19.0%.

Figure 6 shows that probabilistic update [2] reduces STMS' traffic at the cost of performance. In particular, STMS with probabilistic

update reduces STMS' speedup from 10.6% to 5.4%, and it reduces traffic overhead from 342% to 209%, which is still much higher than MISB's traffic overhead of 70%.

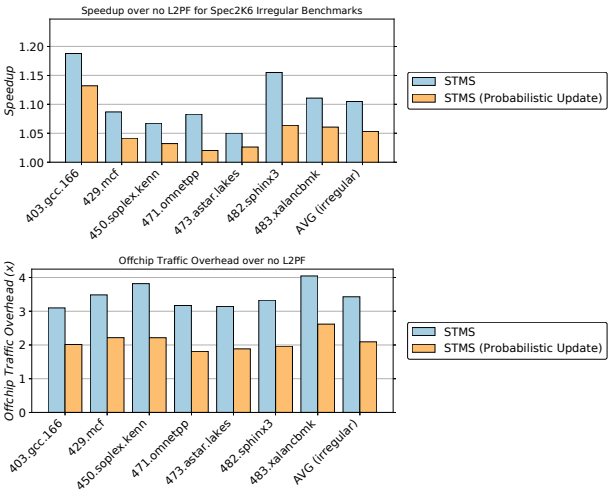

Figure 6: Impact of Probabilistic Update on STMS

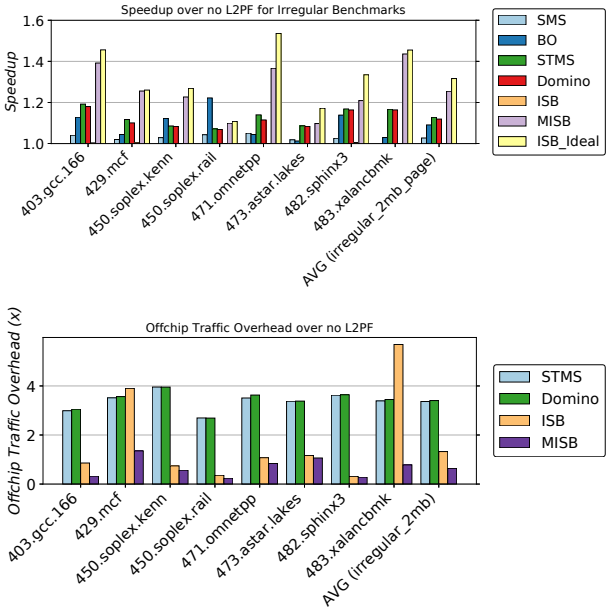

Figure 8: Large Page Results

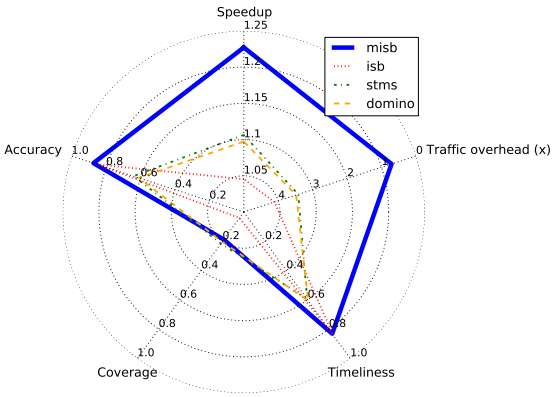

Figure 7: MISB Improves Along Multiple Dimensions.

To summarize MISB's benefits, Figure 7 shows that MISB outperforms other temporal prefetchers in nearly all dimensions, including accuracy and timeliness. Like ISB, MISB has high accuracy (87.3% for MISB vs. 64.1% for STMS and 60.9% for Domino) and good timeliness (83.1% for MISB vs. 59.6% for STMS and 60.4% for Domino). MISB's 18.8% coverage is slightly lower than that of Domino (20.3%) and STMS (21.5%) because our idealized implementations of Domino and STMS do not incur any latency for accessing off-chip metadata, whereas for MISB, the metadata latency causes a 5.0% loss in coverage. Nevertheless, MISB achieves higher speedup than idealized STMS and Domino because benefits in other dimensions easily compensate for the small loss in coverage.

Large Page Workloads. Figure 8 shows that MISB retains its benefits in the presence of 2MB pages. In particular, MISB achieves 25.5% speedup over no L2 prefetching (vs. 9.1% for BO, 2.7% for SMS, 12.8% for STMS, and 12.0% for Domino). As we would expect, with huge pages, ISB sees only a 0.2% speedup because at 8MB, the metadata for TLB-resident pages is too large for ISB to maintain in its on-chip caches. MISB retains its traffic benefits with large pages: Its traffic overhead is 64%, which is much lower than ISB's 132%, Domino's 340%, and STMS' 337%.

Hybrid Prefetchers. It would be difficult to imagine a chip vendor providing an irregular prefetcher without also providing a regular prefetcher, so we combine each of our temporal prefetchers with BO and SMS. Figure 9 shows that for the irregular subset of SPEC2006, the BO-MISB hybrid outperforms other hybrids with a 25.6% speedup (vs. 14.1% for BO-STMS). Since BO alone sees only a 6.3% speedup, we conclude that the remaining performance benefit comes from MISB's ability to prefetch irregular memory access patterns. If we further add SMS to the hybrid prefetcher, the BO-SMS-MISB hybrid achieves a 26.2% speedup. For SPECfp benchmarks, the BO-MISB hybrid improves performance by 23.9%, a slight improvement over BO alone (21.5% speedup).

5.3 Multi-Core Results

We now evaluate MISB on multi-core configurations.

CloudSuite Benchmarks. Figure 10 shows that a realistic MISB outperforms idealized STMS and idealized Domino on CloudSuite benchmarks, even though the idealized prefetchers incur no latency

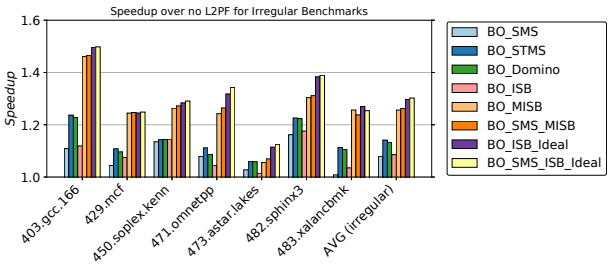

Figure 9: Hybrid Results

or traffic penalty for metadata accesses[4]. In particular, MISB improves performance by 7.2%, while idealized STMS and Domino improve performance by 4.0% and 3.9%, respectively. These performance improvements can be explained by MISB's superior coverage (31.0% for MISB vs. 13.6% for STMS and 13.4% for Domino) and accuracy (89.8% for MISB vs. 79.0% for STMS and 77.7% for Domino).

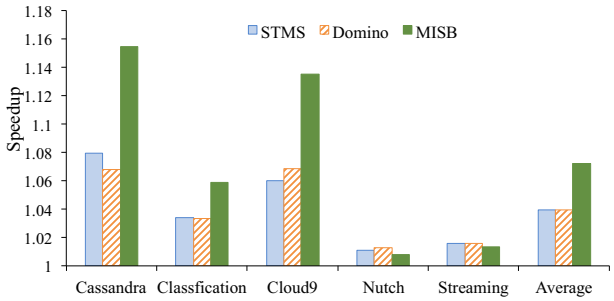

Figure 10: Speedup Comparison on CloudSuite

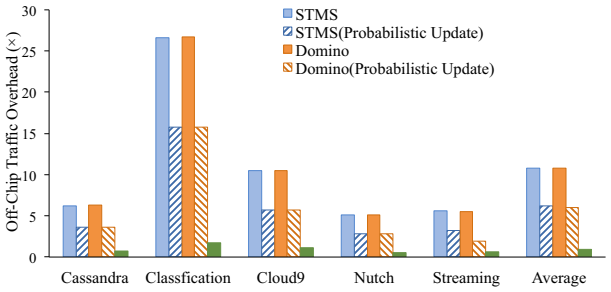

Figure 11: Traffic Comparison on CloudSuite

Figure 11 shows that MISB's metadata traffic overhead is significantly lower than that of STMS and Domino: MISB's traffic overhead is 96.2%, while idealized STMS' and Domino's are 1082.7% and 1081.5%, respectively. The traffic overhead of STMS and Domino can be reduced to 621.6% and 596.9%, respectively, by employing probabilistic updates to the off-chip structures [2], but this optimization degrades performance. For STMS, the performance drops from 4.0% to 2.0%, whereas for Domino, performance drops from 3.9% to 1.8%.

[4]For CloudSuite workloads, we train all prefetchers on L2 misses instead of L2 accesses, which results in better IPC and lower traffic for all prefetchers.

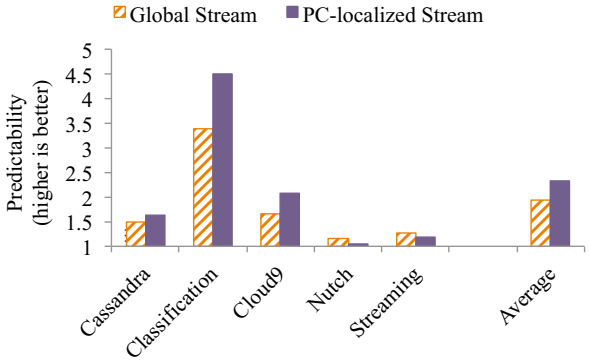

Figure 12: Benefits of PC-Localization For CloudSuite

Our results show that contrary to prior claims [5], PC-localization is quite beneficial for server benchmarks. Figure 12 compares the compressibility of PC-localized cache access streams to global access streams, showing that PC-localized streams are more compressible and therefore more predictable than the global stream.[5] These results also explain MISB's higher coverage and accuracy on server workloads. A second concern [5] is that PC-localized predictions are untimely for server workloads because instructions repeat much less frequently than in scientific workloads. Our results show that timeliness is not an issue when prefetching into the L2 or L3 (prior work prefetches into a prefetch buffer that is probed in parallel to the L1 [5]). In fact, at the LLC, MISB is more timely than even idealized STMS and Domino.

Multi-Programmed SPEC Benchmarks. To avoid aggravating memory pressure, low metadata overhead is critical for scaling the benefits of temporal prefetchers to multi-core systems. MISB works well for 4-core and 8-core systems. On 4-core multi-programmed workloads, a realistic MISB improves performance by 19.9%, compared to 8.8% for STMS and 9.6% for Domino. On 8-cores, MISB's benefit reduces to 12.1% because the metadata traffic overhead starts to stress available bandwidth, but the top graph in Figure 13 shows that MISB still outperforms idealized versions of STMS (7.5% speedup) and Domino (7.6% speedup) that do not incur performance penalty for metadata traffic. The bottom graph in Figure 13 shows that the key to MISB's scalability is its low traffic overhead, which is 72.5%, while idealized STMS and Domino incur 304.7% and 306.8% traffic overhead respectively.

5.4 Understanding MISB's Benefits

The left graph in Figure 14 shows that MISB's benefits depend on its metadata cache and prefetcher working in concert. We make two observations. First, without metadata prefetching, MISB's speedup

[5]We use the Sequitur algorithm [31] to compute compressibility of global and per-PC streams. Given a sequence of symbols, Sequitur constructs a compressed representation of the sequence by substituting repeating phrases with concise *rules*.

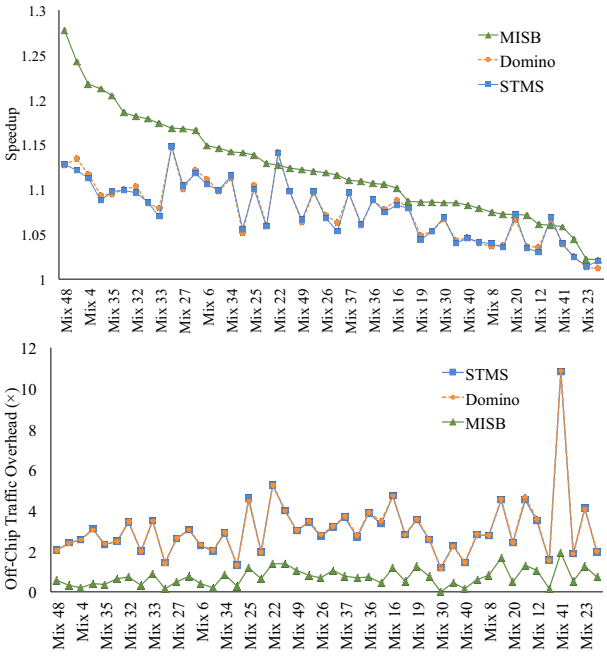

Figure 13: MISB Scales to 8-Core Systems.

is reduced from 22.7% to 6.5%; without an adequate metadata cache budget [6], its speedup is reduced from 22.7% to 8.9%. Second, MISB's caching and prefetching scheme can be applied even in the absence of PC-localization, but the loss of PC-localization severely hurts performance, reducing speedup from 22.7% to 7.3%.

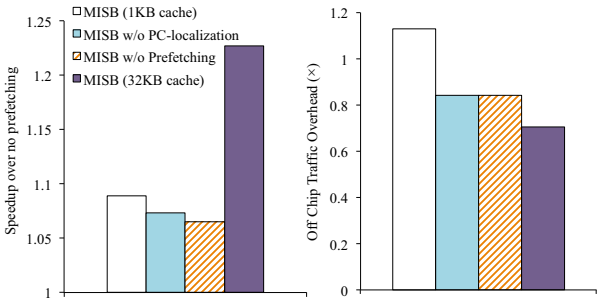

Figure 14: MISB Benefits from Both Metadata Caching and Prefetching.

The right graph in Figure 14 shows that metadata caching significantly reduces MISB's traffic overhead. If we were to reduce the metadata cache budget from 32KB to 1KB, traffic overhead increases from 70% to 113%.

[6]We reduce the metadata cache size from 32KB to 1KB to evaluate the benefit of metadata caching. The specifics of the MISB design require a little bit of on-chip metadata cache to properly train off-chip metadata.

Metadata Cache Hit Rates. Figure 15 shows that for both the PS and SP caches, MISB's metadata management yields significantly better hit rates than ISB's (43.0% vs. 27.1% for the PS cache, and 66.5% vs. 32.5% for the SP cache). MISB's improved hit rates are primarily caused by its accurate metadata prefetching. We find that more than 90% of metadata retrieved by ISB's TLB-sync scheme is never used, which both hurts metadata cache efficiency and incurs high traffic overhead.

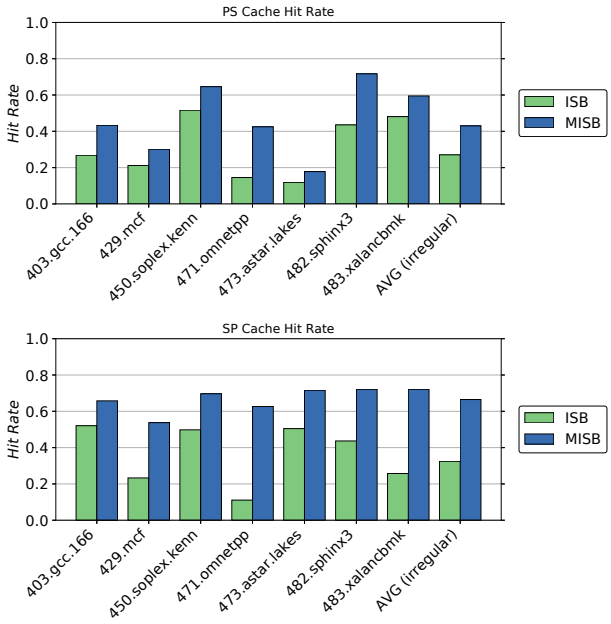

Figure 15: On-Chip Metadata Cache Hit Rate.

Metadata Traffic. Finally, Figure 16 shows a breakdown of MISB's off-chip prefetcher traffic. We see that our Bloom filter reduces spurious PS loads by not issuing traffic requests marked in striped blue, resulting in traffic savings of 8.5% (78.5% traffic overhead without the bloom filter vs. 70.0% traffic overhead with the bloom filter). We also see that by reducing the Bloom filter's false positive rate (unfiltered PS loads), we can further reduce traffic.

5.5 Sensitivity studies

Prefetch Degree. Figure 17 shows that all prefetchers benefit from higher degree, but MISB consistently outperforms both STMS and Domino at all prefetch degrees. At the same time, MISB is highly accurate, and it retains its accuracy advantage even at higher degrees. At degree 8, MISB's accuracy is 77%, while STMS and Domino have an accuracy of 47%.

Metadata Cache Size. Figure 18 shows the traffic and speedup of MISB at different storage budgets. Each point in the graph represents a pareto-optimal configuration at the corresponding storage budget, and the labels show the distribution of the metadata cache size and the bloom filter for each point (the metadata cache size is preceded

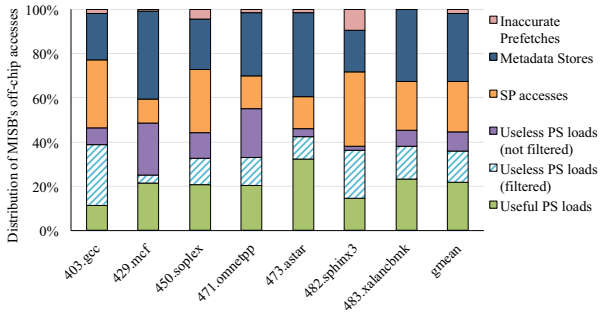

Figure 16: Traffic Breakdown for MISB.

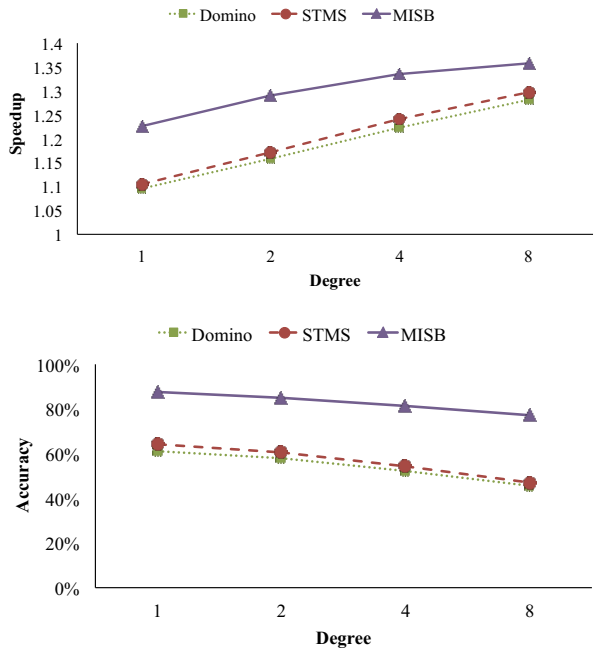

Figure 17: Comparison at Higher Degree.

by the letter *c*, and the bloom filter size is preceded by the letter *b*). The top graph shows that MISB needs a least 8KB of metadata cache to realize its full performance benefit. Without a metadata cache, MISB cannot hide the latency of all metadata accesses, which severely hurts its performance. The bottom graphs shows that as we increase the storage budget, metadata traffic keeps decreasing, which suggests that MISB uses the metadata cache effectively.

Latency Tolerance. To evaluate MISB's ability to tolerate latency, Figure 19 shows that as we vary the latency of off-chip metadata transfer, MISB can tolerate metadata latencies of up to 1000 cycles. We believe that PC-localization is critical here, since it provides a sufficiently large time gap between consecutive accesses in a temporal stream that metadata can be retrieved without delaying future prefetch requests.

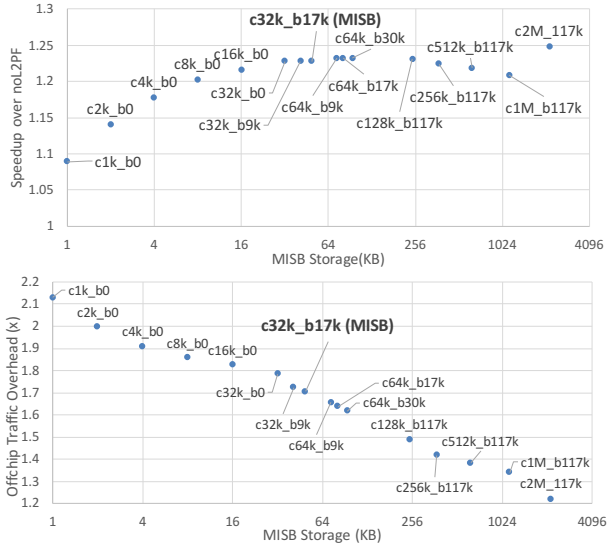

Figure 18: Sensitivity to Metadata Storage Budget.

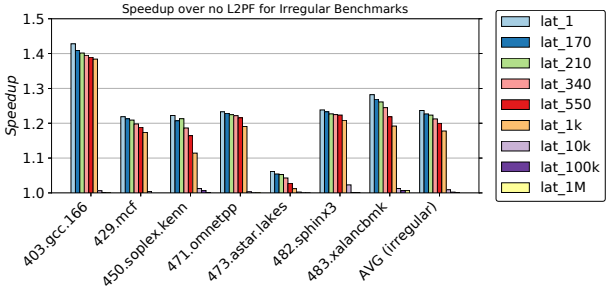

Figure 19: Memory Latency Sweep (in terms of CPU cycles).

Bloom Filter Reset Interval. Since Bloom Filters saturate over time, we reset MISB's Bloom Filter every 30 million instructions. A shorter reset interval reduces both prefetcher coverage and metadata traffic. For example, Figure 20 shows that a reset interval of 5 million instructions reduces speedup by 1% and reduces traffic by 2%.

As an alternative solution, we consider Stable Bloom Filters [32], which continuously evict stale information to make room for more recent elements. However, Stable Bloom Filters increase the area overhead and also add false negatives, so we choose to reset the Bloom Filter instead.

TLB Shootdowns and Context Switches. Events such as TLB shootdowns and page remappings force MISB to relearn the metadata for corresponding pages, while events such as context switches erase MISB's on-chip metadata. To evaluate the impact of such events, we consider the worst-case scenario, where *all* of MISB's metadata (on-chip and off-chip) is reset periodically. Figure 21 shows that even in this extreme case, MISB performs well: For a reset interval of 15 million instructions (long enough for a typical context switch), speedup and traffic reduce marginally. Speedup decreases

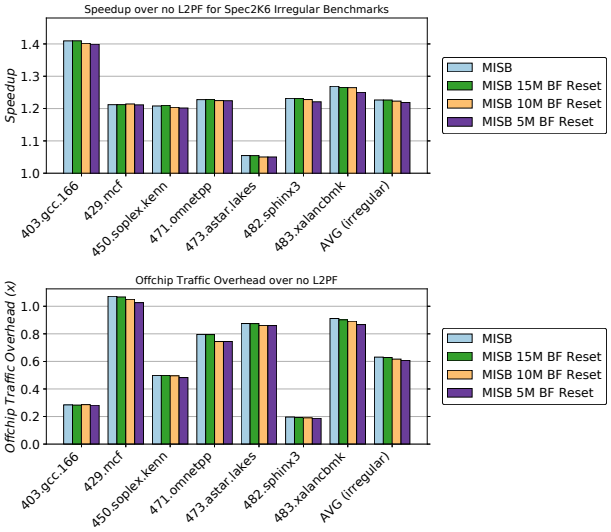

Figure 20: Bloom Filter Reset Interval

gradually as we further reduce reset intervals, and when the reset interval is as short as 5 million instructions, MISB still improves performance by 18% over a baseline with no prefetching.

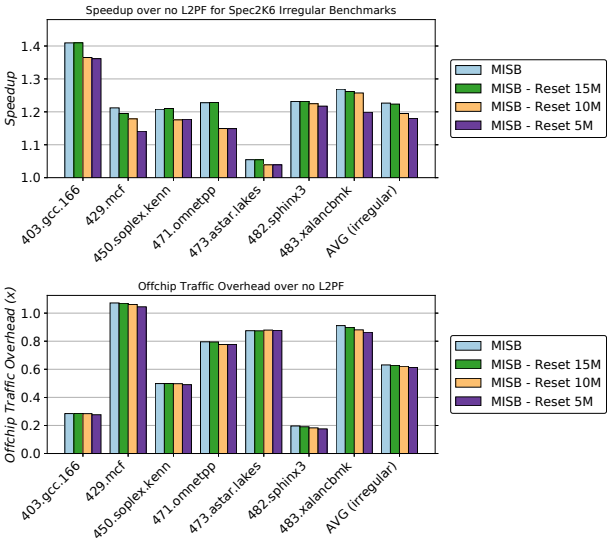

Figure 21: Impact of resetting MISB's metadata

6 CONCLUSIONS

Temporal prefetchers can provide good coverage for irregular memory access patterns, but a key impediment to their commercial adoption has been the high cost of accessing off-chip metadata. The ISB addressed this problem by introducing the structural address space, which provided two benefits: (1) It supported PC-localization and (2) it enabled metadata to be cached. Unfortunately, we have shown in this paper that the ISB metadata caching scheme is not as effective as originally suggested, partly because it caches metadata at a coarse granularity. However, we have also shown that ISB's structural address space has benefits that were not recognized in the original paper. In particular, as embodied by our new MISB prefetcher, the structural address space lends itself to fine-grained caching and accurate metadata prefetching, which together can hide latency and significantly reduce traffic.

Our results show that MISB reduces traffic to the point of commercial viability. Our results from a highly accurate proprietary simulator show that MISB comes close to the performance of an idealized ISB for a variety of workloads on a variety of machine configurations. On single core systems running SPEC 2006 benchmarks, MISB improves performance by 22.7% (vs. 4.5% for ISB and 10.6% for idealized STMS), while reducing off-chip traffic to 70% (vs. 411% for ISB and 342% for STMS). On 4-core systems running CloudSuite workloads, MISB improves performance by 7.2% (vs. 3.9% for idealized Domino and 3.9% for idealized STMS), while reducing traffic overhead to 96.2% (vs. 1082.7% for STMS and 1081.5% for Domino).

ACKNOWLEDGMENTS

We thank Jaekyu Lee for his help in setting up the proprietary simulation infrastructure, and we thank Molly O'Neil and Curtis Dunham for excellent feedback on an early draft of this paper. This work was funded in part by NSF Grant CCF-1823546 and a gift from Intel Corporation through the NSF/Intel Partnership on Foundational Microarchitecture Research.

REFERENCES

[1] T. F. Wenisch, *Temporal Memory Streaming*. PhD thesis, Carnegie Mellon University, Department of Computer Science, 2007.

[2] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Practical off-chip meta-data for temporal memory streaming," in *15th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 79–90, 2009.

[3] A. Jain and C. Lin, "Linearizing irregular memory accesses for improved correlated prefetching," in *46rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2013.

[4] K. J. Nesbit and J. E. Smith, "Data cache prefetching using a global history buffer," in *10th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 90–97, 2005.

[5] M. Bakhshalipour, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Domino temporal data prefetcher," in *24th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 131–142, 2018.

[6] P. Michaud, "Best-offset hardware prefetching," in *22th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[7] C.-K. Luk and T. C. Mowry, "Compiler-based prefetching for recursive data structures," in *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 222–233, September 1996.

[8] A. Roth, A. Moshovos, and G. S. Sohi, "Dependence based prefetching for linked data structures," in *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 115–126, 1998.

[9] J. Collins, S. Sair, B. Calder, and D. M. Tullsen, "Pointer cache assisted prefetching," in *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO)*, pp. 62–73, 2002.

[10] A. Roth and G. S. Sohi, "Effective jump-pointer prefetching for linked data structures," in *Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA)*, pp. 111–121, 1999.

[11] R. Cooksey, S. Jourdan, and D. Grunwald, "A stateless, content-directed data prefetching mechanism," in *Proceedings of the 10th International Conference*

on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 279–290, October 2002.

[12] E. Ebrahimi, O. Mutlu, and Y. N. Patt, "Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems," in *15th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 7–17, 2009.

[13] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *Proceedings of the 33th Annual International Symposium on Computer Architecture (ISCA)*, pp. 252–263, 2006.

[14] X. Yu, C. J. Hughes, N. Satish, and S. Devadas, "IMP: indirect memory prefetcher," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*, pp. 178–190, ACM, 2015.

[15] D. Joseph and D. Grunwald, "Prefetching using Markov predictors," in *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA)*, pp. 252–263, 1997.

[16] T. M. Chilimbi, "Efficient representations and abstractions for quantifying and exploiting data reference locality," in *SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 191–202, 2001.

[17] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi, "Temporal streaming of shared memory," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 222–233, May 2005.

[18] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-temporal memory streaming," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, pp. 69–80, 2009.

[19] Y. Chou, "Low-cost epoch-based correlation prefetching for commercial applications," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 301–313, 2007.

[20] Y. Solihin, J. Lee, and J. Torrellas, "Using a user-level memory thread for correlation prefetching," in *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA)*, pp. 171–182, 2002.

[21] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos, "Temporal streams in commercial server applications," in *IEEE International Symposium on Workload Characterization*, pp. 99–108, 2008.

[22] I. Burcea, S. Somogyi, A. Moshovos, and B. Falsafi, "Predictor virtualization," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pp. 157–167, ACM, 2008.

[23] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[24] M. V. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Transactions on Computers*, vol. 46, pp. 1378–1381, December 1997.

[25] J. Kim, E. Teran, P. V. Gratz, D. A. Jiménez, S. H. Pugsley, and C. Wilkerson, "Kill the program counter: Reconstructing program behavior in the processor cache hierarchy," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 737–749, 2017.

[26] *2nd Cache Replacement Championship*, 2017.

[27] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Computer Architecture News*, vol. 34, pp. 1–17, September 2006.

[28] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 45–57, ACM, 2002.

[29] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 37–48, 2012.

[30] *2nd Data Prefetching Championship*, 2015.

[31] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *Journal of Artificial Intelligence Research*, vol. 7, pp. 67–82, 1997.

[32] F. Deng and D. Rafiei, "Approximately detecting duplicates for streaming data using stable bloom filters," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 25–36, ACM Press, 2006.