Linearizing Irregular Memory Accesses for Improved Correlated Prefetching

Akanksha Jain Calvin Lin
Department of Computer Science
The University of Texas at Austin
Austin, Texas 78712, USA
{akanksha, lin}@cs.utexas.edu

ABSTRACT

This paper introduces the Irregular Stream Buffer (ISB), a prefetcher that targets irregular sequences of temporally correlated memory references. The key idea is to use an extra level of indirection to translate arbitrary pairs of correlated physical addresses into consecutive addresses in a new *structural address space*, which is visible only to the ISB. This structural address space allows the ISB to organize prefetching meta-data so that it is simultaneously temporally and spatially ordered, which produces technical benefits in terms of coverage, accuracy, and memory traffic overhead.

We evaluate the ISB using the Marss full system simulator and the irregular memory-intensive programs of SPEC CPU 2006 for both single-core and multi-core systems. For example, on a single core, the ISB exhibits an average speedup of 23.1% with 93.7% accuracy, compared to 9.9% speedup and 64.2% accuracy for an idealized prefetcher that over-approximates the STMS prefetcher, the previous best temporal stream prefetcher; this ISB prefetcher uses 32 KB of on-chip storage and sees 8.4% memory traffic overhead due to meta-data accesses. We also show that a hybrid prefetcher that combines a stride-prefetcher and an ISB with just 8 KB of on-chip storage exhibits 40.8% speedup and 66.2% accuracy.

Categories and Subject Descriptors

B.3 [**Memory Structures**]: Miscellaneous

General Terms

Design, Performance, Experimentation

Keywords

Prefetching

1. INTRODUCTION

Prefetching is an important technique for hiding the long memory latencies of modern microprocessors. For regular memory access patterns, prefetching has been commercially successful because stream and stride prefetchers are effective, small, and simple. For irregular access patterns, prefetching has proven to be more problematic. Numerous solutions have been proposed [3-4, 8-15, 17, 19, 22-23, 25-28, 32-34, 37-44], but there appears to be a basic design tradeoff between storage and effectiveness, with large storage required to achieve good coverage and accuracy [40].

For example, prefetchers based on *address correlation,* the subject of this paper, identify sequences of correlated memory addresses—also known as *temporal streams*—by learning the most likely successor for a given memory reference. Because this correlation information grows in proportion to the application's memory footprint, the fundamental challenge for these prefetchers is the management of megabytes of off-chip correlation information [8, 41, 43]. Access to this off-chip meta-data increases prediction latency and memory traffic, which reduces the effectiveness of prefetching.

Recent solutions use the Global History Buffer (GHB) [28], which organizes correlation information by storing recent memory accesses in a time-ordered circular *history buffer*; a spatially organized index table is used to find addresses within the history buffer (see Figure 1). With the temporally ordered history buffer, temporal streams can be efficiently prefetched because each stream is stored contiguously. For address correlation, GHB-based prefetchers can amortize the cost of off-chip meta-data access by fetching long temporal streams [43, 9]. Unfortunately, temporal organizations cannot effectively hide the latency of fetching meta-data for short streams, and even the most optimized implementations incur an average memory traffic overhead of 35% on commercial and scientific workloads [42].

One way to reduce the cost of these off-chip accesses would be to cache only the meta-data that correspond to the TLB-resident pages of memory. The movement of this cached meta-data to and from DRAM could then be synchronized with expensive TLB evictions, largely hiding the latency of these off-chip accesses.

Unfortunately, this proposed caching scheme is ill-suited to temporally organized structures such as the GHB. For example, assume in Figure 1 that physical addresses `B`, `X`, and `D` reside on the same page; we see that these addresses are scattered throughout the history buffer and are likely to appear multiple times in the history buffer, so there is no efficient way to evict these entries from the history buffer

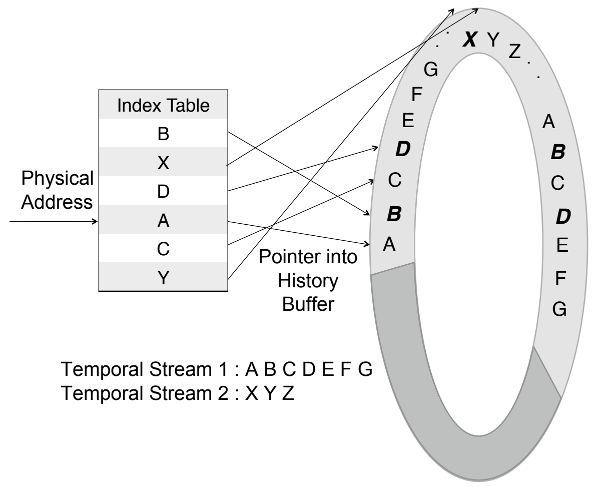

Index Table
B
X
D
A
C
Y
Physical Address
Pointer into History Buffer
Temporal Stream 1 : A B C D E F G
Temporal Stream 2 : X Y Z
A B C D E F G X Y Z
A B C D E F G

Figure 1: Address correlation using the GHB.

when their TLB entry is evicted, nor is it easy to reuse the scattered evicted entries of the history buffer.

This paper introduces the Irregular Stream Buffer (ISB), a new correlation-based prefetcher that employs just such a caching scheme and that provides other significant benefits with respect to coverage and accuracy. The main idea is to introduce an extra level of indirection to create a new *structural address space* in which correlated physical addresses are assigned consecutive structural addresses. The key point is that in this structural address space, streams of correlated memory addresses are both temporally ordered and spatially ordered. For example, we see in Figure 2 that a sequential traversal of the structural address space visits the elements of the irregular temporal stream—`A`, `B`, `C`, `D` and `E`—in temporal order. Thus, the problem of prefetching irregular streams is reduced to sequential prefetching in the structural address space. The mapping to and from structural addresses is performed at a cache line granularity by two spatially indexed on-chip address caches whose contents can be easily synchronized with that of the TLB.

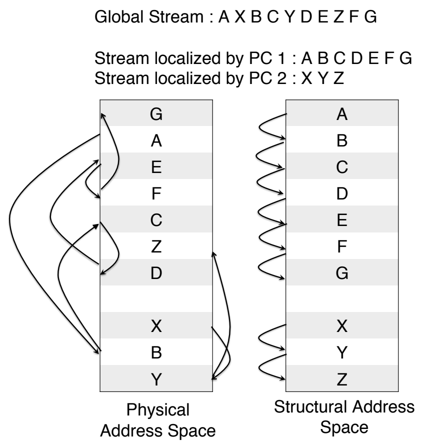

Global Stream : A X B C Y D E Z F G
Stream localized by PC 1 : A B C D E F G
Stream localized by PC 2 : X Y Z
G
A
E
F
C
Z
D
X
B
Y
Physical Address Space
A
B
C
D
E
F
G
X
Y
Z
Structural Address Space

Figure 2: Structural address space.

In addition to the reduced memory traffic provided by our caching scheme, the ISB enjoys several other significant benefits:

- *Improved Prediction Capability:* Unlike GHB-based solutions (see Section 3), the ISB can use PC localization, a technique that segregates the prefetcher input into multiple streams based on the PC of the loading instruction, which is known to improve coverage and accuracy [28, 39, 38, 25]. In particular, the ISB can combine PC localization and address correlation because any PC-localized temporal stream is stored consecutively in the on-chip address cache (see Figure 2), i.e., the localization is performed before physical addresses are translated to structural addresses.

- *Training on the Reference Stream:* Because the vast majority of its meta-data accesses are on-chip, the ISB can train on the LLC (last level cache) access stream instead of its miss stream, which significantly improves the predictability of the reference stream. By contrast, most previous prefetchers that use address correlation train on the LLC miss stream to avoid the significant off-chip traffic that would be generated by accessing off-chip meta-data on every LLC access.[1]

- *Support for Short Streams:* The ISB's caching scheme greatly reduces memory traffic overhead for all streams, not just for long streams.

This paper makes the following contributions:

1. We introduce the ISB, the first prefetcher to combine the use of PC localization and address correlation.

2. We show—using the irregular, memory-intensive subset of the SPEC 2006 benchmarks—that the ISB significantly advances the state-of-the-art in temporal stream prefetching. The ISB obtains 23.1% speedup and 93.7% accuracy, while an idealized STMS prefetcher, which over-approximates the previous state-of-the-art (STMS) [42], obtains 9.9% speedup and 64.2% accuracy. We also show that the ISB is superior to two other recent prefetchers, SMS [39], which exploits spatial locality, and PC/DC [28, 13], which uses delta correlation instead of address correlation.

3. We introduce a method of organizing data that synchronizes the movement of prefetcher meta-data with TLB misses to reduce memory traffic overhead. For a single core with DDR2 memory, the ISB incurs an average of 8.4% memory traffic overhead due to meta-data access. As a point of comparison, Wenisch, et al. report that the STMS prefetcher produces an average memory traffic overhead of roughly 35% for a mix of commercial and scientific workloads [42].

4. We show that the ISB performs well when combined with a state-of-the-art stride prefetcher (AMPM) [20]. A hybrid that uses an 8 KB ISB achieves a 40.8% speedup over a baseline with no prefetching.

This paper is organized as follows. Section 2 places our work in the context of prior work. Section 3 motivates our

[1]The STeMS prefetcher can train on the access stream because it searches for coarse-grained temporal streams, relying on a complex spatial prefetcher to fill in the gaps [38].

solution by describing the technical issues with pure spatial and purely temporal organizations of correlation information. Section 4 then describes our solution, and Section 5 evaluates our solution, before we conclude.

2. RELATED WORK

We now place our work in the context of prior work, first discussing techniques for improving spatial locality, and then discussing the considerable prior work in prefetching.

Improving Spatial Locality.

Spatial locality can be improved by re-ordering the layout of pointer-based data structures during memory allocation [7] or garbage collection [18], but both techniques involve expensive memory copying, and the former relies on programmer hints. Carter, et al., re-order memory accesses in a shadow address space [4] to improve locality and initiate prefetching, but their technique is limited to statically allocated data structures and requires both OS and programmer intervention.

Stride Prefetching.

Stride prefetchers generally target regular memory access patterns, building on Jouppi's next-line prefetcher [36, 24] by adding non-unit strides [29] and by predicting strides [1, 15]. Ishii, et al., introduce a clever data structure that compactly captures information about multiple stride lengths [20]. Sair, et al., support irregular streams by introducing a stride length predictor [34].

Hur and Lin enhance stream prefetchers by adding a small histogram of the stream lengths of recently seen memory accesses [19]. These histograms allow stream buffers to accurately prefetch tiny "streams" that might be as short as two cache lines, thereby providing some coverage for irregular memory accesses that stream buffers alone cannot prefetch.

Pointer-based Prefetching.

Pointer-based data structures are an important source of irregular memory accesses, so many techniques focus on prefetching pointers.

Compilers can insert prefetch instructions—known as *jump pointers*—for all children of a visited node of a linked data structure [26, 32]. The key issue with compiler-based solutions is poor timeliness; to hide long memory latencies, the software prefetches need to be inserted far from their use.

Hardware solutions, such as pointer caches [10] and hardware jump pointer tables [33], can issue timely prefetches but incur storage overheads of up to 1 MB; some also require compiler support and modifications to the ISA. Content Directed Prefetching (CDP) [11] is a stateless mechanism that searches through cache lines for pointer addresses that are then greedily prefetched. CDP is attractive in terms of storage requirements, but it wastes memory bandwidth because of its low accuracy.

Cooperative hardware-software approaches can combine the accuracy of software prefetching and the timeliness of hardware prefetching [33]. Guided Region Prefetching [40] uses static analysis to annotate load instructions with hints to the hardware prefetcher. Ebrahimi, et al. use compiler-guided filtering mechanisms to inform a CDP prefetcher about the pointers that are most likely to be fetched [14].

There are two key differences between the ISB and pointer-based approaches: (1) The ISB does not give special treatment to pointers, so it can exploit other sources of irregular memory accesses; (2) pointer-based approaches can prefetch compulsory misses, while the ISB cannot.

Prefetching Based on Spatial Locality.

Irregular memory accesses can also be prefetched by detecting spatial locality [22, 25, 3, 5]. Variations of the *Spatial Locality Detection Table* [22] track accesses to different regions of memory so that spatially correlated data can be prefetched together. These approaches typically need large tables to detect locality, but Somogyi, et al. [39] show how smaller tables can be used by correlating spatial locality with the program counter in addition to parts of the data address. As a result, *Spatial Memory Streaming (SMS)* can use tables as small as 64 KB, while achieving good performance improvements for commercial workloads.

Prefetching Based on Temporal Locality.

Joseph and Grunwald introduce the notion of correlation-based prefetching with their Markov Prefetcher [23], which uses a table to record possible successors of a given memory address. The presence of address correlation in applications has been studied both quantitatively [6] and qualitatively [41] for scientific and commercial workloads. Studies find that the length of correlated streams can vary from two to several hundred [6, 44], which implies that large amounts of storage are needed to prefetch these workloads effectively. While some designs reduce this on-chip table requirement [17], the table size still grows in proportion to the application's active memory footprint. Thus, a variety of solutions store the Markov table off-chip and optimize the memory bandwidth requirements and prefetch look-ahead distance for off-chip table access [8, 37].

Nesbit and Smith introduce the GHB as a general structure for prefetching streams of temporally correlated memory requests [28]. However, when used to record address correlation [42], the GHB is quite large, requiring about 4 MB of off chip storage for scientific workloads and about 48 MB for commercial server workloads. Thus, Wenisch, et al.'s STMS prefetcher introduces latency and memory traffic optimizations for reading and updating the off-chip history buffer and index table [43]. These techniques reduce the memory traffic from $3\times$ [8, 37, 44] to 1.05-$1.75\times$ [43] for long streams. Rather than use address correlation, other GHB-based prefetchers use delta correlation [28, 27], whose space requirements are dramatically smaller, but we show that for irregular accesses, delta correlation leads to low coverage and accuracy.

PC localization has been used to improve the accuracy and coverage of correlation-based prefetchers [29, 25, 28, 39, 38], but until now, the combination of PC localization and address correlation has been too expensive to be practically considered.

Finally, Diaz et al. propose a method of chaining PC-localized streams for better prefetch timeliness [12]. The ISB is orthogonal to these ideas, so it is possible to use stream chaining to link various PC-localized streams in an ISB design, but we do not explore this option in this paper.

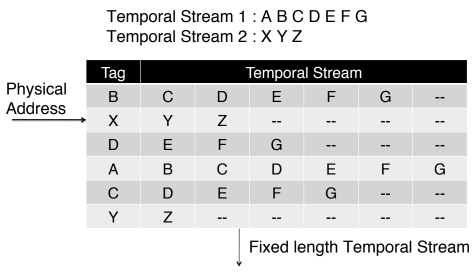
Spatial-Temporal Prefetching.

The best known irregular prefetcher, Somogyi, et al.'s STeMS prefetcher [38], exploits temporal correlation at a coarse granularity and spatial correlation at a finer granularity, essentially learning temporal sequences of spatial regions. The ISB could be employed in a similar fashion to identify the coarse-grain temporal stream, but we do not explore this idea in this paper.

3. THE PROBLEM

To motivate the benefits of the ISB's structural address space, this section explains the problems caused by purely spatial and purely temporal organizations of correlation information.

Early solutions organize correlated address pairs *spatially* in a *Markov table,* which is indexed by memory address [23]. Unfortunately, Markov tables require multiple table lookups to prefetch temporal streams. To reduce the number of table lookups, each table entry could store a fixed-length stream [8], but because temporal stream lengths vary widely from two to several hundred [6, 44], it is difficult to optimize for any single stream length. Thus, fixed-length stream entries lead to inefficient use of on-chip storage, with short streams wasting space (see the entries for Tag X and Y in Figure 3), and long streams storing data redundantly (see the entries for Tags A, B, C in Figure 3).


Temporal Stream 1 : A B C D E F G
Temporal Stream 2 : X Y Z

Physical Address

Tag	Temporal Stream					
B	C	D	E	F	G	--
X	Y	Z	--	--	--	--
D	E	F	G	--	--	--
A	B	C	D	E	F	G
C	D	E	F	G	--	--
Y	Z	--	--	--	--	--

Fixed length Temporal Stream

Figure 3: Markov Table with fixed length temporal streams.

The GHB instead stores correlation information temporally, which supports efficient temporal stream prefetching. Unfortunately, this temporal organization makes it prohibitively expensive for GHB-based solutions to combine PC localization with address correlation, because linked list traversals are needed to find past occurrences of the triggering memory request (see Figure 4). Alternatively, we could imagine allocating a separate fixed-size GHB for each PC, but this solution has issues similar to those of Markov tables: Short streams would waste space, while long streams would require us to chain together multiple GHBs and to follow multiple pointer dereferences to traverse the entire chain. As a result, GHB-based designs forsake either PC localization [43] or address correlation [27, 28, 12], sacrificing significant coverage for design simplicity.

The ISB's structural address space allows the correlation information to be organized both spatially and temporally to provide the advantages of both approaches: (1) Temporal streams can be efficiently prefetched; (2) the ISB can combine PC localization and address correlation; and (3) the ISB can cache correlation information for just the TLB-resident pages and synch the management of this correlation information with TLB misses.

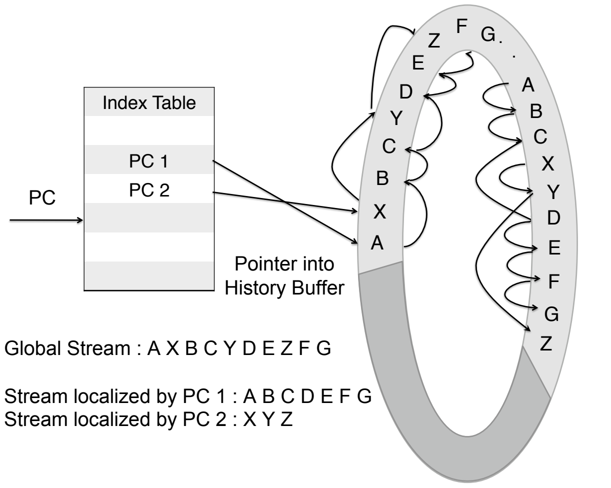

Index Table
PC 1
PC 2
PC
Pointer into History Buffer
A X B C Y D E Z F G .
. A B C X Y D E F G Z
Global Stream : A X B C Y D E Z F G
Stream localized by PC 1 : A B C D E F G
Stream localized by PC 2 : X Y Z

Figure 4: PC-localized address correlation using the GHB.

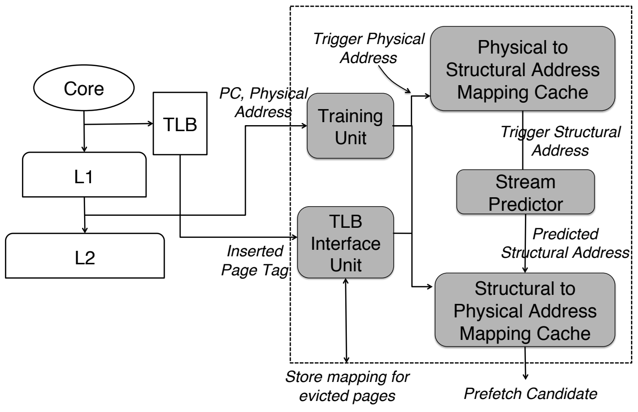

Core
TLB
L1
L2
PC, Physical Address
Training Unit
TLB Interface Unit
Inserted Page Tag
Trigger Physical Address
Physical to Structural Address Mapping Cache
Trigger Structural Address
Stream Predictor
Predicted Structural Address
Structural to Physical Address Mapping Cache
Store mapping for evicted pages
Prefetch Candidate

Figure 5: Block diagram of the Irregular Stream Buffer.

4. OUR SOLUTION

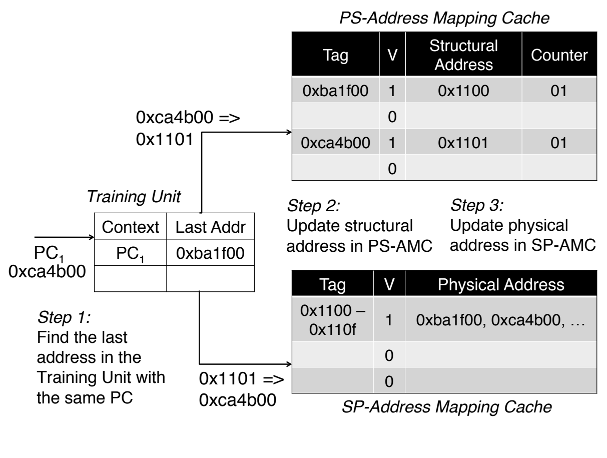
This section describes our solution by first summarizing the overall ISB design and then providing technical details.

The ISB prediction mechanism mimics the simplicity of stream buffers. Just as stream buffers predict regular memory access patterns, the ISB predicts sequences of memory addresses that are consecutive in the structural address space. Thus, the ISB's prediction step is much simpler than that of other correlation-based prefetchers, which can involve traversals through the GHB.

To enable these predictions, the ISB training mechanism translates correlated physical memory addresses to consecutive structural addresses. The mapping from the physical address space to the structural address space is cached on-chip only for pages that are resident in the TLB, and the prefetcher updates these caches during long latency TLB misses to effectively hide the latency of accessing off-chip meta-data.

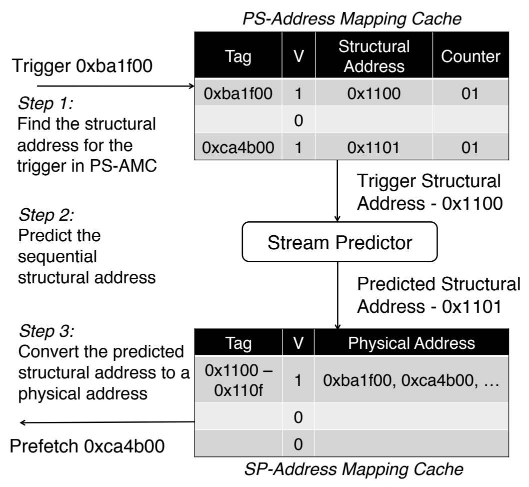
4.1 ISB Components

The key components of the ISB are shown in Figure 5 and are described below.



PS-Address Mapping Cache
Tag | V | Structural Address | Counter
0xba1f00 | 1 | 0x1100 | 01
0
0xca4b00 | 1 | 0x1101 | 01
0
0xca4b00 => 0x1101
Training Unit
Context | Last Addr
PC1 | 0xba1f00
PC1 0xca4b00
Step 1: Find the last address in the Training Unit with the same PC
Step 2: Update structural address in PS-AMC
Step 3: Update physical address in SP-AMC
Tag | V | Physical Address
0x1100 – 0x110f | 1 | 0xba1f00, 0xca4b00, …
0
0
0x1101 => 0xca4b00
SP-Address Mapping Cache

Figure 6: ISB training mechanism.



PS-Address Mapping Cache
Trigger 0xba1f00
Tag | V | Structural Address | Counter
0xba1f00 | 1 | 0x1100 | 01
0
0xca4b00 | 1 | 0x1101 | 01
Step 1: Find the structural address for the trigger in PS-AMC
Trigger Structural Address - 0x1100
Stream Predictor
Step 2: Predict the sequential structural address
Predicted Structural Address - 0x1101
Step 3: Convert the predicted structural address to a physical address
Tag | V | Physical Address
0x1100 – 0x110f | 1 | 0xba1f00, 0xca4b00, …
0
0
Prefetch 0xca4b00
SP-Address Mapping Cache

Figure 7: ISB prediction mechanism.

Training Unit.

The training unit takes as input the load PC and the load address, and it maintains the last observed address in each PC-localized stream. It learns pairs of correlated physical addresses and maps these to consecutive structural addresses.

Address Mapping Caches (AMCs).

The ISB uses two on-chip caches to maintain the mapping between physical and structural addresses. The Physical-to-Structural AMC (PS-AMC) stores the mapping from the physical address space to the structural address space; it is indexed by physical addresses. The Structural-to-Physical AMC (SP-AMC) stores the inverse mapping as the PS-AMC and is indexed by structural addresses. While the SP-AMC is not strictly necessary, it enables efficient temporal stream prediction because each cache line in the SP-AMC can yield in a single lookup 16 prefetch candidates from the current temporal stream.

Stream Predictor.

The stream predictor manages streams in the structural address space. It is analogous to stream buffers that are used for prefetching regular memory accesses [24]. Each entry in the stream predictor stores the starting structural address of the temporal stream, a counter to indicate the length of the observed stream, and a counter to indicate the current prefetch look-ahead distance. Like a stream buffer, the stream predictor can be configured for various prefetch degrees and look-ahead distances.

4.2 Prefetcher Operation

We now discuss in more detail each of the ISB's three key functions—training, prediction, and TLB eviction.

Training.

The training process assigns consecutive structural addresses to the correlated physical addresses that are observed by the training unit. When a correlated pair (A,B) is observed, the PS-AMC is queried to see if A and B have previously been assigned structural addresses. If A and B already have consecutive structural addresses, the ISB increments the confidence counter for B's entry in the PS-AMC. If instead A and B have previously been assigned non-consecutive structural addresses, then the confidence in B's mapping is decremented. When the confidence counter hits 0, B is assigned the structural address following A's structural address. If there is no existing mapping for A in the PS-AMC, the ISB generates a new structural address for A and assigns B the subsequent structural address.

Structural addresses are allocated in fixed size chunks of size c to facilitate temporal streams. To keep track of unassigned structural addresses, the ISB maintains a 64-bit counter and increments it by c after every new allocation. Structural addresses are not de-allocated for future reuse, because the 32-bit structural address space is large enough to map 256 GB of physical address space. Fixed size allocation allows every temporal stream to grow up to length c in the structural address space. Temporal streams of length greater than c must request a new allocation in the structural address space for every $(c + 1)$th element. Shorter temporal streams, on the other hand, can lead to internal fragmentation of the structural address space. Our experiments show that $c = 256$ is a good choice that supports efficient temporal stream prediction without suffering from excessive internal fragmentation.

As an example of the training process, consider a localized stream as shown in Figure 6, where the Training Unit's last observed address is `0xba1f00`, whose structural address is `0x1100`. When the Training Unit receives the physical address `0xca4b00` in the same localized stream, it performs three steps. (1) It assigns `0xca4b00` the structural address following `0xba1f00`'s structural address, namely `0x1101`. (2) It updates the PS-AMC entry indexed by physical address `0xca4b00`, and it updates the SP-AMC entry indexed by structural address `0x1101`. (3) It changes the last observed address in the Training Unit to `0xca4b00`.

Prediction.

One goal of the ISB design is to keep the prediction process (Figure 7) as simple as possible. There are three steps. (1) Each L2 cache access becomes a *trigger address* for the prefetcher, causing the PS-AMC to retrieve the trigger address' structural address. In our above example, an access to physical address `0xba1f00` is translated to structural address

0x1100 by the PS-AMC. (2) The Stream Predictor predicts the next consecutive structural addresses to prefetch, which for degree 1 prefetching is 0x1101. For degree k prefetching, the prediction would include the next k structural addresses, which in this example would be 0x1102, 0x1103, 0x1104 and so forth. (3) The SP-AMC retrieves the physical addresses for each of the predicted structural addresses to prefetch. So, 0x1101 is mapped back to 0xca4b00, and a prefetch request is initiated for this physical address. This mechanism, which consists of two cache lookups, can be used to predict temporal streams efficiently since a single cache line in the SP-AMC contains the translation for 16 consecutive structural addresses.

TLB evictions.

During a TLB eviction, the ISB writes to DRAM any modified mappings for the evicted page, and it fetches from DRAM the structural mapping for the incoming page. The writeback mechanism invalidates the PS-AMC cache lines corresponding to the evicted page, and it initiates a write to memory if the dirty bit is set. Since the PS-AMC and SP-AMC store redundant information, the contents of the SP-AMC need not be written to memory on an eviction. The fetch mechanism initiates a read request for the structural mapping of the newly inserted page and updates both caches appropriately. Since a TLB miss is a long latency operation involving multiple cache and DRAM accesses, these main memory reads and writes are off the critical path and small enough to not interfere with the core-initiated memory requests. In particular, the ISB is able to overlap its off-chip access with the latency of a TLB miss.

4.3 Details of the Address Mapping Caches

To optimize the use of on-chip storage, the ISB uses a compressed representation of the physical/structural addresses in its AMCs. Because the AMCs hold only TLB-resident cache lines, the ISB can use the 7-bit index in the TLB to replace the high order 42 bits of the physical address. The SP-AMC can then store the 13-bit physical address formed by concatenating the 7-bit physical page index and the 6-bit offset in the physical page. Similarly, the PS-AMC can store the 13-bit structural address formed by concatenating the 7-bit structural page index and the 6-bit offset in the structural page. The structural page indices are maintained in a CAM which is updated on a TLB miss or on a new allocation in the structural address space. This compressed representation is used for all internal ISB operations, such as training and prediction. The 13-bit physical address is expanded to the original 64-bit address only when the ISB schedules a prefetch request, and the 13-bit structural page index needs to be expanded only when the off-chip structural mapping is updated on a TLB eviction.

The PS-AMC and SP-AMC are organized as set-associative caches with 32-byte cache lines. Each cache line in the PS-AMC contains the structural mapping for 16 consecutive physical addresses, with each mapping using 2 bytes to store a 13-bit structural address, a 2-bit confidence counter, and a valid bit. Similarly, each 32-byte cache line in the SP-AMC contains the physical address maps for 16 consecutive structural addresses. If we were to fully provision each cache to map all pages in a 128 entry data TLB, the SP-AMC and PS-AMC would store 8K mapping entries, requiring a total of 32 KB of storage. However, our evaluation shows that in a hybrid setting, provisioning for more than 2K entries has diminishing performance gains and that an 8 KB ISB provides an attractive trade-off between on-chip storage and performance.

Core	Out-of-order, 4 Int/2 Mem/4 FP Func Units, 128-entry ROB, 4-wide dispatch/commit, 80-entry LSQ, 256 physical registers
Front-End	4-wide Fetch, 32-entry Fetch Queue, 4K entry BTB, 1K entry RAS, Hybrid Two-Level Branch Predictor, 128 KB 8-way L1 I-Cache
L1	64 KB 8-way, 2-cycle latency
L2	2 MB 8-way, 18-cycle latency, 64 MSHRs
DTLB	128 entries per core
DRAM	50 ns latency
Two-core	Private L1 cache, 4 MB shared L2 cache
Four-core	Private L1 cache, 8 MB shared L2 cache

Table 1: Baseline configuration.

4.4 Off-chip Storage

To organize the ISB's off-chip meta-data, we use the Predictor Virtualization framework proposed by Burcea at al [2]. In particular, we use a dedicated region of physical memory to maintain the mapping from the physical to the structural address space, which precludes the need for virtual address translation or OS intervention for meta-data accesses.

For our workloads, it suffices to reserve for the ISB 8 MB of off-chip storage. By contrast, the GHB-based prefetchers that we simulate require up to 128 MB of off-chip storage for the same workloads. This discrepancy in off-chip storage arises because the ISB's meta-data grows with the application's memory footprint, whereas the GHB's meta-data is proportional to the number of memory requests made by the application.

5. EVALUATION

5.1 Methodology

We evaluate the ISB using Marss, a cycle accurate full-system x86 simulator [30], to model single-core, 2-core, and 4-core systems (see Table 1 for details). Our simulation infrastructure faithfully models cache queue contention, port conflicts and memory traffic due to prefetch requests. Our TLB simulation allows page entries to be cached in the last-level cache and accurately accounts for the latency of TLB misses. For single-core simulations, we disable timer interrupts. For multi-core simulations, we account for the occasional variation in IPC due to kernel interrupts by taking the median of five runs.

Benchmarks.

Because we are interested in irregular memory accesses, our evaluation uses the memory-intensive benchmarks from SPECint2006, which generally use irregular pointer-based data structures. We consider a benchmark to be memory-intensive if it has a CPI > 2 and an L2 miss rate $> 50\%$, according to Jaleel's careful characterization of SPEC2006 [21]. We also use two benchmarks from SPECfp2006, soplex and sphinx3, which contain a mix of both regular and irregular memory accesses. The benchmarks are compiled using gcc-4.2 with the -O2 option. We compile the benchmarks disabling SSE3/4 instructions because our simulator lacks SSE support. All benchmarks are run using the reference input set. We use the SimPoint sampling methodology, generating for each benchmark multiple SimPoints of 250 million instructions to accurately capture all phases of the benchmark. The SimPoints are generated using the SimPoint Tool [31, 16]. We choose a SimPoint length of 250 million instruction because it is large enough to capture long-range behavior, including multiple L2 cache misses on a given address.

Multi-programmed Workloads.
We simulate multi-programmed workloads by choosing different combinations of our existing benchmarks, simulating two benchmarks at a time on our 2-core configuration and four benchmarks at a time on our 4-core configuration. For each benchmark, we fast-forward to a single SimPoint of 250 million instructions. We then simulate the simultaneous execution of the SimPoint regions for the particular benchmark combinations.

Evaluated Prefetchers.
In addition to the ISB, we simulate four other prefetchers that target irregular memory accesses.

First, we simulate Idealized STMS, an idealized version of Wenisch, et al's Sampled Temporal Memory Streaming (STMS) prefetcher [42]). Rather than implement all of the STMS optimizations, we simply simulate an idealized G/AC prefetcher,[2] which represents an upper bound on STMS' performance. In particular, the performance of STMS has been shown to approach that of an idealized G/AC prefetcher for long streams [42]. Idealized STMS uses a 64 MB GHB with 8M index table entries and optimistically assumes that its accesses to the DRAM-resident GHB are free in terms of access latency, DRAM traffic, and memory controller contention. In terms of accuracy and coverage, Idealized STMS primarily differs from STMS in two ways. First, Idealized STMS performs well for short streams, while STMS does not. Second, Idealized STMS trains on the L2 access stream instead of the L2 miss stream.

Second, we simulate an idealized PC/AC prefetcher that represents an upper bound for what any GHB-based prefetcher could achieve, because it uses the combination of PC localization and address correlation. This idealized PC/AC prefetcher is completely unrealistic. In addition to the optimistic assumptions that we make for Idealized STMS, we give PC/AC—when possible—an infinite number of linked list traversals per prediction, which is essential to its speedup. For example, when limited to 10,000 linked list traversals per prediction, coverage falls by 50%. However, for mcf and libquantum, we limit the linked list traversals per prediction to 10,000 to allow our simulations to finish within 3 days.

Third, we simulate Nesbit and Smith's PC/DC prefetcher, which which learns the deltas, or differences, between consecutive memory addresses [28]. Delta correlation allows PC/DC to store all meta-data on chip, so this prefetcher can realistically train on the L2 access stream. We tune PC/DC using all of the optimizations described by Dimitrov and Zhou [13], who submitted the best GHB-based prefetcher in the 2009 Data Prefetching Competition. As with Dimitrov and Zhou's design, our PC/DC prefetcher uses the GHB to exploit delta correlation in both the local and global streams.

Fourth, we simulate the Spatial Memory Streaming (SMS) prefetcher [39], the best known prefetcher that purely exploits spatial locality. The SMS prefetcher realistically trains on the L2 access stream.

We also study the benefit of using irregular prefetchers in conjunction with regular stride prefetchers. For this study, we use Ishii et al.'s AMPM prefetcher [20], the winner of the 2009 Data Prefetching Competition. AMPM identifies hot zones in memory and stores a bitmap to infer strided patterns in the access stream. AMPM is extremely effective and aggressive because it can detect regular memory accesses independent of the order in which they are observed. We give the AMPM 4 KB of storage and tune it by adjusting its threshold and associativity parameters to produce the best coverage.

For the hybrid experiments, we use an 8 KB ISB, because a 32 KB ISB provides only a small speedup improvement. For the non-hybrid experiments, we use a 32 KB ISB, which contains a 16 KB direct-mapped PS-AMC with 32-byte cache lines, and which uses an 8-way set-associative SP-AMC with 32-byte cache lines.

5.2 Single-Core Results

Figure 8 compares the speedup, accuracy, and coverage of our five prefetchers on a single core. We see that the two PC/AC-based prefetchers—ISB and idealized PC/AC—achieve significantly better speedup and accuracy than the others. In particular, the speedups over a baseline with no prefetching are 26.9% for idealized PC/AC, 23.1% for ISB, 14.1% for PC/DC, 9.97% for Idealized STMS, and 6.9% for SMS. Idealized PC/AC and ISB also see impressive accuracies of 88.0% and 93.7%, respectively, while the other irregular prefetchers observe less than 65% accuracy on average. These results indicate that PC-localized address correlation is superior to the other techniques—global address correlation (STMS), delta correlation (PC/DC), and spatial footprints (SMS)—for prefetching irregular accesses.

These graphs also show that a practically provisioned ISB approaches the performance of an idealized PC/AC. By contrast, STMS, the previous state-of-the-art in correlation prefetching [42], approaches the performance of idealized G/AC. Figure 8 shows two anomalies, namely, that the ISB performs better than the idealized PC/AC on mcf and libquantum. Idealized PC/AC performs poorly on these two benchmarks because it is prohibitively expensive to completely idealize the PC/AC prefetcher for these two benchmarks due to their large memory footprint, so for these two benchmarks, we limited the number of linked list iterations to 10,000 and used the largest possible GHB that allowed the simulations to complete in 3 days.

If we make Idealized PC/AC a bit more realistic by letting it train on the L2 miss stream instead of the L2 access stream, its speedup falls to 10.4% and its accuracy falls to 86.3%. Similarly, if Idealized STMS trains on the L2 miss stream, its speedup falls to 8.3% and its accuracy to 58.6%.

[2] Using Nesbit and Smith's terminology [28], in which the name before the slash describes the reference scheme and the name after the slash describes the type of correlation that is used, a G/AC prefetcher trains on a Global reference stream and uses Address Correlation.

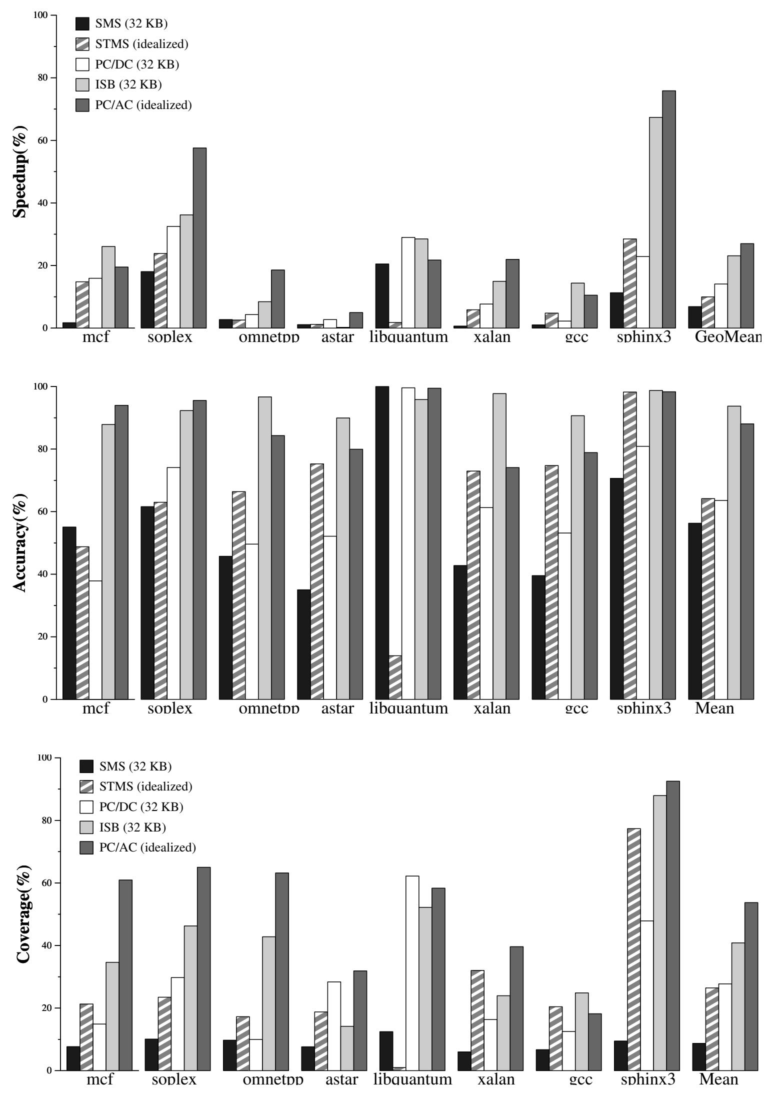


Figure 8: Comparison of irregular prefetchers on single core (degree 1)

Finally, we note that the ISB sees speedup of just 2.3% on the remaining SPEC FP benchmarks, because the ISB cannot predict compulsory misses, whereas many stride prefetchers can. The ISB does not slow down any of the benchmarks.

5.3 Memory Traffic Overhead

The ISB's memory traffic overhead approaches that of prefetchers, such as SMS and PC/DC. that store all of their meta-data on chip. In particular, the ISB incurs an average of 14.7% memory traffic overhead, while Dimitrov and Zhou's PC/DC prefetcher [13] incurs 12.6% overhead and SMS just 10.5% overhead. The highly accurate ISB incurs just 6.3% overhead due to useless prefetches. The ISB accesses off-chip meta-data only during a TLB miss, reading at most 256 bytes of mapping information per page. Assuming a bus width of 64-bytes with DDR2, this information can be fetched from DRAM in four requests. Since not all cache lines in a page are necessarily mapped, the actual traffic per page can vary from one to four requests. As seen in Table 2, the ISB's access to off-chip correlation data increases memory traffic by an average of 8.4%. With DDR3's 128-byte bus width, the traffic would be reduced to 4.2% because the information for the entire page could be fetched in a single DRAM request. By contrast, the STMS prefetcher incurs about 35% overhead due to meta-data access [42].

5.4 Degree Evaluation

Figures 9 and 10 show how the speedup and accuracy of four prefetchers—ISB, PC/DC SMS, and Idealized STMS—vary as the prefetch degree is increased from 1 to 8, revealing several trends:

- The ISB performs well as the degree increases: With degree 8, its speedup rises from 23.1% to 38.6%, and its accuracy decreases by just 3.8%.

- PC/DC has the most severe tradeoff between speedup and accuracy: With degree 8, its speedup almost doubles to 28.8%, but its accuracy falls down to 46.8%, which is the worst among all prefetchers.

	mcf	soplex	omnetpp	astar	libquantum	xalan	gcc	sphinx3	Mean
Useless prefetches	2.8%	5.3%	9.5%	5%	0.05%	5.2%	16%	4.1%	6.3%
Meta-data traffic	5.7%	3.8%	5.7%	12%	1.6%	12.6%	11.3%	3.9%	8.4%

Table 2: Memory traffic overhead of the ISB with DDR2.

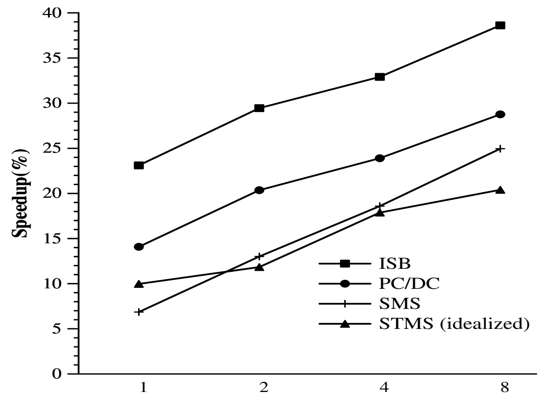


Figure 9: Impact of prefetch degree on speedup.

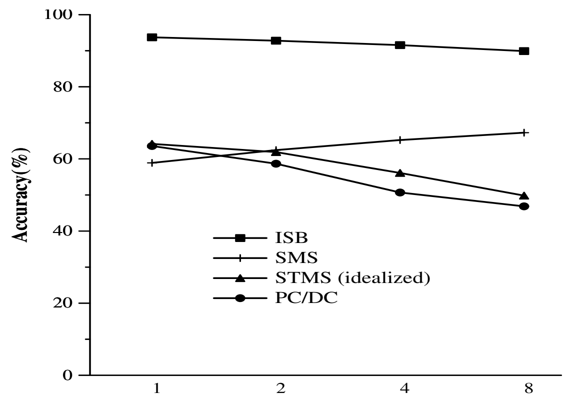


Figure 10: Impact of prefetch degree on accuracy.

- By contrast, the SMS prefetcher has the best trade-off between speedup and accuracy, as it improves in both speedup *and* accuracy as the degree is increased, indicating that prefetches from higher density spatial regions are more accurate, but even for degree 8, the ISB exhibits significantly better speedup and accuracy than SMS.

- Finally, except at degree 1, Idealized STMS has the worst performance of all of the prefetchers, and its accuracy curve closely matches that of PC/DC.

5.5 Hybrid Design with AMPM

Vendors that implement an irregular prefetcher will undoubtedly also implement a regular prefetcher, so we now consider hybrid designs that combine an irregular prefetcher with an AMPM stride prefetcher. Here, we only consider the three practical prefetchers, namely, ISB, PC/DC, and SMS.

When combined with a regular prefetcher, the ISB is much less sensitive to the size of the AMC. As a result, in a hybrid setting with AMPM, an ISB with 8 KB of storage sees

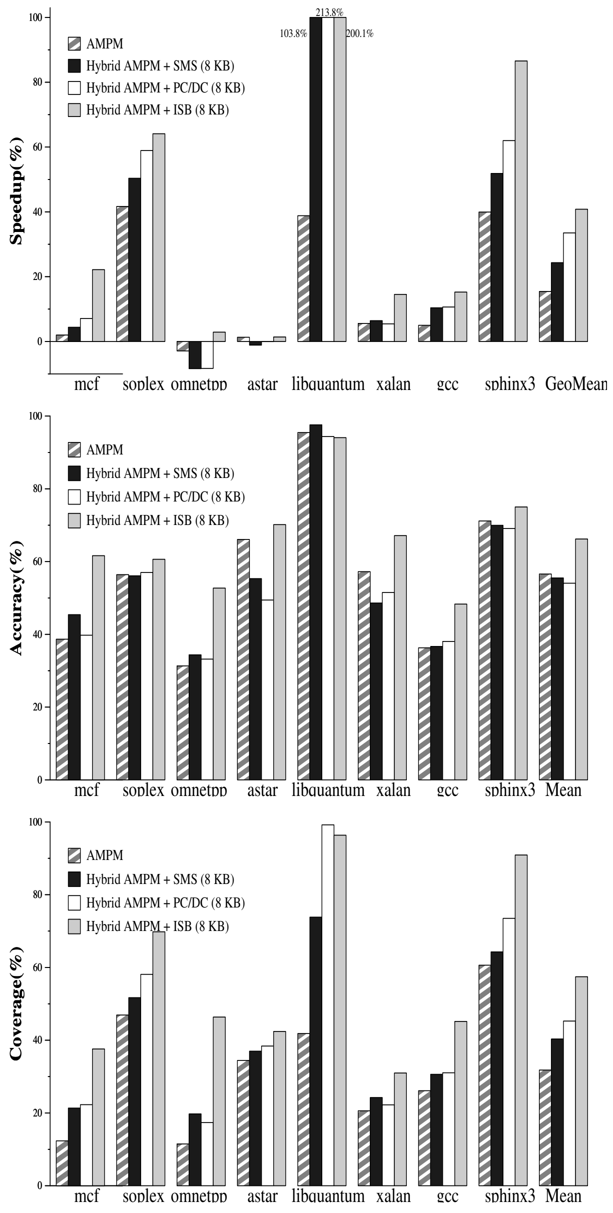


Figure 11: Comparison of hybrid prefetchers

speedup of 40.8%, whereas an ISB with 32 KB of storage sees an additional speedup of only 6.3%. This behavior can be understood by observing that in our workloads, phases of regular and irregular accesses see little overlap and that the ISB requires large on-chip memory to prefetch long regular streams. In a hybrid setting, 8 KB is sufficient for the ISB to prefetch the irregular phases, while AMPM can prefetch the regular phases.

Figure 11 compares AMPM against hybrid prefetchers that combine AMPM with an 8 KB ISB, an 8K SMS, and an 8K PC/DC, respectively. The AMPM + SMS hybrid achieves a 24.3% speedup over a baseline with no prefetching, the AMPM + PC/DC achieves a 33.5% speedup, while AMPM alone achieves 15.4% speedup. The AMPM + ISB hybrid achieves a speedup of 40.8% over a baseline with no prefetching, which is an improvement of 25.4% over AMPM. The coverage graph shows that SMS achieves just 4.5% coverage and PC/DC only 9.4% additional coverage over AMPM, while ISB achieves an extra 21.6% coverage over AMPM. A closer inspection of Figure 11 indicates several other key points.

1. For libquantum, the AMPM + PC/DC hybrid outperforms the AMPM + ISB hybrid because the ISB is not capable of prefetching cold misses, while PC/DC is.

2. The three benchmarks that contain both regular and irregular accesses—soplex, sphinx, and gcc—see good speedups over AMPM with all hybrids.

3. For four of the benchmarks—mcf, omnetpp, astar, and xalan—only the AMPM + ISB hybrid achieves a significant improvement over AMPM. These benchmarks are dominated by pointer-based accesses to a graph, a graph, a tree, and a tree, respectively. This indicates that delta correlation and spatial footprints are not very effective for irregular accesses. Moreover, poor coverage combined with poor accuracy causes the AMPM + SMS hybrid and the AMPM + PC/DC hybrid to slow down omnetpp and astar.

4. The AMPM + ISB hybrid has the highest accuracy among the hybrids at 66.2%. This accuracy is significantly lower than the ISB's accuracy of 93.7% because of AMPM's poor accuracy of 56.6%. For chips with a larger number of cores, a less aggressive stride prefetcher than AMPM would probably be wise.

5.6 Multi-Core Results

Figure 12 compares the ISB with SMS, PC/DC, and Idealized STMS on a multi-core system using multi-programmed workloads as described in Section 5.1. We see that the ISB outperforms the three prefetchers on both the 2-core and 4-core machines. On the 2-core machine, the ISB sees a speedup of 23.69%, whereas SMS, PC/DC and Idealized STMS see average speedups of 11.9%, 13.9% and 15.7%, respectively. The average speedup for all prefetchers is lower on the 4-core machine, with the ISB observing a 10.3% speedup, and with SMS, PC/DC, and Idealized STMS achieving 3.7%, 5.8% and 6.5% speedup, respectively. The ISB's accuracy is consistently above 95% for both configurations, which makes it attractive in a multi-core setting, since useless prefetches increase both memory traffic and cache pollution. As the number of cores increase, prefetching accuracy can have a significant bearing on system performance. For example, Ebrahimi, et al. show that in a multi-core environment with 4 cores, any prefetcher whose accuracy is below 40% needs to be throttled down to preserve overall system performance [14].

Figure 13 evaluates hybrid prefetchers by combining AMPM with the ISB, SMS and PC/DC. In a hybrid setting, only the ISB is able to significantly outperform AMPM on both machines, which supports our claim that the ISB is more effective at irregular prefetching than PC/DC and SMS. The AMPM + ISB hybrid observes speedups of 28.6% and 13% on 2-core and 4-core machines, respectively, which is much less than the 40.8% speedup that it achieved on the single-core machine. This decline can be attributed to AMPM, which generates considerable useless traffic due to its poor accuracy.

5.7 Power Evaluation

While training on the L2 reference stream provides significant coverage and accuracy benefits, its increased activity increases the prefetcher's power consumption. We thus evaluate the power and energy consumption of the ISB by comparing them against that of the GHB-based PC/DC prefetcher that trains on the L2 miss stream. This discussion will not consider the power impact of useless prefetches on cache and memory subsystem behavior. We use CACTI [35] to estimate the energy consumed by the prefetching hardware per read/write operation, and we then multiply that cost by the activity counters of the prefetching hardware. We find that the ISB consumes 0.77 times the energy of PC/DC but 1.07 times the power. The increase in average power consumption can be attributed to the faster execution time with the ISB. The ISB generates more activity by training on the L2 access stream, but uses a simple training and prediction logic. By contrast, PC/DC consumes far more energy per input due to its linked list traversals through the GHB, so for the same energy budget, the ISB is able to use localization and exploit the information available in the entire L2 access stream with minimal power overhead.

6. CONCLUSIONS

In this paper, we have introduced the Irregular Stream Buffer, which represents a significant milestone in the long quest to build prefetchers for irregular memory accesses: The ISB is the first practical prefetcher that combines address correlation with PC localization. While the previous state-of-the-art in temporal stream prefetching, STMS, approaches the behavior of an idealized G/AC prefetcher for long streams, the ISB approaches the superior coverage and accuracy of an idealized PC/AC prefetcher for all streams.

The key idea behind the ISB is an extra level of indirection that translates correlated physical addresses to consecutive addresses in a new structural address space. Thus, in the structural address space, the elements of a temporal stream appear in sequential order, which greatly simplifies prediction.

The structural address space provides three important benefits.

1. It allows the ISB to manage meta-data efficiently by caching TLB-resident meta-data on chip and synchronizing the contents of this cache with TLB misses.

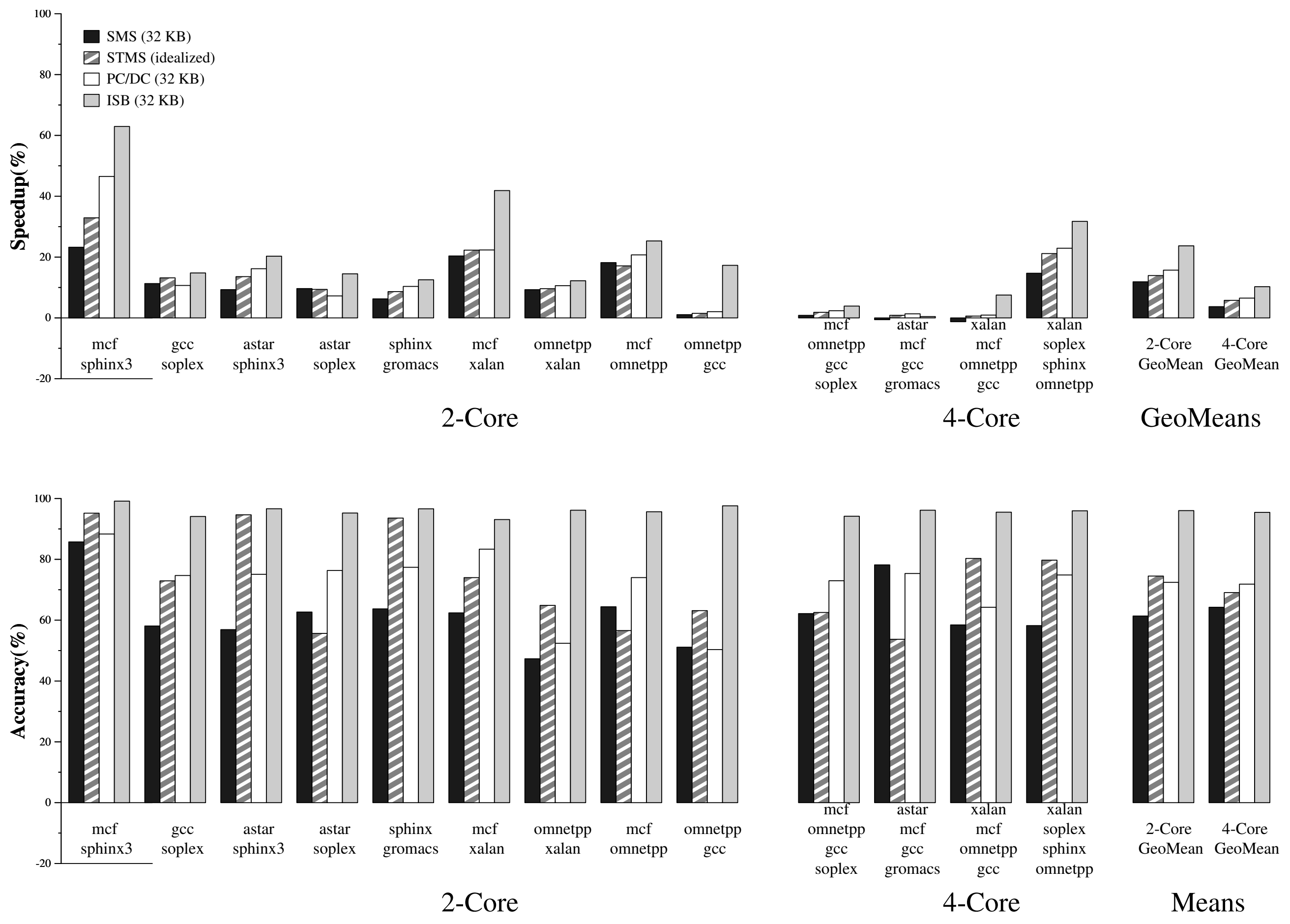


Figure 12: Comparison of irregular prefetchers on 2-core (left) and 4-core (right) systems.

The result is just 8.4% memory traffic overhead for accessing off-chip meta-data, significantly lower than the overheads reported for other address correlation-based prefetchers, such as STMS [42], which itself represented an order of magnitude improvement over its predecessors [8].

2. It improves coverage and accuracy by supporting the combination of PC localization and address correlation. For example, on a single core, an idealized PC/AC prefetcher obtains 26.9% average speedup and 88% accuracy, compared with 14.1% speedup and 65% accuracy for PC/DC; an idealized G/AC prefetcher (ie, Idealized STMS) sees 9.97% speedup and 65% accuracy.

3. Our caching scheme improves coverage and accuracy by allowing the ISB to train on the LLC reference stream instead of the LLC miss stream, which in our experiments more than doubles the observed speedup. For example, the idealized PC/AC prefetcher sees 26.9% speedup when trained the L2 access stream, as opposed to just 10.4% when trained on the L2 miss stream.

Looking to the future, we plan to evaluate the ISB on commercial workloads. We expect that the ISB will perform well on these workloads, because unlike the GHB, the ISB's on-chip storage and memory traffic overhead depend only on the size of the TLB, not the application's memory footprint. To extend the ISB's benefits to TLBs with large pages, including superpages, we plan to explore a two-level ISB design that can synchronize with pages of any size without undermining the ISB's small on-chip budget. We also plan to evaluate the use of ISB as the temporal component of spatial-temporal prefetchers similar to Somogyi, et al.'s STeMS prefetcher [38]. More broadly, we believe that the use of a linearized structural address space can be used to drive other micro-architectural optimizations for irregular programs.

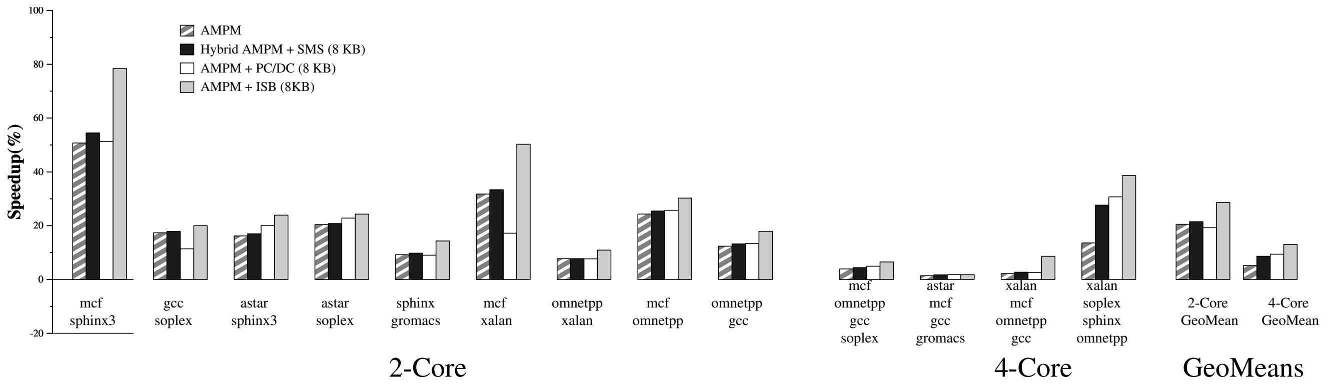
Acknowledgments.

We thank Curtis Dunham, Ibrahim Hur, Don Fussell, and Daniel Jimenez for their valuable comments on early drafts of this paper. This work was funded in part by NSF grant CNS-1138506.

7. REFERENCES

[1] J.-L. Baer and T.-F. Chen. Effective hardware-based data prefetching for high-performance processors. *IEEE Transactions on Computers*, 44(5):609–623, May 1995.

[2] I. Burcea, S. Somogyi, A. Moshovos, and B. Falsafi. Predictor virtualization. In *Proceedings of the 13th*



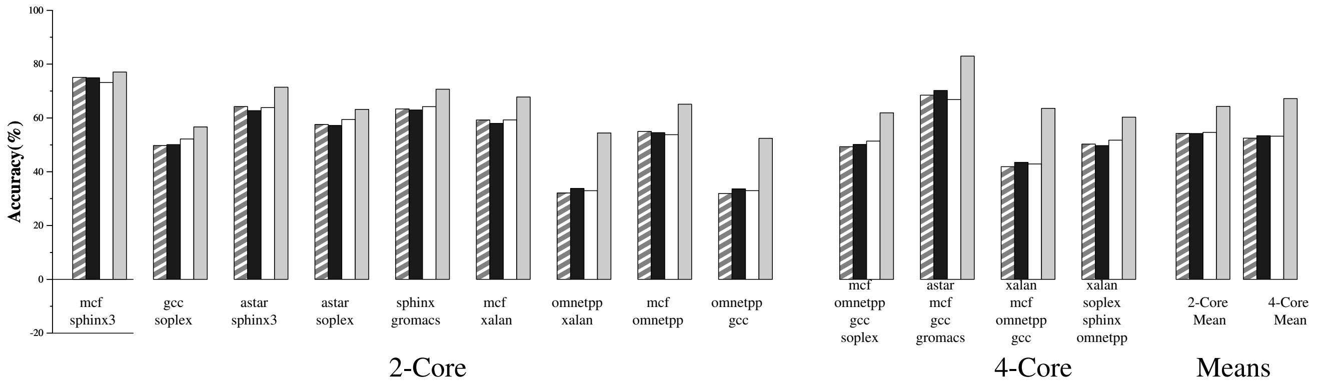


Figure 13: Comparison of hybrid prefetchers on 2-core (left) and 4-core (right) systems.

international conference on Architectural support for programming languages and operating systems, ASPLOS XIII, pages 157–167. ACM, 2008.

[3] D. Burger, T. R. Puzak, W.-F. Lin, and S. K. Reinhardt. Filtering superfluous prefetches using density vectors. In *ICCD '01: Proceedings of the International Conference on Computer Design: VLSI in Computers & Processors*, pages 124–133, 2001.

[4] J. B. Carter, W. C. Hsieh, L. Stoller, M. R. Swanson, L. Zhang, E. Brunvand, A. Davis, C.-C. Kuo, R. Kuramkote, M. A. Parker, L. Schaelicke, and T. Tateyama. Impulse: Building a smarter memory controller. In *HPCA*, pages 70–79, 1999.

[5] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and complexity-effective spatial pattern prediction. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, HPCA '04, pages 276–288, 2004.

[6] T. M. Chilimbi. Efficient representations and abstractions for quantifying and exploiting data reference locality. In *PLDI*, pages 191–202, 2001.

[7] T. M. Chilimbi, M. D. Hill, and J. R. Larus. Cache-conscious structure layout. In *Proceedings of the ACM SIGPLAN 1999 conference on Programming Language Design and Implementation*, PLDI '99, pages 1–12, 1999.

[8] Y. Chou. Low-cost epoch-based correlation prefetching for commercial applications. In *MICRO*, pages 301–313, 2007.

[9] I.-H. Chung, C. Kim, H.-F. Wen, and G. Cong. Application data prefetching on the ibm blue gene/q supercomputer. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, 2012.

[10] 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 35, pages 62–73, 2002.

[11] R. Cooksey, S. Jourdan, and D. Grunwald. A stateless, content-directed data prefetching mechanism. *SIGARCH Computer Architecture News*, 30(5):279–290, October 2002.

[12] P. Diaz and M. Cintra. Stream chaining: exploiting multiple levels of correlation in data prefetching. In *ISCA*, pages 81–92, 2009.

[13] M. Dimitrov and H. Zhou. Combining local and global history for high performance data prefetching. In *Journal of Instruction-Level Parallelism Data Prefetching Championship*, volume 13, 2011.

[14] E. Ebrahimi, O. Mutlu, and Y. N. Patt. Techniques for bandwidth-efficient prefetching of linked data structures in hybrid prefetching systems. In *HPCA*, pages 7–17, 2009.

[15] K. I. Farkas, P. Chow, N. P. Jouppi, and Z. Vranesic. Memory-system design considerations for dynamically-scheduled processors. In *ISCA '97: Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 133–143,

1997.

[16] G. Hamerly, E. Perelman, J. Lau, and B. Calder. Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4):1–28, 2005.

[17] Z. Hu, M. Martonosi, and S. Kaxiras. TCP: tag correlating prefetchers. In *HPCA*, pages 317–326, 2003.

[18] X. Huang, S. M. Blackburn, K. S. McKinley, J. E. B. Moss, Z. Wang, and P. Cheng. The garbage collection advantage: improving program locality. In *Proceedings of the 19th annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '04, pages 69–80, 2004.

[19] I. Hur and C. Lin. Memory prefetching using adaptive stream detection. In *Proceedings of the 39th International Symposium on Microarchitecture*, pages 397–408, 2006.

[20] Y. Ishii, M. Inaba, and K. Hiraki. Access map pattern matching for high performance data cache prefetch. In *Journal of Instruction-Level Parallelism*, volume 13, pages 1–24, 2011.

[21] A. Jaleel. Memory characterization of workloads using instrumentation-driven simulation – a pin-based memory characterization of the SPEC CPU2000 and SPEC CPU2006 benchmark suites. Technical report, VSSAD Technical Report 2007, 2007.

[22] T. L. Johnson, M. C. Merten, and W.-M. W. Hwu. Run-time spatial locality detection and optimization. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 57–64, 1997.

[23] D. Joseph and D. Grunwald. Prefetching using markov predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 252–263, 1997.

[24] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *SIGARCH Computer Architecture News*, 18(3a):364–373, May 1990.

[25] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. *SIGARCH Computer Architecture News*, 26(3):357–368, April 1998.

[26] C.-K. Luk and T. C. Mowry. Compiler-based prefetching for recursive data structures. *SIGOPS Operating Systems Review*, 30(5):222–233, September 1996.

[27] K. J. Nesbit, A. S. Dhodapkar, and J. E. Smith. Ac/dc: An adaptive data cache prefetcher. In *IEEE PACT*, pages 135–145, 2004.

[28] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. *IEEE Micro*, 25(1):90–97, 2005.

[29] S. Palacharla and R. E. Kessler. Evaluating stream buffers as a secondary cache replacement. In *Proceedings of the International Symposium on Computer Architecture*, pages 24–33, April 1994.

[30] A. Patel, F. Afram, S. Chen, and K. Ghose. MARSSx86: A Full System Simulator for x86 CPUs. In *Design Automation Conference 2011 (DAC'11)*, 2011.

[31] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder. Using simpoint for accurate and efficient simulation. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 318–319, 2003.

[32] 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-VIII, pages 115–126, 1998.

[33] 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 '99, pages 111–121, 1999.

[34] S. Sair, T. Sherwood, and B. Calder. A decoupled predictor-directed stream prefetching architecture. *IEEE Transactions on Computers*, 52(3):260–276, March 2003.

[35] P. Shivakumar and N. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical report, Technical Report 2001/2, Compaq Computer Corporation, 2001.

[36] A. Smith. Sequential program prefetching in memory hierarchies. *IEEE Transactions on Computers*, 11(12):7–12, December 1978.

[37] 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*, pages 171–182, 2002.

[38] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi. Spatio-temporal memory streaming. In *ISCA*, pages 69–80, 2009.

[39] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos. Spatial memory streaming. In *ISCA '06: Proceedings of the 33rd Annual International Symposium on Computer Architecture*, pages 252–263, 2006.

[40] Z. Wang, D. Burger, K. S. McKinley, S. K. Reinhardt, and C. C. Weems. Guided region prefetching: a cooperative hardware/software approach. *SIGARCH Computer Architecture News*, 31(2):388–398, May 2003.

[41] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos. Temporal streams in commercial server applications. In *IISWC*, pages 99–108, 2008.

[42] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos. Practical off-chip meta-data for temporal memory streaming. In *HPCA*, pages 79–90, 2009.

[43] T. F. Wenisch, M. Ferdman, A. Ailamaki, B. Falsafi, and A. Moshovos. Making address-correlated prefetching practical. *IEEE Micro*, 30(1):50–59, 2010.

[44] T. F. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. Temporal streaming of shared memory. *SIGARCH Computer Architecture News*, 33(2):222–233, May 2005.