Efficiently Prefetching Complex Address Patterns

Manjunath Shevgoor[†], Sahil Koladiya[†], Rajeev Balasubramonian[†], Zeshan Chishti[‡]
[†]University of Utah, [‡]Intel Labs

Abstract

Prefetching continues to be an active area for research, given the memory-intensive nature of several relevant workloads. Prior work has focused on predicting 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). VLDP correlates address deltas between successive cache line accesses that are seen across many physical pages, and then uses those correlations to issue prefetches in new pages. These correlations are built up in several tables, and each table corresponds to a history of delta patterns with a different length. For example, the shortest history table uses only the most recent access delta to make a prediction, but the longest history table uses the most recent three access deltas to make a prediction. Longer histories generally yield more accurate predictions, so VLDP fits the current delta history into 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 confirmation, and it is even able to prefetch some patterns which show no repetition within a physical page.

1. Introduction

Memory latency continues to be a significant bottleneck in today's processors. Work addressing this bottleneck ranges from building DRAM caches and exotic new memories, to continuing efforts to improve mature technologies such as replacement and prefetching algorithms. Prefetching in particular 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, 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 particular problem for prefetchers that rely on discovering common deltas between consecutive 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.

Multi-Delta Sequences: In this paper, we define a delta to be the difference between the addresses of successive accesses to the same physical page. 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.

We define a multi delta sequence as a repeating sequence of different deltas. To discover patterns with multi delta sequences, 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 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 Intel's stream prefetcher [4], 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 [2] that do support both of these capabilities.

The key disadvantage of algorithms like AMPM is training time; only when a stream has been confirmed, can it start prefetching. We therefore introduce the Variable Length Delta Prefetcher (VLDP), which is designed to efficiently predict multi-delta sequences. The VLDP learns from multi-delta sequences by remembering previously occurring delta pairs. 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.

The following key observations motivate this work. First, virtual memory restricts pattern discovery to occur only within the boundaries of a physical page. Second, the discovered patterns will often be short, because they are interrupted by transitions between physical pages. Third, in light of the previous two observations, maximizing prefetch coverage relies

on minimizing the training time in a newly accessed physical page. Fourth, many workloads contain multi-delta sequences, in addition to the more commonly exploited single delta sequences.

In light of these observations, our proposed Variable Length Delta Prefetcher (VLDP) 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 the evaluated previously proposed regular data prefetchers like AMPM, across our suite of workloads.

2. Proposal

The Variable Length Delta Prefetcher (VLDP) relies on history to predict and prefetch future memory requests, as seen in Figure 1. A separate local history is maintained for each active physical page in a small structure we refer to as the Delta History Buffer (DHB). When a reference to one of the active pages results in a prefetching opportunity, the Delta History is used to look up a prediction in the Delta Prediction Table (DPT). The DPT consists of a number of key-value pairs with the intent of correlating previously occurring delta history with a subsequently occurring delta. This allows the DPT to make history based predictions about future deltas.

In this section, we describe the organization of VLDP in more detail, beginning with the DHB in Section 2.1, followed by the DPT in Section 2.2, and finally describing a variant of the DPT we refer to as the Offset Prediction Table (OPT) in Section 2.3. In each of these sections, we have made the following assumptions about how the VLDP interacts with the rest of the memory hierarchy. First, we assume that each core is allocated a separate VLDP. Second, we evaluate a 2 level cache hierarchy and we assume all prefetches bring data into the L2 cache.

2.1. 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 3 recently observed deltas, and (iv) the DPT level used for the latest delta prediction.

When a cache access occurs, the page number is looked up in the DHB. If no matching entry is found (DHB miss), then the oldest DHB entry is evicted and assigned to the new page number (FIFO 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 3 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 3 most recent deltas and is tracked with a 3-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 2.2). 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 the next section.

2.2. Delta Prediction Table.

Although the DHB maintains separate histories for different physical pages, there is only a single DPT, shared by all active pages. In fact, predictions stored in the DPT may survive across many instances of DHB entries being allocated and evicted. This enables delta histories observed in some pages in the past to be used for prefetching new pages that have never been encountered before. Each of the DPT tables contains multiple rows, with each row 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. An additional 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.

The need for multiple DPTs can be explained with an example. If the delta (2) is followed by (3) in one case and (4) in the other case, the accuracy of the first DPT table is at most 50%. The addition of a second table, which uses a two-delta history, can distinguish the (1,2,3) pattern from the (5,2,4) pattern, enabling accurate prediction of the delta following (2) in both cases.

2.2.1. Managing Cascaded Tables. Our DPT implementation uses a set of 3 DPT tables, allowing the use of histories up to 3 deltas long. When a cache access occurs, we look for delta history matches in all tables. If multiple tables have a match, VLDP prioritizes predictions made by the table which 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 three basic elements: a delta history (delta), a delta prediction (pred) and accuracy. On each cache access, the delta history in the DPT will be compared to the delta history from the DHB. When the histories match, a prediction will be made based on the delta stored in the delta prediction field.

Since DPT lookups may produce multiple matches, many DPT entries in the lower priority tables may become stale over time. To prevent these dead entries from taking up space in the DPT, we evict the oldest entry in the DPT.

The DPT is updated to reflect the accuracy of its previous

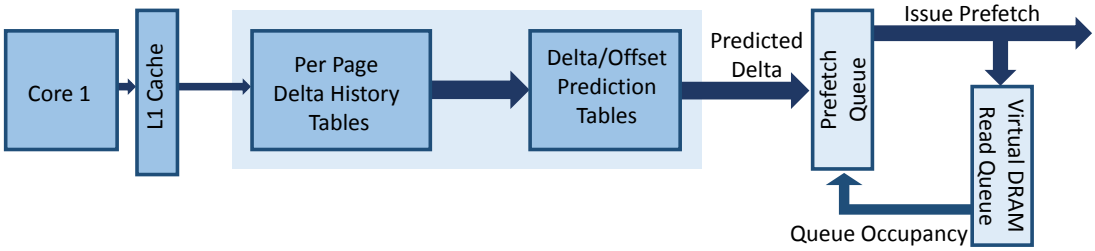

Figure 1: Overview of the Variable Length Delta Prefetcher.

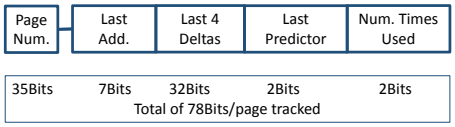

Figure 2: Delta History Buffer entry.

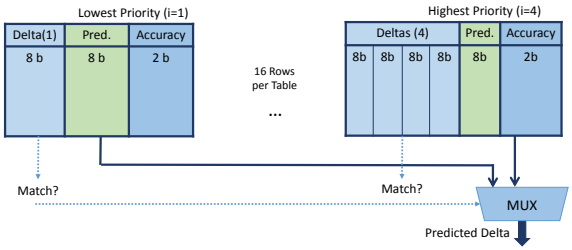

Figure 3: Delta Prediction Tables.

predictions as well as update the DPT with new deltas. These updates involve two steps: (i) updating the prediction delta and accuracy, and (ii) updating the delta patterns themselves. The accuracy of the delta predictions can only be updated after the predicted address has been requested. When a cache access occurs, the DPT must be checked to see if the request had already been predicted by an earlier prefetch. Using the last predictor field in the Delta History Buffer (DHB), we can identify the DPT table responsible for the prediction, and compare its prediction against the current address. If the DPT table correctly anticipated the current address, the accuracy of the DPT table is incremented, otherwise it is decremented.

Incorrect predictions in a DPT table T will prompt the promotion of the delta to the next table (T+1). If the DPT table T+1 is full, then the oldest entry is evicted. When the DPT is updated, if the delta pattern is not present in any tables, then an entry is created for the latest delta in the shortest-history table.

2.2.2. Multi-Degree Prefetch. Once the VLDP has predicted the next access in a sequence, the VLDP can predict 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 modeling, we assume that each lookup requires 5 cycles.

2.3. Offset Prediction Table.

One drawback to using deltas to make predictions is that we must have at least two references to a page before we can compute a delta. Some pages are sparsely used, and can benefit from the ability to make prefetch decisions without any delta information. To address this, we introduce the Offset Prediction Table (OPT). In contrast to the DPT, which relies on using deltas to make predictions, the OPT relies only on the first cache line offset referenced in a page. Since the OPT can make predictions with only a single offset (and not a delta), prefetching requests can be made as early as the very first reference to a new page.

2.4. Virtual Memory Read Queue.

Prefetchers add to the memory traffic, because most prefetch predictions are within a page, there is a high chance that they end up becoming row hits. If the memory scheduler uses an FR-FCFS policy, this can lead to unfairness. Demand requests which are row misses end up waiting for the row hits to be serviced, hence increasing the latency for demand requests. Kaseridis et al [3] proposed prioritizing demand requests over prefetch requests even when the prefetch requests are row hits. The memory scheduling policy used in the second Data Prefetching Championship treats both demand requests and prefetch requests in the same way. In order to prevent prefetch requests from delaying service to demand requests, we implement a Virtual DRAM Queue (VDQ).

Whenever there is an L2 cache miss or when we issue a prefetch, we add this request to the DRAM queue. The length of the VDQ is used as the proxy for the traffic seen at the memory system. When there is a cache fill into L2, the corresponding entry is evicted from the VDQ.

2.5. Issuing Prefetches.

When the OPT or the DPT makes a prediction, the prefetch is not issued immediately. Instead, this prefetch is inserted into the Prefetch Queue (PFQ). Prefetch requests from the PFQ are issued only when the page to which the prefetch request belongs to has less than 3 entries in the VDQ, and also when the current VDQ occupancy is less than 100 requests. This is done in order to prevent overloading the memory system with prefetches.

3. Results

3.1. Prefetcher Configurations

VLDP was evaluated using the infrastructure released for the 2nd Data Prefetching Championship. We evaluate 11 workloads from the SPEC2006 benchmark suite [1]. All workloads have been simulated for 1 Billion instructions after fast forwarding each workload by 10B instructions. We simulate **VLDP** with 1 offset prediction table and 3 delta prediction tables. Each Delta prediction table has 64 entries, and offset prediction table has 64 entries. The Delta History Buffer keeps track of the last 128 pages that were accessed by the program. On every cache access, prefetches up to the fourth degree may be issued. Table 1 shows per core hardware area overhead for VLDP.

Offset Prediction Table	392 B
Delta History Buffer	1648 B
Delta Prediction Table	552 B
Prefetch Queue	1675 B
Virtual Dram Queue	1250 B
Total	**5.5 KB**

Table 1: VLDP Hardware Area Overhead

AMPM stores an access map table, which tracks the status of every line in an 850KB region. 2 bit counters are used to track status of every cache line within 850KB region. The total storage overhead of AMPM is 4KB per core.

3.2. Performance Evaluation

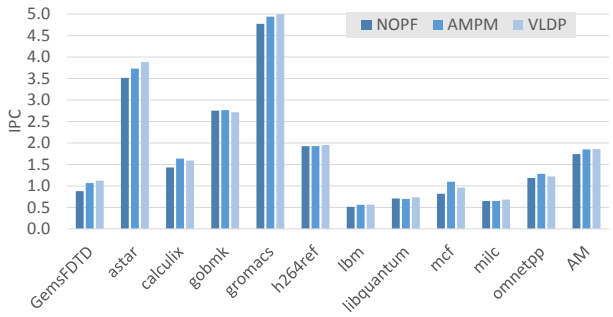

Figure 4: Mean IPC for all four Configurations

Figure 4 shows the IPC of VLDP in comparison to AMPM and the baseline (no prefetch). On average, VLDP is better than AMPM by 0.2% for all four configurations. The performance gains seen by VLDP come as a result of two factors.

First, workloads with multi-delta sequences such as milc, lbm, have long multi-delta sequences which repeat across pages. VLDP is ideally suited to handle such access patterns.

Figure 5 shows the latency of demand misses of VLDP in comparison with AMPM and the baseline for Configuration 1. The L2 miss latency is a function of L3 hit rate, DRAM row buffer hit rate and the queuing delay at the memory controller. VLDP and AMPM prefetch to the L3 cache, when the L2 read queue is nearing its maximum capacity. On average, VLDP decreases L2 demand miss latencies by 3%.

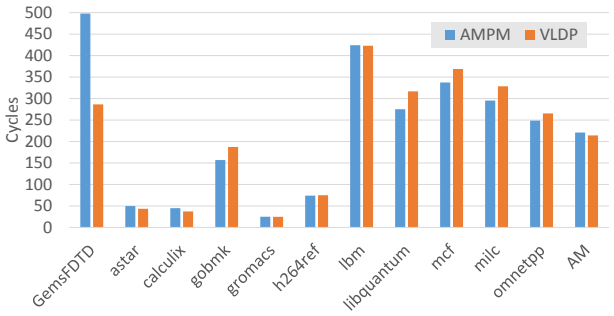

Figure 5: Demand Miss Latencies for Configuration 1

Figure 6 shows the prefetch latencies. In the given infrastructure, it is not possible to know when a cache line is filled into L3. For this reason, we only show the prefetch latency for prefetches in to L2. Overall, the prefetch latency for VLDP is less than AMPM by 89%.

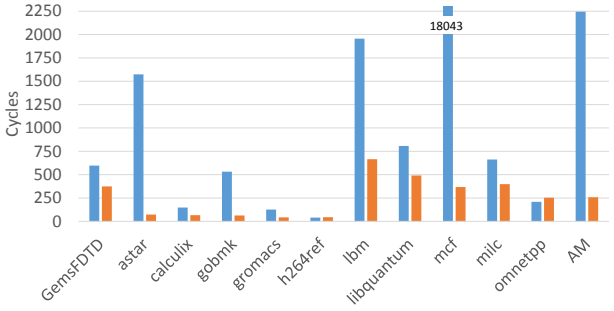

Figure 6: Prefetch Miss Latencies for Configuration 1

4. Conclusions

We've described the VLDP algorithm with several key improvements over prior work. First, it uses a delta history-based prediction scheme, allowing the prediction of complex address patterns. Second, we introduce the use of variable length histories, maximizing both prefetching coverage and accuracy. Overall, the improvement in both accuracy and coverage enable VLDP to improve overall performance by about 0.2% over AMPM. VLDP produces these benefits at a significant reduction in hardware overhead relative to AMPM and other alternatives.

References

[1] "Standard Performance Evaluation Corporation CPU2006 Benchmark Suite," http://www.spec.org/cpu2006/.

[2] 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.

[3] D. Kaseridis, J. Stuecheli, , and L. K. John, "Minimalist Open-page: A DRAM Page-mode Scheduling Policy for the Many-Core Era," in *In Proceedings of MICRO*, 2011.

[4] Ravi Hegde, "Optimizing Application Performance on Intel Core Microarchitecture Using Hardware-Implemented Prefetchers," 2008.