Towards Bandwidth-Efficient Prefetching with Slim AMPM

Vinson Young
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: vyoung@gatech.edu

Ajit Krisshna
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: ajitkrisshna@gatech.edu

***Abstract*—On chip bandwidth is a scarce resource and the performance of prefetchers can be very sensitive to the L2 cache bandwidth. Therefore any intelligent prefetcher should optimize prefetches with L2 cache bandwidth as an important metric. Furthermore, with varying workload behaviour, there is a strong case for using multiple state-of-the-art prefetchers dynamically.**

To this end, this paper proposes Slim AMPM which is a combination of two schemes. First, Slim AMPM optimizes the state-of-the-art prefetcher AMPM while taking into account L2 cache bandwidth. Second, for workloads that do not benefit from AMPM prefetcher, Slim AMPM uses a dynamic selection logic to use Delta-Correlation Prediction Tables (DCPT). Furthermore, we find that the concept of hot-pages(pages with high re-reference) in AMPM can be improved by making hot-pages more reflective of the lines being in the L2 cache or the last level cache (LLC), to better handle cache pollution. Additionally, we propose policies to better refresh AMPM hot-pages for longer workloads. These optimizations provide a speedup of 19.5% and 3.3% over no prefetching and AMPM with the best prefetch degree, respectively, for multiple configurations.

I. Introduction

In the age of very large data sets[1] and complex workloads, designers strive to improve performance by using intelligent data prefetching. Since a prefetcher moves data inside the cores alongside demand requests, the on-chip bandwidth of the system needs to be efficiently utilized. Furthermore, designers must ensure that prefetch requests do not pollute caches and increase the latency of demand requests. To make matters worse, with an increase in the number of cores, the on-chip bandwidth and caches become scarce resources. The goal of this paper is to provide insights into the merits of co-optimizing prefetches by taking into account on-chip bandwidth and cache pollution. To this end, we propose Slim AMPM, a hybrid of two state-of-the-art prefetching techniques, AMPM[2] and DCPT[3]. Slim AMPM co-optimizes for L2 cache bandwidth and Last Level Cache pollution.

A. *Efficiently Handling On-Chip Bandwidth*

A Miss Status Holding Register (MSHR) is a structure that keeps track of pending requests. Prefetch requests consume bandwidth by queuing into the MSHR while accessing the next level of the memory hierarchy. Unfortunately, the size of MSHR is fixed, and therefore, MSHRs must ideally queue only the most useful prefetches. However, not all prefetches are useful, and queuing wasteful prefetches inefficiently consumes on-chip bandwidth. Furthermore, in the limiting case wherein the MSHR is filled with wasteful prefetches, demand requests cannot occupy the MSHR, thereby stalling execution. Therefore, on-chip bandwidth can be handled efficiently by developing policies that handle prefetch access placement into MSHRs.

Based on the access pattern, prefetchers can be classified into regular access and irregular access prefetchers. Most current state-of-the-art prefetchers do not account for bandwidth and hence do not manage the MSHR efficiently. Address Map Pattern Matching (AMPM) [2] is one of the state-of-the-art regular access prefetcher that falls into this category. AMPM keeps track of previous accesses to lines in “hot-pages” to find access patterns, such as strides. This enables AMPM to determine the number of cache lines (stride) to skip before prefetching a useful cache line based on history. However, AMPM checks all possible stride lengths within a “hot-page” at once, and, as a result, it can waste bandwidth and increase cache pollution by sending bad prefetches when accesses are not following those stride patterns. Thus, AMPM often has high coverage but low accuracy, which can hurt bandwidth. Furthermore, a “hot-page” tracked too long may not be indicative of what is inside the cache, which can cause wasteful and incorrect stale prefetches. To handle MSHRs queues efficiently, Slim AMPM configures for the bursty bandwidth usage and takes into account stale prefetches.

As another drawback, AMPM does not prefetch well for irregular access patterns. Due to this, workloads with irregular access patterns will waste on-chip bandwidth if they employ AMPM. Ideally, we want to be able to prefetch for both the regular and irregular patterns efficiently. This paper proposes a bandwidth-efficient hybrid of a regular access-pattern prefetcher AMPM with an irregular access-pattern prefetcher, Delta Correlating Prediction Tables [3], to handle both access patterns effectively.

B. *Mitigating Cache Pollution Due to Wasteful Prefetches*

Demand request can be serviced quicker if they encounter higher cache hits. Unfortunately, a prefetcher places its data in the cache alongside useful data. Due to this, wasteful prefetch requests can cause useful lines to be evicted from the cache. Eviction of useful cache lines and replacing it with wasteful prefetched lines is called cache pollution. While it is possible to change the number of prefetches issued (aggressiveness) based on cache pollution[2], we find that doing so is not effective. This is because such an optimization causes bandwidth to be consumed in a bursty manner, impacting performance. To this end, our paper improves on the concept of "hot-pages" in AMPM to make tracking of "hot-pages" more reflective of the lines currently in the L2 cache or the last level cache (LLC) and effectively handles cache pollution.

II. Design

We show how to design for bandwidth and cache pollution, by building on state-of-the-art regular and irregular access-pattern prefetching using DCPT and AMPM.

For regular memory access patterns, prefetching has been 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, but there appears to be a basic design trade-off between storage and effectiveness, with large storage required to achieve good coverage and accuracy.

Regular Access Prefetcher

We build upon the current state-of-the-art regular access-pattern prefetching algorithm, called Access Map Pattern Matching (AMPM). AMPM operates by keeping access history of a large number of pages at once and searching for all possible strides at once. AMPM keeps access history by dividing the memory address space into memory regions of a set size, called "zones." AMPM then allocates one or two bitmaps for each "zone," with each bit corresponding to a cache line covered by the zone, to keep track of which lines have been accessed. To reduce space requirements, it only keeps access history of recently accessed zones, called "hot zones." The AMPM prefetcher employs two key components: a memory access map and pattern matching logic.

Memory Access Map: The memory access map is an indexed structure that keeps access-history information for each "hot zone." For each "hot zone," a bitmap records which cache lines inside the "zone" have been *accessed*, and another bitmap records which cache lines inside the "zone" have been *prefetched*. AMPM uses this access history to predict strides. But if the access map becomes stale, the prefetches become less useful. As such, this structure should be kept up-to-date.

To keep storage requirements low and keep memory access map up-to-date, AMPM keeps track of only a set number of "hot zones" at once. If a memory access occurs in a new "zone" arrives, an older "zone" is evicted to make room to track access history for the new "zone." This process of eviction has the benefit of refreshing the access map every so often.

Pattern Matching Logic: The pattern matching logic is a combinational logic for detecting memory access patterns from the memory locations held in the memory access map. AMPM detects strides by using the history information in the memory access map and the current access. If the current memory access is to x, and if (x - n) and (x - 2n) have been accessed previously, AMPM predicts (x + n) will be accessed. To achieve high coverage, AMPM simultaneously looks for all possible strides within the "zone" accessed.

Irregular Access prefetcher

A drawback of AMPM is that it does not handle irregular access patterns well and therefore, wastes bandwidth and increases cache pollution by sending inaccurate prefetches when accesses are not following stride patterns. To overcome this problem, an irregular access-pattern prefetcher is desired. Some of them include the Irregular Stream Buffer (ISB)[4], Spatial Memory Streaming (SMS)[5], Program Counter/ Delta Correlation (PC/DC)[6], and the more recent Delta Correlating Prediction Tables (DCPT)[3]. Although these algorithms have a superior performance compared to their regular access-pattern prefetching counterparts, they require huge storage.

One proposal to reduce storage overhead is to only record the deltas between two accesses handled by a single-table-based approach called DCPT[3]. DCPT operates by storing access-history, stored as deltas, into a table hashed by PC. If a PC hits in the DCPT, a delta is calculated based on current and previous address stored in the table, and, if deltas match, they are added to the current address and the remaining deltas to form candidate prefetches. Further, these candidates go through a filtering stage where prefetches which have been previously marked are filtered out. Additionally, any prefetches outside the current page are also ignored. Finally, the prefetches that satisfy all the valid prefetch cases will be sent to the L2/LLC.

The proposed methodology in this paper, for Slim AMPM, is a combination of a regular access-pattern prefetcher, such as AMPM, and an irregular access-pattern prefetcher, such as DCPT, in a bandwidth and pollution-efficient manner. When combined, both irregular and regular access patterns can be fetched ahead of time, with a minimal amount of storage, bandwidth, and cache pollution overhead. We explore these design parameters in the following subsections.

A. Optimizing for Bandwidth

1) For Bandwidth: AMPM: To reduce AMPMs bursty bandwidth usage and false prefetches, Slim AMPM adjusts the number of candidate strides dynamically. Through analysis by sandbox prefetching[7], we have found that strides -4 to 4 are most accurate, with -6/6 and -8/8 being accurate at times as well. By default, Slim AMPM limits candidate prefetch strides from -4 to 4, and it increases candidate strides when coverage and accuracy are low ($<90\%$ accuracy and $<50\%$ coverage). Limiting prefetch strides has the dual benefit of reducing false prefetches and reducing bursty prefetch behavior in the case of embarrassingly streaming benchmarks.

Additionally, prefetches are done to the LLC by default to reduce l2 bandwidth contention. But, in the case of high L2 MPKI or low coverage, we prefetch into the L2 cache as well to fully utilize both caches. To account for cache pollution, we adjust the threshold to insert into L2 based on L2 cache hit rate. The idea is, the lower the hit rate, the more we can prefetch to L2 without polluting the cache. Another key point is that filling the L2 MSHR with inaccurate prefetches can degrade performance. So, when the prefetch accuracy is critically low, we penalize the L2 insertion threshold to reduce the number of L2 prefetches.

2) For Bandwidth: Delta-Correlating Prediction Tables: To reduce AMPMs false prefetches in the case of random or pointer-chasing benchmarks, one can use DCPT in combination with AMPM for more accurate prefetches. We can capture both irregular and regular phases by using both AMPM and DCPT. However, when used in combination, AMPM can overshoot and fetch bad prefetches during irregular phases, unnecessarily consuming bandwidth. As such, we design our hybrid to use DCPT when it is able to prefetch, and AMPM when DCPT cannot. When configured as DCPT $\rightarrow$ AMPM as in Figure 1, Slim AMPM is able to prefetch both irregular and regular phases, while minimizing bad prefetches for pointer chases.

B. *Optimizing for Cache Pollution*

An intelligent prefetcher should also be able to account for cache pollution. One way to handle cache pollution is to adjust aggressiveness based on L2 cache hit rate. However, in the case of AMPM, adjusting aggressiveness affects bandwidth negatively by making it bursty. Instead, the concept of "hot zones[1]," in AMPM is adapted to handle cache pollution.

[1]As DPC-2 framework allows maximum zone size of a single page, we call "hot zones" "hot-pages" to more accurately show space usage.

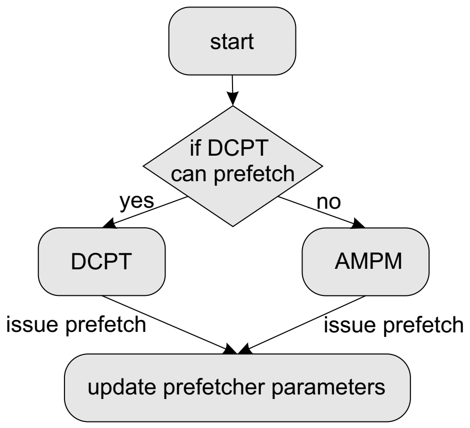

Fig. 1 Flowchart for the proposed Slim AMPM prefetcher. The backbone comprises a Hybrid DCPT $\rightarrow$ AMPM sceme.

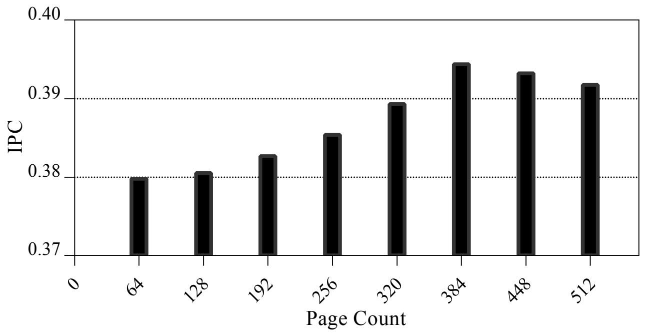

Fig. 2 IPC with different AMPM page counts for the mcf benchmark. An experiment was done to show that the IPC consistently peaks at a page count of 384.

LLC and Bandwidth Configurations
Configuration 1: L3 size is 1 MB. Memory frequency is 1600 MT/s (12.8 GB/s)
Configuration 2: L3 size is 256 KB. Memory frequency is 1600 MT/s (12.8 GB/s)
Configuration 3: L3 size is 1 MB. Memory frequency is 400 MT/s (3.2 GB/s)
Configuration 4: Configuration 1, with randomness added to the L1 access stream

TABLE I Slim AMPM is tested with multiple configurations to show applicability to other infrastructure

For reference, the ideal number of "hot-pages" to use for AMPM varies per benchmark, and this is most evidenced by mcf in Fig. 2, where the ideal page number is 384. 384 corresponds to mapping 1.5MB of space, even though LLC is only 1MB. This highlights the fact that number of "hot-pages" should not be arbitrarily increased to always hit–it should be adjusted such that "hot-pages" are more reflective of what is inside L2 or LLC.

To make "hot" more reflective of L2 or LLC, we assume a baseline of 512 fully associative AMPM "hot-pages", and decrease the number of pages used when AMPM page miss rate is high. What this means is that if AMPM page miss rate is high ($>.5\%$), the benchmark accesses many unique pages, and likely causes content in L2 or LLC to change frequently. To make AMPM "hot-pages", more reflective of the frequent changes in L2 or LLC, AMPM "hot-pages" span is reduced from 2MB (512 hot-pages), to 1.5MB (384) or .8MB (200), to refresh more frequently. Additionally, to guarantee AMPM pages are always refreshed, for longer workloads, a random replacement policy is used 1% of the time.

III. Evaluation

The prefetcher is evaluated for four configurations: 1. default, 2. small LLC, 3. low bandwidth, and 4. random access, as in Table I.

A. *Prefetcher Configuration*

The configuration of the proposed prefetcher "Slim AMPM" is shown in Section III-A. The table describes the parameters used and their respective details. Table III shows the storage overhead for the prefetcher. The two main storage components of our prefetcher are the AMPM structure and the DCPT table. AMPM is configured to use 512 pages, where each page structure contains an access map and a prefetch map vector, which are set if that particular line is accessed or prefetched respectively. An LRU status register is also present for LRU page replacement. Further, the page address takes up 52 bits, making up the storage overhead for this component close to 12096 bytes.

The second component of the storage overhead is the DCPT table which comprises a 64-bit register for the Program Counter (PC), 64 bits for the last address that was accessed, 64 bits for the last address that was prefetched, a valid bit, an LRU status register used for LRU replacement of DCPT entries, and a 9-entry "Delta" register to store the corresponding deltas between addresses. This amounts to a overhead of 12250 bytes, when added to the AMPM storage, roughly comes out to 24446 bytes, which fits within the 32KB storage budget.

Slim AMPM	Parameter	Configuration
AMPM	Number pages Replacement Policy Acc/cov metrics Candidate Prefetches Prefetch to L2/LLC Max prefetches	512, dynamically decrease use 99%LRU and 1%Random as proposed in AMPM[2] -4 to 4, dynamically increase dynamic, based on L2 hit rate 2, or 1 for low bw
DCPT	DCPT Entries Number deltas Prefetch to L2/LLC Max Prefetches	200 9 dynamic, based on L2 hit rate 4, or 3 for low bw

TABLE II Paremeter values for Slim AMPM. These are tuned dynamically to reduce bandwidth consumption and to decrease cache pollution.

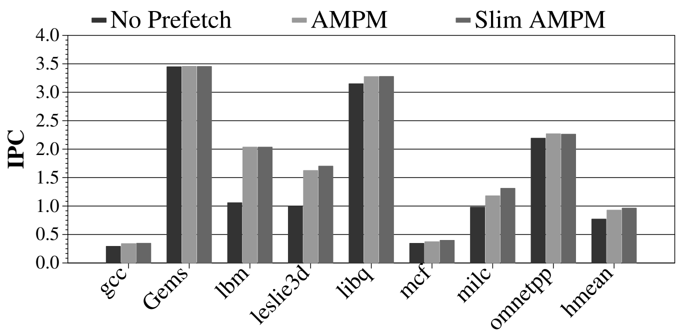

Fig. 3 IPC using three different prefetching scenarios: No Prefetch, AMPM, and Slim AMPM is shown for configuration 1. Slim AMPM performs better than AMPM in both regular and irregular workloads

B. Results

We evaluated our prefetcher with the DPC2 framework for gcc, GemsFTD, lbm, leslie3d, libquantum, mcf, milc, and omnetpp, with the 4 configurations. We have shown individual benchmark IPC for configuration 1, and aggregate IPC for other configurations in Table I. The simulator skips the first 100M instructions and evaluates the remaining instructions and produces performance statistics for every period.

The evaluation results are shown in Fig. 3 and Fig. 4. Slim AMPM performs consistently better than AMPM in almost all of the benchmarks except omnetpp, which is a network

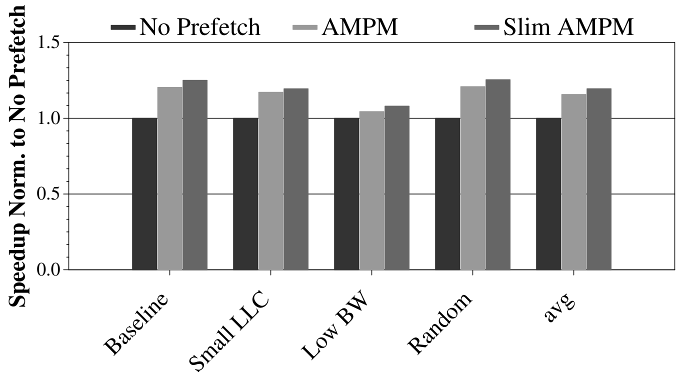

Fig. 4 Slim AMPM has a speedup of 19.5% and 3.3% compared to no prefetching and AMPM, respectively

Slim AMPM	Components	Entries	Budget
AMPM table	Access Map(1-bit) Prefetch Map(1-bit) LRU status(9-bit) Page address(52-bit)	64 64 1 1	189-bit x 512 =12096 bytes
DCPT table	Program Counter(64-bit) Previous Address(64-bit) Last Prefetch (64-bit) Valid(1-bit) LRU status(9-bit) Delta(32-bit) Assorted metrics	1 1 1 1 1 9 	490-bit x 200 =12250 bytes <100 bytes
Total			24446 bytes

TABLE III Storage requirements for Slim AMPM are within the total allotted budget of 32KB.

simulator that has sudden bursts in its access pattern. With this kind of an access pattern, the L2 cache, which commonly observes 0% hits, would sometimes experience a series of hits. In this case, AMPM prefetching with all strides i.e., -16 to 16, performs better as the L2 cache is effectively empty. However, optimizing for this bursty behavior is more likely to degrade performance for other types of access patterns, so we do not highly optimize for this case.

C. Performance

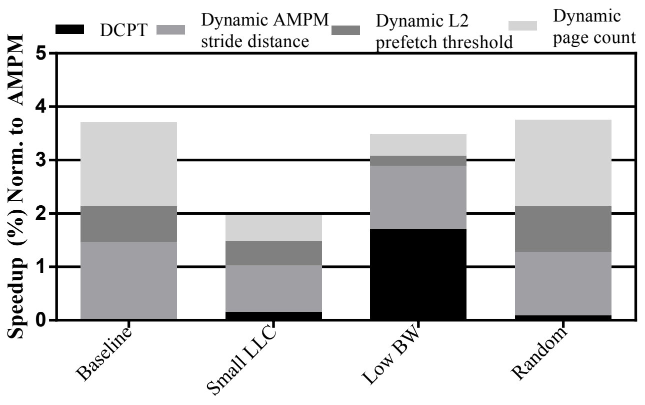

Fig. 5 Speedup(%) for the 4 configurations with the four major contributors: DCPT, Dynamic AMPM stride prefeteching, Dynamic L2 prefetch threshold, and Dynamic AMPM page count

Almost a 4% speedup was observed for default (3.9%), low_bw (3.5%), and rand_mem (3.8%) configurations because bandwidth-optimized-ampm and DCPT work well together for streaming and randomized workloads, respectively. We lower AMPM and DCPT aggressiveness for low bandwidth configurations.

A 2% speedup was observed for small LLC, because we fetch to LLC and are sensitive to LLC size. We lower number of ampm "hot-pages" to 1/4 of what is used for the default case, which is a dynamic selection of 64 (256KB), 96 (384KB), and 128 (512KB) pages.

Fig. 5 shows the breakdown of the performance impact derived from the high bandwidth and cache pollution countering strategies. Hybrid AMPM+DCPT achieved a speedup of 0.5%. Dynamically varying AMPM stride achieved speedup of 1.2%. Dynamically prefetching to L2 or L3 achieved a speedup of A 0.5%. And dynamically varying "hot-page" count achieved a speedup of 1.0%.

IV. Conclusion

Regular-access-pattern prefetchers like AMPM, by themselves do not work well for randomized workloads. They can be improved adding a history-based predictor to prefetch certain irregular accesses. DCPT offers an efficient way of keeping a track of past accesses and storing the history of patterns. Once a pattern is recognized, a number of prefetches can be sent at one time with high confidence.

Combining modern state-of-the-art prefetchers, such as AMPM and DCPT, works well for various types of workloads including streaming and random, by offering high levels of coverage and accuracy, but they can be further improved by intelligent design to reduce their overhead in terms of bandwidth and cache pollution.

We propose a hybrid solution called Slim AMPM that contains a stripped-down bandwidth-efficient version of AMPM along with DCPT, a pc-and-history-based prefetcher, to cover all types of workloads, and reduce bandwidth consumption and cache pollution while prefetching with sufficient confidence.

We evaluate our prefetcher with a 32KB budget in the DPC2 framework. The evaluation results show almost 4% increase in speedup in eight of the SPEC CPU2006 benchmarks, for three configurations, and a 2% increase for the small_llc configuration. An overall speedup of 19.5% and 3.3% was obtained over the no prefetching and AMPM respectively.

Acknowledgment

The authors would like to thank Moinuddin Qureshi, Prashant Nair, and Swamit Tannu for their input and help in this work.

References

[1] Richard L Villars, Carl W Olofson, and Matthew Eastwood. Big data: What it is and why you should care. *White Paper, IDC*, 2011.

[2] Yasuo Ishii, Mary Inaba, and Kei Hiraki. Access map pattern matching for data cache prefetch. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 499–500, New York, NY, USA, 2009. ACM.

[3] Marius Grannaes, Magnus Jahre, and Lasse Natvig. Multi-level hardware prefetching using low complexity delta correlating prediction tables with partial matching. In *Proceedings of the 5th International Conference on High Performance Embedded Architectures and Compilers*, HiPEAC'10, pages 247–261, 2010.

[4] Akanksha Jain and Calvin Lin. Linearizing irregular memory accesses for improved correlated prefetching. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 247–259, New York, NY, USA, 2013. ACM.

[5] Stephen Somogyi, Thomas F. Wenisch, Anastassia Ailamaki, Babak Falsafi, and Andreas Moshovos. Spatial memory streaming. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 252–263, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Kyle J. Nesbit and James E. Smith. Data cache prefetching using a global history buffer. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, HPCA '04, pages 96–, Washington, DC, USA, 2004. IEEE Computer Society.

[7] S.H. Pugsley, Z. Chishti, C. Wilkerson, Peng fei Chuang, R.L. Scott, A. Jaleel, Shih-Lien Lu, K. Chow, and R. Balasubramonian. Sandbox prefetching: Safe run-time evaluation of aggressive prefetchers. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on*, pages 626–637, Feb 2014.