# Efficiently Prefetching Complex Address Patterns

Manjunath Shevgoor
University of Utah
Salt Lake City, UT, USA
shevgoor@cs.utah.edu

Sahil Koladiya
University of Utah
Salt Lake City, UT, USA
koladiya@cs.utah.edu

Rajeev Balasubramonian
University of Utah
Salt Lake City, UT, USA
rajeev@cs.utah.edu

Chris Wilkerson
Intel Labs
Hillsboro, OR, USA
chris.wilkerson@intel.com

Seth H Pugsley
Intel Labs
Hillsboro, OR, USA
seth.h.pugsley@intel.com

Zeshan Chishti
Intel Labs
Hillsboro, OR, USA
zeshan.a.chishti@intel.com

## ABSTRACT

Prior work in hardware prefetching has focused mostly on either predicting regular streams with uniform strides, or predicting irregular access patterns at the cost of large hardware structures. This paper introduces the Variable Length Delta Prefetcher (VLDP), which builds up delta histories between successive cache line misses within physical pages, and then uses these histories to predict the order of cache line misses in new pages. One of VLDP's distinguishing features is its use of multiple prediction tables, each of which stores predictions based on a different length of input history. For example, the first prediction table takes as input only the single most recent delta between cache misses within a page, and attempts to predict the next cache miss in that page. The second prediction table takes as input a sequence of the two most recent deltas between cache misses within a page, and also attempts to predict the next cache miss in that page, and so on with additional tables. Longer histories generally yield more accurate predictions, so VLDP prefers to make predictions based on the longest history table that has a matching entry.

Using a global history of patterns it has seen in the past, VLDP is able to issue prefetches without having to wait for additional per-page confirmation, and it is even able to prefetch patterns that show no repetition within a physical page. VLDP does not use the program counter (PC) to make its predictions, but our evaluation shows that it out-performs the highest-performing PC-based prefetcher by 7.1%, and the highest performing prefetcher that doesn't employ the PC by 5.8%.

## Categories and Subject Descriptors

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

*MICRO-48,* December 05-09, 2015, Waikiki, HI, USA
©2015 ACM ISBN 978-1-4503-4034-2/15/12 ...$15.00
DOI: http://dx.doi.org/10.1145/2830772.2830793

## Keywords

Prefetching

## 1. INTRODUCTION

Memory latency continues to be a significant bottleneck in today's processors. Prefetching is an effective approach to hide this latency. It is a well-studied, complex problem with several aspects that have first-order effects on performance. For example, where the prefetcher is located in the memory hierarchy will constrain the information available to it. Prefetching from DRAM into the lowest level of the cache, therefore, introduces a number of unique challenges.

First, in most high volume CPU designs, the program counter (PC) is unavailable at this level in the cache hierarchy. This can make PC-based patterns more difficult to detect. Second, a prefetcher located at the last level cache must deal with physical addresses directly without the benefit of a TLB or other page table information. This means that address patterns must be discovered using only sequences of physical addresses. Since virtual to physical page mapping is often arbitrary, easily predictable sequences of virtual addresses spread over different pages may not exhibit discernible patterns after translation to the physical address space. This presents a particularly challenging problem for prefetchers that rely on discovering common deltas between consecutively requested addresses and applying them to future requests. In light of these constraints, many modern prefetchers track addresses on a per physical page basis, discovering patterns and prefetching within multiple simultaneously tracked physical pages.

The first step to performing a successful prefetch is to predict an address that is likely to be requested in the near future. Increasing the number of prefetch requests typically leads to higher coverage, i.e., it increases the number of successfully prefetched lines. But this is often at the expense of reduced accuracy, i.e., it increases the number of superfluous prefetches. Conversely, accuracy can often be improved by limiting the prefetcher to very predictable, easy to analyze patterns. The tradeoff between accuracy and coverage is a fundamental property of prefetcher design. Increasing accuracy at the expense of coverage limits performance potential, while reducing accuracy for higher coverage wastes memory bandwidth, which can also limit performance potential. To improve coverage and accuracy simultaneously, prefetchers must be able to recognize complex patterns in the address stream.

**Multi-Delta Sequences:** One common approach to discovering patterns in a sequence of physical addresses is to isolate the addresses in different regions (often physical pages) and identify sequences of addresses in those regions with repeating deltas. For example, the delta +2 would be identified in the address sequence A, A+2, A+4 and used to prefetch A+6, A+8 and so on, depending on the degree. The drawback of this approach is that it can only identify patterns consisting of a single repeated delta.

Although the single repeated delta, +1 in particular, is common in a wide range of workloads, many workloads contain repeating multi-delta sequences, as in the SPEC CPU2006 workload LBM. An analysis of accesses to a single page in LBM yields the following sequence of addresses: (A, A-24, A+1, A-23, A+2, A-22, A+3). Extracting the deltas from this sequence of addresses shows that the delta sequence (-24, +25) occurs repeatedly. In fact, LBM contains many delta sequences, each of which cycles through multiple deltas. Five common delta sequences found in LBM are listed below:

1. (-24, +25), (-24, +25)...
2. (-24, -24, +49), (-24, -24, +49)...
3. (+3, +2), (+3, +2)...
4. (-2, +3, +4), (-2, +3, +4)...
5. (-1, +3, -1, +4), (-1, +3, -1, +4)...

Some multi-delta sequences can be expressed as multiple single delta sequences. For example, the sequence of addresses (A, A-24, A+1, A-23, A+2, A-22, A+3) can be expressed as two +1 deltas, one beginning with A and the other beginning with A-24. To discover this pattern, a prefetcher would need two key features. First, it would need the ability to track multiple streams within a physical page. Second, it would need the ability to compare the new access address with multiple prior addresses in a window, comparing A and A+1, for example, to identify the +1 stream starting with A. Commonly implemented prefetchers, such as the stream prefetcher, lack both of these capabilities and would therefore be unable to prefetch these address sequences. However, prior work has described more sophisticated prefetchers such as AMPM [1] that do support both of these capabilities.

The key disadvantage of algorithms like AMPM is training time. In the address sequence (A, A-24, A+1, A-23, A+2, A-22, A+3, A-21, A+4, A-20), both streams must be independently established and confirmed. This means that the earliest prefetch opportunities for AMPM would be A+3 and A-21. We therefore introduce the Variable Length Delta Prefetcher (VLDP), which is designed to efficiently predict multi-delta sequences. VLDP learns from multi-delta sequences by remembering previously occurring delta pairs.

In this instance, VLDP will remember that the delta +25 follows delta -24, and that this pattern repeats. Subsequent references to delta -24 will cause a prefetch to the address corresponding to delta +25, and vice versa. In the address sequence described above, the VDLP can lock on to the address sequence and make a prefetch request as early as A+2. Additionally, VLDP has the ability to learn these patterns from one physical page, and apply them in every new physical page it encounters without having to re-learn them.

For example, pattern 3 from above, consisting of a repeating multi-delta sequence of (+3, +2), actually occurs across many physical pages in LBM. After establishing the (+3, +2) pattern on an initial physical page, VLDP can generalize and predict that +2 follows a +3 delta on future pages as well. While VLDP does not save much time by remembering the simple (+3, +2) pattern between pages, VLDP is also able to learn patterns with very long sequences of deltas, including patterns that do not repeat within a given physical page.

In the SPEC CPU2006 workload milc, we see the pattern (A, A+1, A+10, A+2, A+3, A+11, A+12, A+4, A+5, A+6, A+13). This pattern never repeats within a physical page, but it is seen across many pages. VLDP can learn this from one page and apply it to others.

Non-uniform deltas within a page are caused when an application touches various elements in a large data structure. Somogyi et al. [6] describe a few example applications that exhibit this behavior. For example, they point out the non-uniform, but often repeating, deltas observed during binary search in a B-tree. Different delta patterns are produced if the traversal through the data structure has data-dependent control flow. For workloads like milc, we observed similar code patterns (large nested structures with many access functions).

The proposed VLDP design has the following features not found in other prefetchers. First, VLDP enables the prediction of complex multi-delta access patterns. Second, VLDP works on a per-page basis, and it can prefetch a different complex pattern for each page. Third, VLDP uses multiple global prediction tables that can learn common access patterns across many pages. Fourth, these prediction tables are indexed by varying lengths of delta histories, using the longest history match found in the prediction tables to make the most accurate prediction. This combination of features allows VLDP to outperform previously proposed regular data prefetchers like AMPM by 6% and Sandbox by 9% on average.

## 2. BACKGROUND

We evaluate VLDP by comparing it to 6 state-of-the-art regular prefetchers, detailed below. The first four prefetchers, like VLDP, do not utilize the program counter (PC), while the last two do use the PC.

**Feedback Directed Prefetching** (FDP) [2] begins with a stream prefetcher, and then adds feedback mechanisms to control how aggressive it is. The stream prefetcher works by observing several memory accesses to a page of memory that are close to each other, and then determines a direction for prefetching. Each subsequent access to that page along that stream will cause more cache lines to be prefetched. FDP measures prefetcher accuracy, lateness, and cache pollution caused by prefetching, and then uses those metrics to control prefetch degree (number of items prefetched), and prefetch distance (how far ahead of the demand stream to prefetch).

Unlike VLDP, FDP must detect a stream on each individual page before prefetching. Outside of tuning prefetcher aggressiveness, there is nothing that can be learned from one page and applied to another page. Once a stream is detected, all cache lines in the stream are prefetched sequentially, with no gaps in the pattern, possibly leading to excessive prefetch.

**Access Map Pattern Matching** (AMPM) [1] works by maintaining arrays of 2-bit values, representing cache lines, for large regions of memory. It tracks the status of each cache line within the region (untouched, demand accessed, prefetched), and then analyzes the patterns in the arrays to generate prefetch addresses. The access map for a region starts with all 2-bit values zeroed out, and then marks each line that has been demand accessed. On each cache access to address A, the determination for whether or not to prefetch address A+N is made by checking the access map to see if A-N and A-2N have both been accessed before. This is done for a wide variety of values N, both positive and negative, for each cache access. Preference is given to prefetch values closer to A, and there are also feedback mechanisms to control prefetcher aggressiveness.

Unlike VLDP, AMPM only looks for simple repeating stride patterns. It requires 3 accesses along a stride pattern before prefetching begins. It cannot prefetch non-repeating patterns. Also, AMPM has to warm up each region independently of all other regions, just as FDP does. Finally, observations made about one region cannot affect prefetching in another region.

**Sandbox Prefetching** (SBP) [3] works by testing out several aggressive prefetchers in a sandboxed environment outside the real memory hierarchy in order to determine which prefetchers should be used in the real memory hierarchy. SBP evaluates prefetchers by placing prefetch addresses in a Bloom filter, rather than issuing real prefetches. Demand cache accesses check the Bloom filter to see if the address could have been prefetched by the prefetcher currently being evaluated. Hits in the Bloom filter give confidence that the evaluated prefetcher would be accurate if it were deployed in the real memory hierarchy. Several prefetchers are evaluated in round-robin fashion, and the prefetchers with the most Bloom filter hits are used to issue real prefetches. SBP evaluates aggressive prefetchers that immediately prefetch addresses with a fixed offset from the current demand access, like a next-line prefetcher. Once deployed in the real memory hierarchy, the chosen prefetchers perform no additional warm-up or confirmation before issuing prefetches.

Unlike VLDP, SBP cannot prefetch multi-delta patterns. SBP works by learning patterns observed across many pages, and then applies the most commonly observed patterns universally. Since it only prefetches the on-average most commonly observed patterns, SBP is prone to over-generalize and prefetch even when it is not appropriate, or it might miss out on patterns that are observed less often, but still performance-critical.

**The Global History Buffer** (GHB) [4] provides a framework that can be used to implement various prefetching algorithms. Here, we evaluate the GHB PC/DC algorithm, where the program counter is used to localize streams of L2 cache misses. The GHB technique requires the use of two hardware structures, namely the index table and the global history buffer itself. The GHB is a circular buffer, and each cache miss adds a new entry to the buffer. The index table contains pointers into the GHB, and can be looked up using the PC. The index table points to the most recent occurrence of the PC in the global history. In the GHB, each buffer entry is comprised of the delta from the last cache miss, and a pointer to the next instance of the current PC. The GHB therefore allows us to follow the links back into history, and replay the deltas that were seen previously.

Our GHB implementation is an adaptation of the original proposal. We keep track of PCs of load instructions in an index table, which is a direct mapped structure. Each entry in the index table points to the most recent delta seen in its page in the GHB structure.

Unlike VLDP, the GHB PC/DC algorithm can only make predictions based on histories of length one. It can, however, prefetch multi-delta sequences, and can learn patterns in one page and apply them in another.

**Spatial Memory Streaming** (SMS) [5] correlates the PC of the first load instruction to a region of memory with all of the expected cache misses in that region. Whenever a new region of memory is accessed for the first time, a new entry in the Active Generation Table (AGT) is created, indexed by a combination of the PC of the first load instruction and the offset of the initial access to that region. Subsequent accesses in that region build up a spatial pattern bitmap of all cache blocks used, until a cache block in that region is evicted, whereupon the entry is removed from the AGT and placed in a Pattern History Table (PHT). Whenever a new region is accessed for the first time, the PC+offset of the load instruction are used to look up the PHT, and if there is a match, all of the cache blocks indicated by the PHT entry's spatial pattern bitmap will be prefetched.

The original proposal for SMS assumed prefetching was done at the L1 level, however in our evaluation, all prefetchers live at the L2 level, so the stream of accesses visible to them has been filtered by the L1 cache, and all prefetching is done into the L2 cache. Our implementation of SMS tracks 2 KB regions. SMS is able to learn access patterns from one region of memory and prefetch them in another, but unlike VLDP, it cannot predict the order in which cache lines should be prefetched.

## 3. PROPOSAL

The Variable Length Delta Prefetcher (VLDP) relies on history to predict future memory requests (Figure 1). A separate local history is maintained for each physical

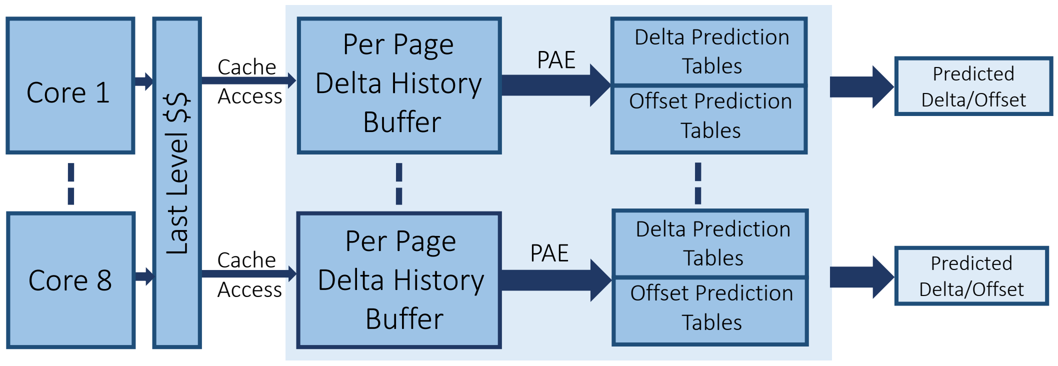

**Figure 1: Overview of VLDP.**

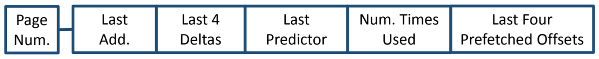

**Figure 2: Delta History Buffer entry.**

page in the workload's current working set in a small structure we refer to as the Delta History Buffer (DHB). When a reference to one of these tracked pages results in a prefetching opportunity, the page's Delta History is used to look up a prediction in the Delta Prediction Table (DPT). The DPT is structured as a set of key-value pairs that associate a delta history within a page with the delta of the next expected access in that page. This allows the DPT to make history-based predictions about what to prefetch next.

A key innovation of VLDP is the use of multiple DPT tables, where each successive table corresponds to a longer history length. This maximizes prefetch coverage and accuracy, by prefering to use long histories to make highly accurate prefetches, and using shorter histories to fill in the gaps when long histories are unavailable.

In this section, we describe the organization of VLDP in more detail, beginning with the DHB in Section 3.1, followed by the Prediction Tables in Section 3.2. The Offset Prediction Table (OPT) is described in Section 3.2.1 and the Delta Prediction Table (DPT) is described in Section 3.2.2. In each of these sections, we have made assumptions about how VLDP interacts with the rest of the memory hierarchy. First, we assume that each core has its own, separate VLDP. Second, we evaluate a 2 level cache hierarchy. Third, VLDP and all other evaluated prefetchers sit at the L2 level, and we assume all prefetches bring data into the L2 cache only. Finally, the VLDP mechanism does not take action on all L2 cache accesses. Instead, VLDP takes action only when there are accesses to the L2 cache that either result in a miss, or when a cache line is accessed that was previously prefetched into the cache. We detect these prefetch hits using a mechanism described in Section 3.1. We refer to this subset of requests as Prefetch Activation Events (PAE).

## 3.1 Delta History Buffer

The Delta History Buffer (DHB) tracks delta histories for recently accessed pages. These histories, in turn, are used to lookup the DPT and predict future memory requests. Figure 2 shows an entry in the DHB. Each entry in the DHB contains the following data for a tracked physical page: (i) page number, (ii) page offset of the last address accessed in this page, (iii) sequence of up to 4 recently observed deltas, (iv) the DPT level used for the latest delta prediction, (v) the number of times this page has been used, and (vi) sequence of up to 4 recently prefetched offsets.

Only PAEs can cause the state stored in the DHB to change. When a PAE occurs, there is a fully associative search in the DHB to find an entry with a matching page number. If no matching entry is found (DHB miss), then a not-Most Recently Used (nMRU) DHB entry is evicted and assigned to the new page number (nMRU replacement policy). The page offset of the cache line is recorded in the last address field. On subsequent hits to this page in the DHB, a delta is computed between the current access and the last address. This delta is then added to the delta sequence (last 4 deltas), and the offset of the most recent cache line (last add.) is updated to reflect the current access. The delta history maintained in the DHB is limited to the 4 most recent deltas and is tracked with a 4-entry shift register.

On a DHB hit, after the DHB entry has been updated with the most recent delta, the newly updated delta history is used to index the DPT (Section 3.2). The DHB entry for a page also stores the ID of the DPT table which was most recently used to predict the prefetch candidates for this page. This ID is used to update the accuracy of the DPT and will be described in more detail in Section 3.2.

While VLDP only issues prefetches and updates the DHB on PAEs, the DHB must be read on all L2 hits to determine if the current access was to a prefetched line, and therefore qualifies as a PAE. The DHB tracks the 4 most recently issued prefetches in each page. Since VLDP attempts to predict not only which cache lines in a page will be accessed, but also the order in which they will be accessed, this method is enough to detect most L2 hits to prefetched data, and therefore most PAEs.

## 3.2 Prediction Tables

### *3.2.1 Offset Prediction Table*

If we rely solely on deltas to make predictions, we must wait for at least a second access to a page be-

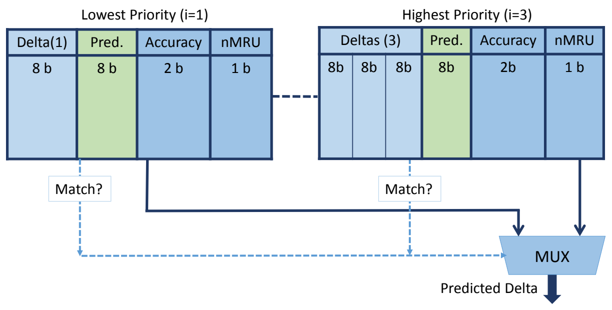

**Figure 3: Delta Prediction Tables map sequences of deltas to the next expected delta.**

fore we can begin prefetching. However, some pages are accessed very few times, and missing out on a single prefetching opportunity could have a large negative impact on performance. Therefore, in order to begin prefetching as soon as a page is accessed for the first time, VLDP uses an Offset Prediction Table (OPT).

The OPT is a global table, shared by all pages, with an entry for each cache line in a page. In our model, we use 8KB pages and 64B cache lines. The OPT is direct-mapped with 64 entries, using the page offset of the initial cache line access to a new page as the index. Each OPT entry stores a delta prediction, which predicts the second access to the page, and a 1-bit accuracy field.

On the first access to a page, the OPT is looked up using the page offset, and if the accuracy bit is set for this entry, a prefetch is issued with the predicted delta. On the second access to a page, a delta can be computed and compared with the contents of the OPT. If the OPT prediction matches the observed delta, the accuracy bit is set to 1, or remains 1 if it was already 1. If the OPT prediction does not match the observed delta, the accuracy bit is set to 0. If the accuracy bit was already 0, the old predicted delta is replaced with the new observed delta, and the accuracy bit remains 0.

### 3.2.2 *Delta Prediction Table*

While the DHB maintains separate histories for each active physical page, there is only a single, global DPT, which is shared by all active pages. Predictions stored in the DPT may survive across many allocations and evictions of DHB entries, as the program touches many pages. This makes it possible for delta sequences observed in the distant past to be used for prefetching in new pages that have never been touched before. A key feature of the DPT is that it is not just a single table, but rather a set of cascaded tables, where each table handles a different length of delta history, as seen in Figure 3. Each of the DPT tables contains multiple entries, and each entry is comprised of a key-value pair. The delta histories obtained from the DHB are used as the keys, and the delta predictions stored in the DPT are the values. Also, each DPT entry has a 2-bit accuracy counter, and a 1-bit nMRU value.

Figure 4 shows how the use of cascaded tables enables the DPT to differentiate between two different instances of the delta (2) in a repeating sequence of deltas. In

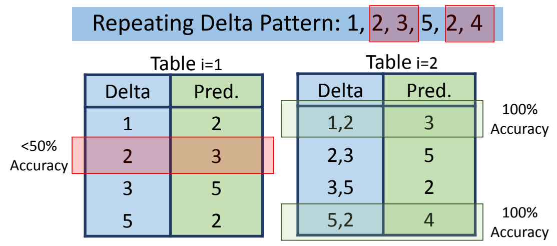

**Figure 4: Aliasing motivates the need for cascaded Delta Predictions Tables.**

one instance, (2) is followed by (3), and in the other instance, (2) is followed by (4). If we relied only on the first DPT table to predict what will come after (2), it would give the answer (3) and be correct only 50% of the time. However, the second DPT table tracks longer histories, and takes into account the delta preceding (2) to make a prediction. Here we have 2 separate entries for the delta sequences (1,2) and (5,2), eliminating the aliasing that occurred in the first DPT table. Using the second DPT table, (1,2) predicts (3), and (5,2) predicts (4), both with 100% accuracy.

When searching for a delta to prefetch, VLDP prefers to use predictions that come from DPT tables that track longer histories, in order to avoid aliasing that can occur in shorter histories, and thereby increase accuracy. Conceptually, using multiple cascaded DPT tables to track histories of varying lengths is similar to the state-of-the-art TAGE branch predictor [6].

## 3.3 Managing Cascaded Tables

Our DPT implementation uses a set of 3 DPT tables, allowing for histories up to 3 deltas long. When a PAE occurs, we look for delta history matches in all tables. The number of tables that might contain matching entries depends on the number of deltas accumulated in the page history up to this point. Pages that have only been accessed twice will have only one delta available and will be restricted to using the first DPT table for prediction. However, as the page is repeatedly accessed, and more history is accumulated, DPT lookups may produce matches in more than one of the DPT tables. In these cases, VLDP prioritizes predictions made by the table that uses the longest matching delta history, which maximizes accuracy.

Figure 3 illustrates the cascaded tables, showing the lowest priority single-delta table on the left, and the highest priority 3-delta table on the right. Each entry in the table consists of 4 basic elements: a delta history (delta), a delta prediction (pred), a 2-bit accuracy counter, and a single nMRU bit used to select one of the not Most Recently Used entries as a victim when allocating a new entry. On each PAE, the delta history in the DHB is used to lookup the DPT. When the histories match, a prediction will be made based on the delta stored in the delta prediction field. As future PAEs arrive, the DPT will be updated to reflect any changes in the delta sequence that are observed. PAEs can cause the DPT to be updated in three different ways. First, any new delta patterns will be allocated in the DPT; this will require the eviction of an earlier delta pat-

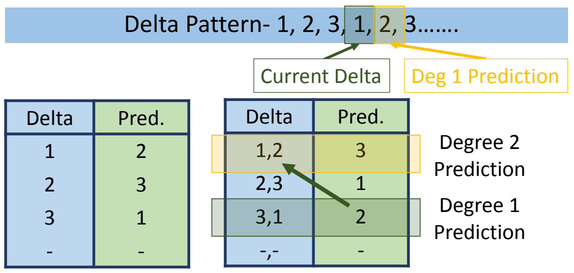

**Figure 5: Issuing multi degree prefetches.**

tern. We use a nMRU policy to make this selection. Second, each PAE can be compared with the previous delta prediction. Based on the accuracy of the previous prediction the accuracy bits of that prediction entry can be updated; incremented in the case of an accurate prediction, decremented otherwise. Finally, if the prediction accuracy is sufficiently low, the delta prediction field may be updated to reflect the new delta.

Inaccurate predictions in a DPT table T will prompt the promotion of the delta pattern to the next table T+1. If the DPT table T+1 is full, then the nMRU entry is evicted, and replaced. When the DPT is updated, if a matching delta pattern is not currently present in any DPT tables, then an entry is created for the latest delta in the shortest-history table.

## 3.4 Multi-Degree Prefetch

Once VLDP has predicted the next access in a sequence, it can make additional predictions and prefetch even further ahead by appending the predicted delta to the original history from the DHB to recursively look up the DPT. This process can be repeated as long as the predicted pattern is found in the DPT. Note that consecutive lookups in the DPT are sequential and each lookup adds latency to the subsequent prefetch. In our modelling, we assume that each lookup requires 5 cycles, and we limit ourselves to prefetch 4 deltas ahead of the current PAE. This means the fourth degree prefetch will be issued 20 cycles after the initial PAE. Despite the fact that the DPT yields 4 separate predictions in this case, the accuracy is updated only once for the table that issued the original prediction.

Figure 5 shows how multi-degree prefetching can be done by speculatively adding deltas to the history. Here, we begin with a delta history (3,1) and predict the delta 2 based on the contents of the DPT. Appending the predicted delta of 2 gives us a new delta history of (3,1,2), or simply (1,2) if only a two delta history is used. The history (1,2) can be later used to lookup the DPT and retrieve prediction associated with that history (3).

VLDP maintains a maximum prefetch distance of 4 predicted cache lines (although the delta between the current access's cache line address and the prefetched lines may be much greater than 4). After it has prefetched the next 4 predicted cache lines in a page, subsequent PAEs in that page will only yield a single additional prefetch. However, in the event of an inaccurate prefetch, VLDP may again prefetch 4 new cache lines, as it follows a different recursive chain of predictions based on the new observed history.

## 4. METHODOLOGY

### 4.1 Simulation Environment

All simulations are carried out using Wind River Simics full system simulator [7]. We interface Simics with the detailed USIMM memory model [8].

### 4.2 Simulator Parameters

We model a multi-core system with 8 OoO cores and 2 DDR3 memory channels. We assume that each channel can support two ranks. Simics and USIMM parameters are summarized in Table 1. All of our experiments are run with a shared 8MB last level L2 cache.

| Processor | |
|---|---|
| Core Parameters: | UltraSPARC III ISA, 8-core, 3.2 GHz,. 128-entry ROB, 4-wide OoO. |
| **Cache Hierarchy** | |
| L1 I-cache | 32KB/2-way, private, 1-cycle |
| L1 D-cache | 32KB/2-way, private, 1-cycle |
| L2 Cache Coherence Protocol | 8MB , 10-cycle 64B,8-way, shared Snooping MESI |
| **DRAM** | |
| DRAM Frequency | 1600 Mbps |
| Channels, ranks, banks | 2 channels, 2 ranks/channel, 8 banks/rank |
| Read Queue Length | 64 per channel |
| DRAM Timing Parameters | $t_{RC}$ = 48.75 ns $t_{RCD}$ = 13.75 ns $t_{RAS}$ = 35 ns $t_{FAW}$ = 30 ns $t_{WTR}$ = 7.5 ns $t_{RP}$ = 13.75 ns |

**Table 1: Simulator parameters [9].**

We use FR-FCFS and Open Page policies for DRAM scheduling, where row buffer hits are prioritized over row buffer misses. We use the state of the art PACMan cache replacement policy [10] to decide where to insert the prefetched cache line in the LRU stack.

### 4.3 Workloads

We use multi-programmed workloads constructed out of SPEC CPU 2006 benchmarks and multi-threaded workloads from NAS Parallel Benchmarks (NPB) and CloudSuite [11]. Because prefetchers are practically useful only for workloads with high memory traffic, we pick high MPKI workloads from SPEC CPU 2006. We also pick benchmarks like astar to show that our prefetcher does not degrade performance of workloads which don't have repeating delta patterns. For SPEC CPU 2006, we run 8 instances of each benchmark. CloudSuite, and NPB are run with 8 threads. The SPEC workloads are fast-forwarded for 50 billion instructions before starting simulations and the NPB/CloudSuite programs are simulated at the start of their region of interest. The measurements in the early part of our cycle-accurate simulations are discarded to account for various warm-up effects. All simulations were executed until a fixed point in the program to form a simulation length of approximately 500M instructions.

We also test two heterogeneous workload mixes. These mixes are a combination of 4 benchmarks. We run two instances of each benchmark, resulting in a total of 8 programs in each workload mix. The purpose of the mixed workloads is to show that VLDP does not starve workloads which are not amenable to prefetching.

Mix1 is a combination of three benchmarks that VLDP is able to aggressively issue prefetches for (*milc, lbm and zeusmp*), and one that is not amenable to prefetching (*xalancbmk*). Mix2 is a combinations of benchmarks which are not amenable to prefetching (*omnetpp, xalancbmk, soplex, mcf*), and hence VLDP is not very effective here. Once again, the purpose of this Mix is to show that VLDP does not degrade performance, in case it is not able to accurately issue prefetches.

## 5. RESULTS

### 5.1 Prefetcher Configurations

We simulate VLDP with 1 offset prediction table and 3 delta prediction tables. Each DPT has 64 entries, and the OPT also has 64 entries. The DHB keeps track of the last 16 pages that were accessed by the application. We assume that each VLDP lookup has a 5 cycle latency. On every PAE, prefetches up to the fourth degree may be issued. While issuing multi degree prefetches, we only accept predictions of tables 2-3 for degrees greater than 1. While the first DPT will enjoy high hit rates, it also has lower accuracy when compared to tables that use a longer delta history. Sensitivity analysis shown in Figure 14 shows how increasing the number of DPTs actually decreases the number of DRAM accesses by increasing the accuracy of prefetches issued. Table 2 details the per-core storage requirements for VLDP and competing schemes. Note that the per-core storage overhead of SBP is directly tied to its evaluation period; increasing SBP's storage resulted in lower performance.

| Pref | Storage | Parameters |
|---|---|---|
| SBP | 296B | 256B Bloom filter, 16 candidates |
| FDP | 3.1KB | 2.5KB tag array, 4Kb bloom filter |
| AMPM | 4KB | 52 Access Maps, each tracking 16KB |
| GHB | 4KB | 256 entry GHB, 256 entry Index Table |
| SMS | 22KB | AGT 4KB, PHT 17.6KB, PR 256B |
| VLDP | 998B | OPT 128B, DHB 222B, DPT 648B |

**Table 2: Prefetcher Storage Overheads**

### 5.2 Performance Evaluation

Figure 6 shows the weighted speedup of VLDP in comparison to FDP, SBP, and AMPM. Weighted speedup is the geometric mean of speedup for each thread, where the speedup is calculated against the IPC of that thread without a prefetcher. Hence, the baseline has a speedup of 1, and a speedup of less than 1 shows a performance degradation.

The performance gains seen by VLDP come as a result of two factors.

- Long repeating multi-delta sequences
- Long single-delta sequences

Workloads such as *milc, lbm, LU,* and *soplex* have long multi-delta sequences that repeat across pages. VLDP is ideally suited to handle such access patterns. *Lbm* ends up accessing every cache line inside a page. However, initially, *lbm* starts with positive deltas, of (+2, +3), and then upon reaching the end of the page returns back to the beginning of the page, and then accesses the cachelines that were not accessed in the initial page traversal, eventually touching all the cachelines in the page. Prefetchers like SBP identify this pattern as one that touches all lines in the page, and end up prefetching all cache lines starting from the start of the page. Many of these cache lines will not be used till the stream reaches the end of the page and then reverses direction. In comparison, VLDP is able to detect these complex patterns and issue prefetches in a timely manner.

*Omnetpp* accesses a majority of pages in short sequences of 3-4 accesses. The same page is accessed again only after a long time. However, the same short delta pattern is repeated again. VLDP shows a 10% speedup over no prefetch for *omnetpp*, whereas AMPM shows a speedup of 4%. This kind of pattern leads to low but similar accuracies in several candidate prefetchers in the Sandbox prefetcher. Depending on the accuracy threshold to issue prefetches, this leads to either no prefetches, or several inaccurate prefetches. Short non-homogeneous patterns lead to several identified but unconfirmed streams in FDP, thus leading to a low number of prefetches issued.

Workloads such as *libquantum, zeusmp, CG,* and *IS* have uniform delta sequences that mostly consist of "+1" deltas. VLDP is able to predict these deltas while using very few entries in the Delta Prediction Tables. *Libquantum* is dominated by "+1" streams, where every cache line in a page is accessed sequentially. All prefetchers we evaluate perform well for this workload. FDP needs a couple of accesses to a stream before it can be confirmed whereas SBP do not have this problem. However, SBP and AMPM issue prefetches in bursts, leading to congestion at the DRAM controller for short periods of time. VLDP suffers from neither of these problems and hence is able to outperform these prefetchers. For these streaming workloads, VLDP effectively works like a streaming prefetcher that has a Prefetch Distance of 4 and a Prefetch Degree of 1, thus issuing prefetches only as fast as the access stream.

*MG* has a combination of both repeating "+1" deltas, and multi-delta sequences that span the entire page. To predict "+1" streams, VLDP uses just one entry in the third table. The rest of the entries can be used to predict the multi-delta sequences. It is for this reason that VLDP does exceedingly well with *MG*.

Workloads such as *mcf, xalancbmk,* and *astar* do not exhibit streaming behavior. VLDP does not yield much performance improvement for these workloads.

The performance of VLDP is 5.8% better than AMPM, 17.2% better than FDP and 8.5% better than SBP.

When VLDP was evaluated with a higher degree of 8, it resulted in <1% performance improvement. In our platform, workloads benefiting from higher degree

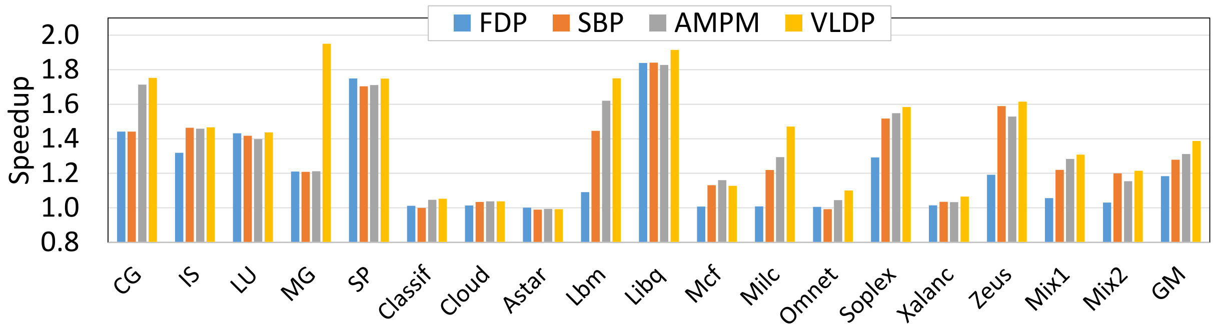

**Figure 6: Performance comparison of prefetchers that do not use the program counter.**

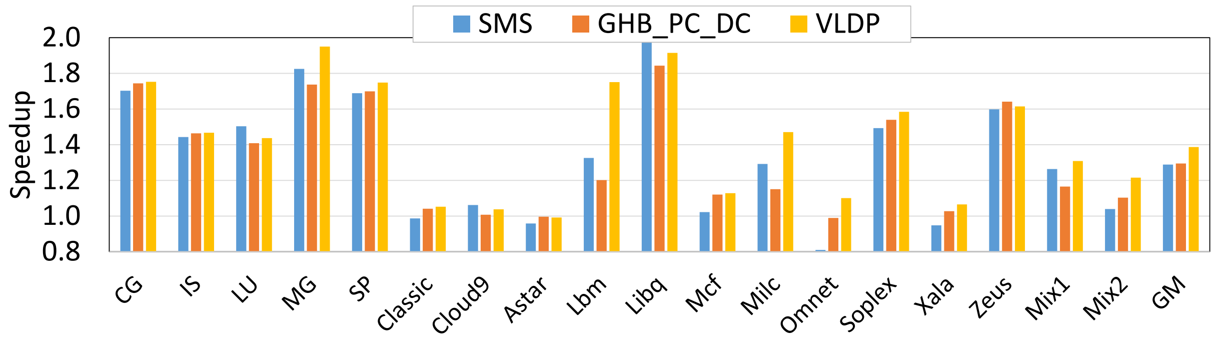

**Figure 7: Comparing the performance of VLDP to SMS and GHB PC/DC.**

prefetching (e.g., libquantum) are already operating the memory bus near saturation.

## 5.3 Comparing VLDP to Prefetchers that use the Program Counter

Figure 7 shows the weighted speedup of VLDP in comparison to SMS and GHB, both of which use the program counter. GHB memorizes the sequence of deltas that are seen by the application. It then uses the PC of the current load instruction and the current delta to index into this list of deltas. GHB stores all the deltas seen in a circular buffer, whereas VLDP stores only unique delta patterns, which leads to much lower storage overhead for VLDP. GHB uses only the current delta to make a prediction while VLDP tries to correlate all the latest deltas to make a prediction. The cascaded tables in VLDP provide better immunity against aliasing (Figure 4), which leads to better accuracy. Overall, VLDP has an accuracy of 61% while GHB has an accuracy of 33%. Across all the benchmarks evaluated, VLDP performs 7.1% better than GHB PC/DC.

SMS uses the footprint of previously seen access patterns to make prefetch predictions. It uses the PC of the load and the page offset of the load instruction to index into the table that stores the Spatial Patterns. Unlike VLDP and GHB, SMS does not remember the order in which the cachelines within a spatial pattern were accessed. For workloads like *MG* and *lbm* whose access patterns zigzag from the start to the end of the page, this leads to poor timeliness. Overall, VLDP has 7.6% higher performance than SMS. SMS can make as many predictions as the number of cachelines in a region (32 for a region size of 2KB) based on a single correlation between the PC and the trigger offset. VLDP can at the most issue 4 prefetches based on a single delta (in the worst case). Among the 6 prefetchers we evaluated, SMS outperforms all others for *libquantum*. This is because load instructions are issued by only a few unique PC values and moreover, *libquantum* has only one dominant access pattern. The performance of *libquantum* with SMS is 5% better than that with VLDP. SMS avoids the bursty behavior that affects SBP and AMPM by making use of the Prediction Register that enqueues the prefetches issued, and then dispatches them over time. However, this aggressive behavior can also lead to inaccurate predictions for benchmarks that do not show strong spatial correlation such as *astar, omnetpp and mcf*.

The key advantage of VLDP is the fact that it uses many events/correlations to make a few predictions – this leads to better accuracy and coverage. SMS uses a single correlation (PC and offset) to trigger a large number of prefetches. If SMS sees multiple spatial patterns corresponding to one PC/offset tuple, it has to either merge the two patterns or pick one of the two.

## 5.4 Cache Misses and Prefetcher Coverage

Figure 8 shows the number of cache misses per thousand instructions normalized to the MPKI of the no prefetch baseline. Workloads like *milc, lbm* show large drops in MPKI for VLDP; this is because of the presence of repeating multi-delta sequences. None of the prefetchers we evaluated were able to reduce the MPKI of astar. In fact, SBP increases the MPKI of *astar* by 8% and AMPM increases it by 1%. Increase in MPKI is a result of cache pollution caused by overly aggressive prefetching.

For workloads that have streams with uniform deltas,

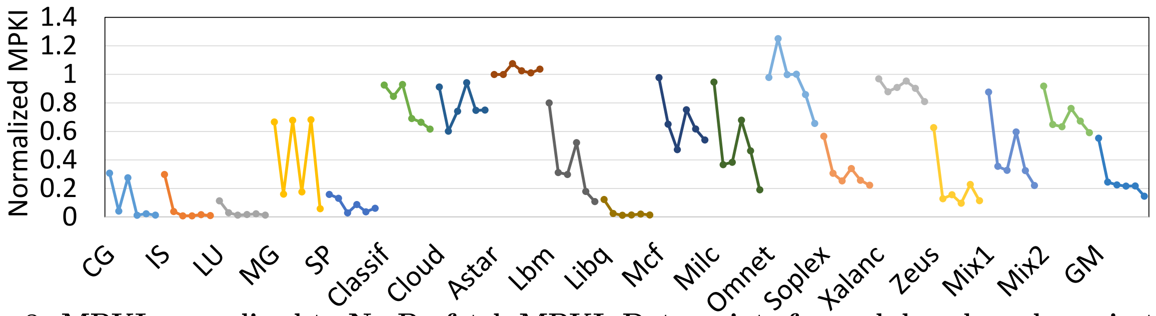

**Figure 8: MPKI normalized to No-Prefetch MPKI. Data points for each benchmark are in the order of FDP, SMS, SBP, GHB_PC_DC, AMPM, VLDP.**

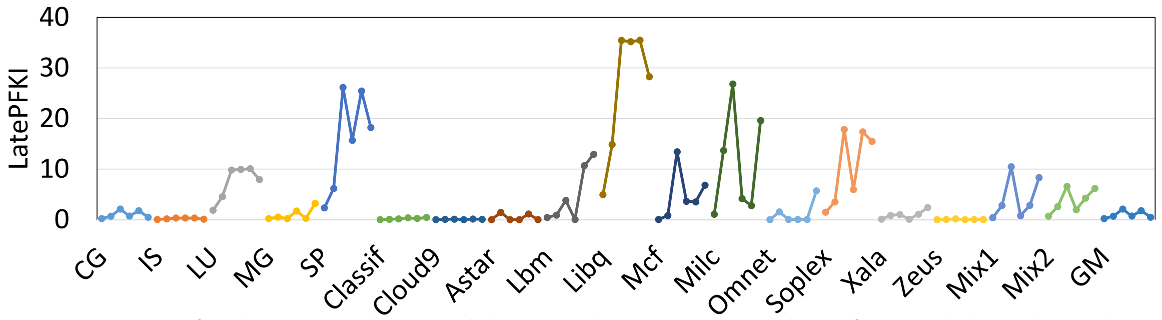

**Figure 9: Late Prefetches per thousand instructions. Data points for each benchmark are in the order of FDP, SMS, SBP, GHB_PC_DC, AMPM, VLDP.**

most prefetchers are able to predict the cache accesses accurately. NPB benchmarks, *libquantum, zeusmp* show a large decrease in MPKI when compared to the baseline. As seen in Figure 9, even though prefetchers like SBP and AMPM are successful in predicting cache miss patterns, they often issue very large number of attempted prefetches, most of which are cache hits. Depending on the workload, the number of prefetches per cache miss can even exceed 100. When the number of attempted prefetches per cycle per core is constrained, these prefetchers end up issuing prefetches too late. By construction, the number of prefetches VLDP issues is less than or equal to the maximum degree that it is allowed to prefetch. In our design this is 4. SBP and AMPM can have multiple candidate prefetchers issuing predictions for multiple degrees. Even though these prefetches are cache hits, they still need cache tag lookups, and hence can be a bottleneck for these prefetchers.

Figure 10 shows the coverage of the different prefetchers. Coverage is the ratio of number of unique cache accesses that were prefetched to all unique cache accesses. VLDP had the highest coverage for most of the workloads we simulated.

## 5.5 Prefetcher Accuracy and DRAM accesses

Figure 11 shows the increase in DRAM accesses for each prefetching configuration. SBP dynamically decides aggressiveness depending on the bandwidth available. Cloud workloads like *classification* and *cloud9* are low bandwidth workloads; SBP increases the number of DRAM accesses of classification by 152%, and cloud9 by 68% while yielding less than 3% improvement over the baseline.

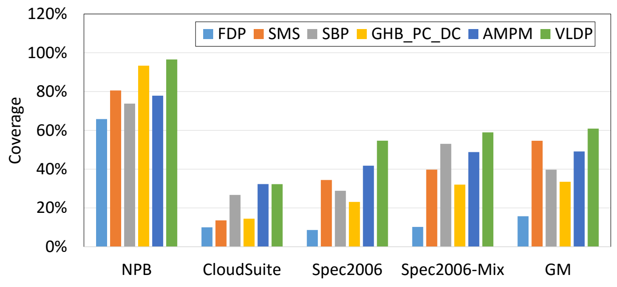

**Figure 10: Coverage per benchmark suite.**

For all workloads, FDP favors accuracy over aggressiveness. On average, FDP has the lowest increase in DRAM accesses of 3.7%, GHB has 5.4%, AMPM has 13.4%, SBP has 22.6%, SMS has 60.5%, and VLDP has 17.2%.

Prefetchers like AMPM, SBP, and FDP use some form of negative feedback to reduce prefetcher aggressiveness. Because SMS does not have negative feedback built into it, for workloads that do not show strong spatial correlation, SMS ends up issuing many inaccurate prefetches.

Figure 12 shows the accuracy of the prefetchers. Accuracy is the ratio of useful prefetches to total prefetches issued. NPB workloads all show high accuracy for all evaluated prefetchers. In CloudSuite applications, the lower coverage FDP and GHB show the highest accuracy. SBP and SMS are consistently the least accurate prefetchers, and GHB is the overall most accurate, at the cost of prefetch coverage. VLDP strikes a balance between having the highest coverage of any evaluated prefetcher, while at the same time being among the most accurate.

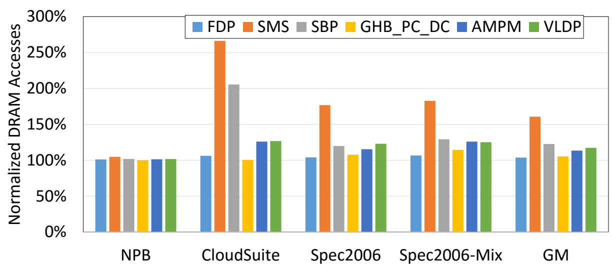

**Figure 11: Normalized DRAM accesses per benchmark suite.**

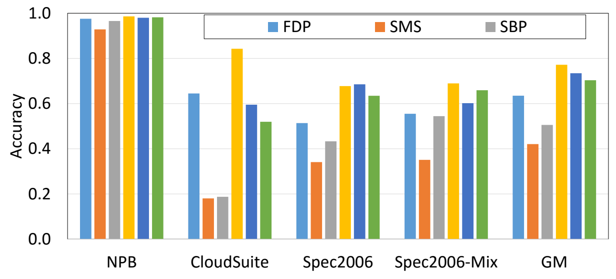

**Figure 12: Accuracy per benchmark suite.**

## 5.6 Comparison to the Best-Offset Prefetcher

We evaluated the Best Offset prefetcher [12] from the second Data Prefetching Championship. We swept through the prefetcher parameters to determine the best configuration for the BO prefetcher on our simulation infrastructure. On average, the BO prefetcher's performance is 7% below VLDP and it issued 6% fewer DRAM accesses for our workloads. The BO prefetcher outperformed VLDP for 4/18 workloads that we evaluated.

The biggest drawback of the BO prefetcher on our simulation infrastructure is that it does not issue multi degree prefetches, thus causing late prefetches. Workloads like libquantum and lbm, which have long streams, benefit greatly from multi degree prefetches. Even though the BO prefetcher correctly predicts most of these accesses, the prefetches were issued too late. Note that multi-degree prefetching did not yield benefits in the DPC2 simulator for the BO prefetcher, even in simple workloads like libquantum.

In an attempt to get the most out of the BO prefetcher on our simulation infrastructure, we also evaluated a multi degree version of the BO prefetcher, where we statically set the highest degree prefetch BO is allowed to issue. The BO prefetcher with prefetch degree 2 was able to perform within 3.5% of VLDP while issuing 0.2% more DRAM accesses than VLDP. When the degree was increased further, the average accuracy and performance dropped.

## 5.7 Page + PC Localization

VLDP uses page localization to identify streams. Other prefetchers like the GHB and SMS have used the Program Counter (PC) to localize streams. In this section, we explore the effect of using the PC, as well as the page number to localize streams. We call this configuration VLDP-PC. In the new version (VLDP-PC), successive accesses to the same page from the same PC are considered part of the same stream.

The accuracy of VLDP-PC is 6% higher than VLDP. However, due to a 7.5% reduction in coverage, there is a 1.7% reduction in performance. The increase in accuracy leads to a 7.5% decrease in the number of DRAM accesses issued. VLDP-PC is able to achieve this performance with 32 entries in the DHT, 16 entries in the DPT, and 16 entries in the OPT, leading to a hardware overhead of only 830B/core. Decomposing accesses in a page across PCs reduces coverage, but lowers overhead because fewer patterns must be tracked in the DPTs.

## 5.8 Sensitivity Analysis

We also simulated FDP, GHB, AMPM, SBP, and VLDP for other cache sizes (512KB and 2MB); our simulations show that the speedup changes by at most 1%. Detailed results have been omitted for space reasons.

### 5.8.1 *Sensitivity to table size*

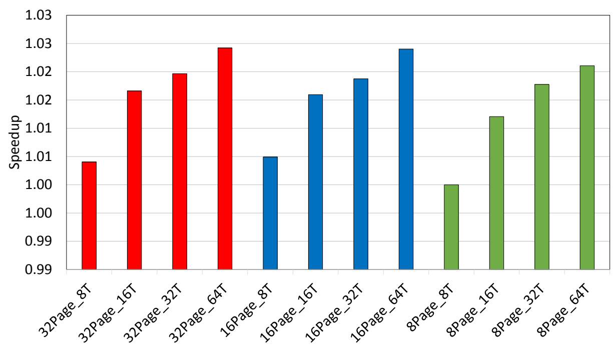

**Figure 13: Sensitivity to Table Sizes**

Figure 13 shows how the normalized performance of VLDP changes across all workloads with VLDP tables of different sizes. In this experiment, we vary the number of pages tracked per core by the DHB, and the number of entries in the DPT. We use the following naming convention for the X axis $iPage\_jT$, where $i$ is the number of pages tracked by DHB, and $j$ is the number of entries in the DPT. The right most bar shows the performance when DHT tracks 8 pages and there are 64 entries in the DPT. The performance changes by less than 1% when the size of the DHB is reduced all the way to 8. There is almost a 2% increase in performance when the number of entries in the DPT is increased from 8 to 64. We choose a DHT size of 16 and a DPT size of 64 as the optimal performance/area/complexity trade-off. All the other VLDP results presented in this paper are for the $16Page\_64T$ configuration.

### 5.8.2 *Sensitivity to number of prediction tables*

Figure 14 shows the impact of changing the number of cascaded Delta Prediction Tables (DPTs) in VLDP in terms of both speedup and number of DRAM accesses. The initial data point, 1DPT_NoOPT, shows the speedup with single delta table (the offset table

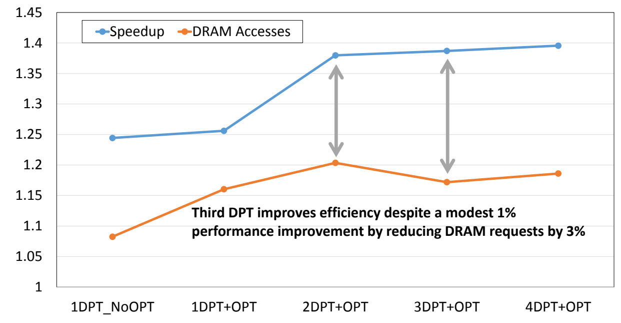

**Figure 14: Sensitivity to Number of DPTs**

is excluded). Note that computing a single delta requires two addresses. As a result, the 1DPT_NoOPT config issues no prefetches on the first access to a page, instead waiting until the second access. The second data point depicts a single DPT supplemented with the OPT (Offset Prediction Table). Moving to the right, each additional data point adds an additional DPT. 3DPT+OPT, for example, includes 3 delta prediction tables (DPT) as a well an offset table (OPT). The intent of this plot is to illustrate how additional tables can impact both speedup and the number DRAM accesses. Compared to 1DPT+OPT, the addition of the second table, 2DPT+OPT, yields significant benefits, improving performance by 10% but at the cost of increasing the DRAM accesses by about 4%. This result matches our intuition; increased prefetching coverage yields better performance but at the cost of increased memory traffic. The addition of third DPT, 3DPT+OPT, however produces the opposite result. Despite a modest performance improvement of about 1%, 3DPT+OPT reduces DRAM requests by 3%. This is because of improved matching made possible by the longer delta histories, which improves accuracy without sacrificing coverage. In fact, this result highlights the benefit of VLDP's cascaded tables. The use of shorter history DPTs ensures high coverage, ensuring that even short delta histories will result in a match and produce a prefetch request. At the same time, the longer history DPTs (the third DPT, in this case) maximize accuracy when longer histories become available.

### 5.8.3 *Sensitivity to DHB Region size*

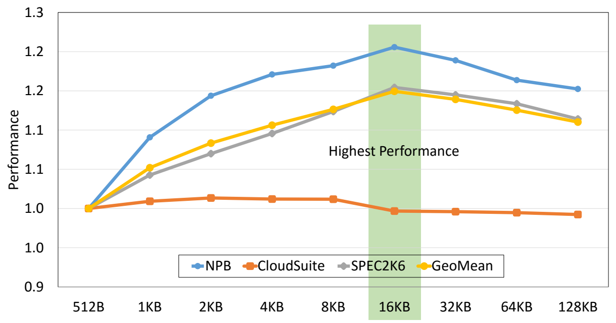

**Figure 15: Sensitivity to DHB region size.**

VLDP uses page localization to identify streams. Localization is a technique used by many prefetchers in the past like GHB and SMS. Localization works with the assumption that most data structures like arrays and structs are limited to certain regions of memory, and if parts of a data structure are accessed in a particular order once, they will likely be accessed in the same order again. Small regions will have limited information when making predictions, and limited coverage. Large regions suffer from interference because accesses to unrelated data structures are merged into a single noisy delta pattern.

Figure 15 shows the performance of VLDP as the size of the region tracked by the DHT is varied from 0.5KB to 128KB. In this experiment, when the region tracked is larger than 8KB, the OS page size is also increased to match the size of the region.

The highest performance is seen when the size of the region and the OS page is 16KB, which is 2% higher than region size of 8KB (used in the rest of the paper). The performance starts to drop after 16KB because of higher interference. A similar observation was also made by SMS. Thus, VLDP is perfectly compatible with architectures that use large page sizes, but it must use smaller DHB region sizes within these large pages to make high-quality predictions.

## 6. RELATED WORK

Prefetching is a well-studied field, and many novel prefetchers have been proposed [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 10, 27]. Prior prefetching work can be characterized as focusing on either regular or irregular patterns. Regular access patterns feature repeating sequences within or across spatial regions of memory [5]. Irregular access patterns have no discernible spatial patterns, and can only be learned through memorization of long sequences of accesses [28]. Somogyi et al., [29] propose STeMS, which does both.

This work focuses on the problem of regular data prefetching. Section 2 contains descriptions of recent regular data prefetchers. In addition, Joseph et al. proposed the use of Markov predictors to make prefetch decisions [19]. In their design, each cache miss to address A generates an entry in a predictor table, and subsequent cache miss addresses are added to A's entry, so they can be prefetched the next time A is seen.

Irregular data prefetching is explored by two recent papers, Spatio-Temporal Memory Streaming [29], and Linearizing Irregular Memory Accesses for Improved Correlated Prefetching [30] (introducing the Irregular Stream Buffer, or ISB).

The STeMS mechanism works by replaying the interleaving access pattern between different spatial maps. Both the spatial maps and the temporal interleaving pattern must be learned and stored. The ISB builds on the concept of the GHB [4] by translating correlated addresses into a new address space where they are consecutive. Observed memory accesses with arbitrary physical addresses are linearized in this new address space, and then translated back to physical addresses to issue prefetches. These proposals both use large amounts of storage to remember entire sequences of accesses (ISB uses 32 KB on-chip storage, and 8 MB of off-chip storage per core, and STeMS uses 1.64 MB per core), and they both require the program counter (PC) to work. Because they require high storage budgets and the PC, they are not direct competitors for VLDP.

## 7. CONCLUSIONS

We have proposed Variable Length Delta Prefetching to identify complex patterns that repeat in many physical pages, and then prefetch those patterns as soon as they appear in new pages. VLDP uses multiple prefetch delta prediction tables, which are indexed by varying lengths of intra-page delta history. This mechanism is able to capture short, simple patterns with only a few accesses per page, as well as complex patterns involving multiple positive and negative delta patterns.

Variable Length Delta Prefetching increases performance by simultaneously increasing coverage and accuracy, compared to the state of the art Access Map Pattern Matching and Sandbox prefetchers. VLDP improves upon the performance of AMPM by 5.8% on average, and improves upon the performance of SBP by 8.5% on average, across the evaluated workloads. It does this by increasing prefetch coverage by 24% compared to AMPM, and 53% compared to SBP, and by improving accuracy by 24% compared to AMPM, and 53% compared to SBP. It achieves this performance using only 998 B of storage per core overhead.

## 8. ACKNOWLEDGEMENTS

This work was supported in part by Intel and by NSF grants CNS-1303663 and CNS-1423583.

## 9. REFERENCES

[1] Y. Ishii, M. Inaba, and K. Hiraki, "Access Map Pattern Matching for High Performance Data Cache Prefetch," *The Journal of Instruction-Level Parallelism*, vol. 13, January 2011.

[2] S. Srinath, O. Mutlu, H. Kim, and Y. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," in *Proceedings of HPCA*, 2007.

[3] S. Pugsley, Z. Chishti, C. Wilkerson, T. Chuang, R. Scott, A. Jaleel, S.-L. Lu, K. Chow, and R. Balasubramonian, "Sandbox Prefetching: Safe, Run-Time Evaluation of Aggressive Prefetchers," in *Proceedings of HPCA*, 2014.

[4] K. Nesbit and J. E. Smith, "Data Cache Prefetching Using a Global History Buffer," in *Proceedings HPCA*, 2004.

[5] S. Somogyi, T. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial Memory Streaming," in *Proceedings of ISCA*, 2006.

[6] A. Seznec and P. Michaud, "A case for (partially) TAgged GEometric history length branch predictor," *Journal of Instruction-Level Parallelism*, vol. 8, 2006.

[7] "Wind River Simics Full System Simulator," 2007. `http://www.windriver.com/products/simics/`.

[8] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. Pugsley, A. Udipi, A. Shafiee, K. Sudan, M. Awasthi, and Z. Chishti, "USIMM: the Utah SImulated Memory Module," tech. rep., University of Utah, 2012. UUCS-12-002.

[9] "Micron DDR3 SDRAM Part MT41J1G4," 2009.

[10] C. Wu, A. Jaleel, M. Martonosi, S. Steely, and J. Emer, "PACMan: Prefetch-Aware Cache Management for High Performance Caching," in *Proceedings of MICRO-44*, 2011.

[11] 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 ASPLOS*, 2012.

[12] P. Michaud, "A Best-Offset Prefetcher," in *Data Prefetching Championship*, 2015.

[13] V. Jimenez, R. Gioiosa, F. Cazorla, A. Buyuktosunoglu, P. Bose, and F. O'Connell, "Making Data Prefetch Smarter: Adaptive Prefetching on POWER7," in *Proceedings of PACT*, 2012.

[14] J. Baer and T. Chen, "An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty," in *Proceedings of Supercomputing*, 1991.

[15] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in *Proceedings of ISCA-17*, pp. 364–373, May 1990.

[16] S. Palacharla and R. Kessler, "Evaluating Stream Buffers as a Secondary Cache Replacement," in *Proceedings of ISCA-21*, pp. 24–33, April 1994.

[17] E. Ebrahimi, O. Mutlu, and Y. Patt, "Techniques for Bandwidth-Efficient Prefetching of Linked Data Structures in Hybrid Prefetching Systems," in *Proceedings of HPCA*, 2009.

[18] A. Roth, A. Moshovos, and G. Sohi, "Dependence Based Prefetching for Linked Data Structures," in *Proceedings of ASPLOS VIII*, pp. 115–126, October 1998.

[19] D. Joseph and D. Grunwald, "Prefetching Using Markov Predictors," in *Proceedings of ISCA*, 1997.

[20] F. Dahlgren, M. Dubois, and P. Stenstrom, "Sequential Hardware Prefetching in Shared-Memory Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, 1995.

[21] K. Nesbit, A. Dhodapkar, and J. Smith, "AC/DC: An Adaptive Data Cache Prefetcher," in *Proceedings of PACT*, 2004.

[22] I. Hur and C. Lin, "Memory Prefetching Using Adaptive Stream Detection," in *Proceedings of MICRO*, 2006.

[23] S. Iacobovici, L. Spracklen, S. Kadambi, Y. Chou, and S. Abraham, "Effective Stream-Based and Execution-Based Data Prefetching," in *Proceedings of ICS*, 2004.

[24] F. Dahlgren and P. Stenstrom, "Evaluation of Hardware-Based Stride and Sequential Prefetching in Shared-Memory Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7(4), pp. 385–395, April 1999.

[25] J. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy, "Power4 System Microarchitecture," tech. rep., Technical White Paper, IBM, October 2001.

[26] J. Fu, J. Patel, and B. Janssens, "Stride Directed Prefetching in Scalar Processors," in *Proceedings of MICRO-25*, pp. 102–110, December 1992.

[27] S. Kumar and C. Wilkerson, "Exploiting Spatial Locality in Data Caches Using Spatial Footprints," in *Proceedings of ISCA*, 1998.

[28] T. Wenisch, S. Somogyi, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi, "Temporal Streaming of Shared Memory," in *Proceedings of ISCA*, 2005.

[29] S. Somogyi, T. Wenisch, A. Ailamaki, and B. Falsafi, "Spatio-Temporal Memory Streaming," in *Proceedings of ISCA*, 2009.

[30] A. Jain and C. Lin, "Linearizing Irregular Memory Accesses for Improved Correlated Prefetching," in *Proceedings of MICRO*, 2013.