Expert Prefetch Prediction: An Expert Predicting the Usefulness of Hardware Prefetchers

Biswabandan Panda and Shankar Balachandran,
Indian Institute of Technology Madras,
{biswa, shankar}@cse.iitm.ac.in

Abstract—Hardware prefetching improves system performance by hiding and tolerating the latencies of lower levels of cache and off-chip DRAM. An accurate prefetcher improves system performance whereas an inaccurate prefetcher can cause cache pollution and consume additional bandwidth. Prefetch address filtering techniques improve prefetch accuracy by predicting the usefulness of a prefetch address and based on the outcome of the prediction, the prefetcher decides whether or not to issue a prefetch request. Existing techniques use only one signature to predict the usefulness of a prefetcher but no single predictor works well across all the applications. In this work, we propose weighted-majority filter, an expert way of predicting the usefulness of prefetch addresses. The proposed filter is adaptive in nature and uses the prediction of the best predictor(s) from a pool of predictors. Our filter is orthogonal to the underlying prefetching algorithm. We evaluate the effectiveness of our technique on 22 SPEC-2000/2006 applications. On an average, when employed with three state-of-the-art prefetchers such as AMPM, SMS, and GHB-PC/DC, our filter provides performance improvement of 8.1%, 9.3%, and 11% respectively.

Index Terms—Hardware Prefetching, Cache, Memory systems

1 Introduction

Hardware prefetchers issue prefetch requests, which bring data into the cache before processor demands for the same. This results in improvement in system performance. However, inaccurate prefetch requests bring data into the cache, which are unlikely to be used by the processor. We call these prefetch requests *useless* and prefetch requests, which provide cache hits as *useful* prefetch requests. Since no hardware prefetcher delivers 100% of prefetch accuracy for all the applications, predicting the *usefulness* of prefetch requests is important.

The problem: Existing technique on predicting the usefulness of a prefetch address such as pollution filter [11] uses only one *signature* for prediction that works well for some applications and fail in some. Even within a single application, there are different phases of a program where a predictor that uses a single signature fails. Also, there is no predictor that is effective across all the applications.

Our goal: Our goal is to propose an efficient prefetch-address filter that can predict the usefulness of prefetch-addresses accurately and can guide a prefetcher to issue only the useful prefetch requests.

Our approach: We use multiple independent prefetch-address filters (we call *prefetch experts*) where experts predict the usefulness of a prefetch address by using their own *signature*s (such as program counter (PC) of a prefetch request or the upper bits of an address of a prefetch request). For a given prefetch address, we collect predictions from each of the experts and apply a modified version of weighted-majority algorithm [4] to find out the best prediction decision (to prefetch or not) for an application. The contributions of this manuscript are as follows:

- We propose a prefetch-address filter called weighted-majority (WM) filter that uses multiple independent experts to predict the usefulness of a prefetch address.
- We show the effectiveness of our filter on 22 SPEC-2000/2006 applications. On an average, when employed with the state-of-the-art AMPM prefetcher, WM-filter delivers a prefetch-accuracy of 0.85, which translates into 8.1% improvement in IPC.

2 Background and Motivation

This section provides the necessary background to understand the usefulness of a prefetcher. A hardware prefetcher exploits various cache access patterns and prefetches data into the cache. But a prefetcher can become less effective if the prefetched blocks[1] do not get hit at the cache. The performance can drop further if prefetched blocks evict useful demand blocks[2] causing cache pollution.

Prior techniques on predicting the usefulness of a prefetch address, such as pollution-filter [11], use a filter to predict the prefetch addresses that are not likely to get demand hits. Based on the outcome of the predictor the prefetcher decides whether or not to prefetch. The filter uses a hardware table of 4096 entries indexed by a signature (hashed using PC or prefetch-address) and each entry consists of a 2-bit saturating counter.

Manuscript submitted: 12-Mar-2015. Manuscript accepted: 20-Apr-2015. Final manuscript received: 22-Apr-2015.

1. Cache block containing prefetch response.
2. Demand blocks that might get re-referenced in the near future.

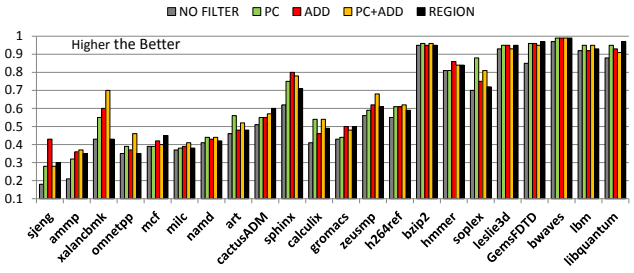

Fig. 1. Prefetch accuracy of different filters.

The filter works as follows: when a cache controller inserts the prefetched block into the cache, it sets two additional bits per block - a *prefetch-bit* to distinguish a prefetched block from a demand block and a *reference-bit* to find out whether a prefetched block is referenced by the processor or not. For an entry, the corresponding 2-bit saturating counter gets incremented whenever a prefetched block is evicted with its *reference-bit* set to 1 and gets decremented whenever a prefetched block is evicted with its *reference-bit* set to 0.

Figure 1 shows the effectiveness of different filters applied on the state-of-the-art AMPM prefetcher [3]. We place the AMPM prefetcher beside the last-level-cache (L2), which prefetches data from DRAM into L2. We use four different filters and evaluate their effectiveness. The filters explored are as follows: PC-based filter (PC), prefetch-address-based filter (ADD) [11], memory region based filter (REGION)[3] and a combination of PC and pref-address (PC+ADD). We create PC+ADD by performing the bitwise OR of ADD and PC of a prefetch request. Unlike the ADD, the REGION filter exploits the spatial locality among the prefetch addresses and makes prediction.

From Figure 1, we conclude that no single filter outperforms all others across all the applications. PC filter performs best in 3 applications. Similarly PC+ADD filter provides best accuracy for 8 out of 22 applications. ADD and REGION filters outperform others for *sjeng* and *mcf* respectively. This behavior motivates us to propose a prefetch-address filter that can perform best across all the applications. Note that our aim is not to propose a new prefetching/filtering algorithm.

3 Weighted-majority Filter

This section describes our prefetch-address filter, which we call weighted majority (WM) filter. WM-filter uses the weighted-majority algorithm to predict the usefulness of a prefetch address. First, we describe the basic weighted majority algorithm. Next, we explain how we use the weighted-majority algorithm to build the WM-filter.

Weighted-majority algorithm: This is an algorithm that is used for binary prediction (which predicts 1 or 0). It uses the predictions of n experts where each *expert* is an independent predictor. Each expert is assigned a weight (w) that corresponds to the expert's confidence (higher the better). Initially, the w of each expert is assigned to 1. After every prediction made by *expert* (say i), the algorithm updates the weight of expert i as follows: $w_i \leftarrow w_i \times (\alpha)$ if the prediction made by the expert is incorrect and $w_i \leftarrow w_i \times (\frac{1}{\alpha})$ if the prediction made by the expert is correct, where α is the *learning rate* that ranges from 0 to 1, and determines the rate at which the weights are updated.

The rationale behind such a decision making is, an expert, which predicts correctly gains its weight but an expert that predicts wrongly loses its weight based on the value of α.

The algorithm uses the prediction of each expert and determines the *majority* in the following way: It finds $w_y = \sum w_{i,p=1}$ and $w_n = \sum w_{i,p=0}$, where $w_{i,p=0}$ is the weight of an *expert* that predicts 0 and $w_{i,p=1}$ is the weight of an *expert* that predicts 1. If $w_y > w_n$, then the algorithm predicts 1 else 0. This algorithm guarantees that the prediction accuracy will be at-least as good as the best expert (expert with highest prediction accuracy).

The basic WM algorithm works well in practice but when we apply it for predicting the usefulness of a prefetch address, we find some limitations: (i) it is very slow to adapt if the *best-expert* changes over time. For example, in *omnetpp*, PC-filter performs better than others for some part of the program whereas ADD-filter performs better for some. Also, a phase-change in an application can decrease the weight of the best-expert if it predicts wrongly and the recovery will take more time. To make the WM algorithm more adaptable, we modify the algorithm and update the weight of an *expert* that makes wrong predictions, as before, but only if its weight is at least γ times of the average weight of all experts. *Rationale: the lower bound on the weight of each expert will prevent any one expert from making repeated wrong predictions for a longer duration. Also, if an expert becomes best-expert for a shorter period of time, the algorithm will be able to recognize it.* Further, we consider a lower limit (ϵ) on weights to prevent the underflow of floating point weights. WM algorithm with ϵ works equally well when compared with the basic WM algorithm [2]. Please note that this approach is different from selection based approaches where only one predictor is selected from a group of predictors.

WM-Filter: To predict the usefulness of a prefetcher, we use WM algorithm and create WM-filter that uses four *experts* in the form of filters: PC, ADD, REGION and PC+ADD. Similar to pollution-filter [11], we model each expert with a 4096-entry table where each entry consists of a 2-bit saturating counter. *The weight (higher the better) of an expert corresponds to the expert's contribution in overall prefetch accuracy of an application.*

Figure 2 shows an example of the prediction process of the WM-filter. The filter works as follows: Before issuing the prefetch request for a prefetch address, the prefetcher sends the prefetch address to the WM-filter. In ❶, the prefetch address is indexed using four different signatures to four experts (expert 1 to expert 4) based on their respective signatures. In ❷, each expert predicts

3. We use a region size of 2KB (performs best which we find empirically) which consists of 64-byte thirty two cache blocks. The region of a prefetch address = prefetch address >> 11.

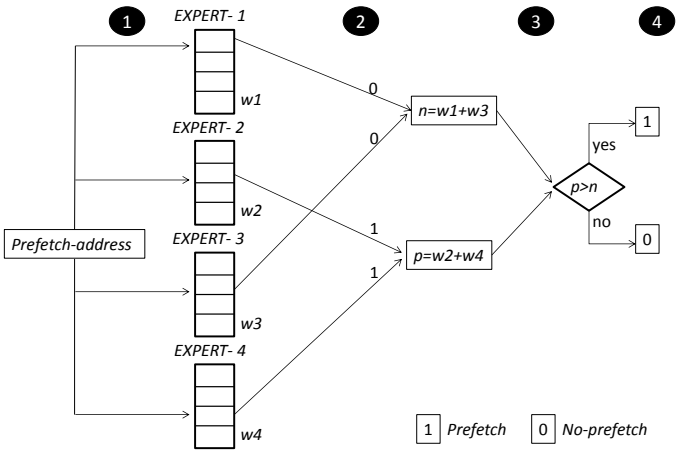

Fig. 2. Prediction process of the WM-filter.

their outcome (1 or 0). In ❸, the filter adds the weights of experts who have predicted '0' and stores it in a register called n. Similarly, it adds the weights of experts who have predicted '1' and stores it in register called p. In ❹, the filter predicts '1' and informs the prefetcher (to issue the prefetch request) if $p > n$ else with '0' (not to issue the prefetch request).

4 Implementation Details

We set α and γ as $\frac{3}{4}$ and $\frac{1}{4}$ respectively after sweeping through various values of α and γ ranging from 0.1 to 0.9. Similarly, we set ϵ to 0.1. To store the weights of each expert, we use four 32-bit registers. WM-filter also uses shifters and multipliers to update the weight, adders to add the weights and a comparator to compare them.

Each filter-table contains 4096 entries and each entry consists of a 2-bit saturating counter leading to a hardware overhead of $4 \times 4096 \times 2$ = 32.76Kbits = 4KB for four filter-tables. Similar to pollution-filter [11], each cache block contains a *prefetch-bit* and a *reference-bit*. But in contrast to the pollution filter, we do not store PC with all the prefetched blocks[4]. Instead, we use set-dueling monitors [8] and store PC with the prefetched blocks of 32 cache sets leading to a reduced hardware overhead of only 1KB. Also to store the prediction of each expert, for a given prefetch address, each cache block contains a bit-vector of width 4 bits (for 4 experts). When a cache block containing a prefetch response is inserted into the cache, the bit-vector is set to 1 or 0 based on the prediction of each expert. When a prefetched block is evicted from the cache, the 4-bit bit-vector is transferred to the WM-filter that helps in updating the weight of each expert based on their prediction and actual outcome (used or not used) of the usefulness of the evicted prefetched block.

For a 64B cache-block size, the hardware overhead with these additional bits is 12.2KB, which leads to a total overhead of 4KB + 1KB + 12.2KB = 17.6KB, which is modest (1.71% for an 1MB cache). Compared to a hardware prefetcher with one filter, WM-filter incurs an additional hardware of modest 3KB. We also find the area requirements and power consumption by using CACTI 6.5 [5] considering 32nm technology. The area overhead because of WM-filter is 0.385mm^2 with power consumption of 12.781mW, which is not significant compared to the power consumed by the LLC.

4. Storing PCs with each block increases the hardware overhead significantly.

TABLE 1
Parameters of Simulated Machine.

Processor core	out of order, 3.7GHz
Fetch/Commit width	8
Branch Predictor	Tournament
ROB/LQ/SQ/Issue Queue	192/96/64/64 entries
L1 D/I Cache	32KB, 4 way, 2 cycle latency, LRU
L2 Unified Cache	1MB, 16 way, 26 cycle latency, LRU
MSHRs	16 at L1/L2
Cache line size	64B in L1 and L2
Write Buffer	64 Entries
DRAM Controller	On-chip, Open Row, FR-FCFS
DRAM Bus	split-transaction, 800 MHz, BL=8
DRAM	DDR3 1600 MHz (11-11-11)

5 Evaluation

To study the effectiveness of our filter we use gem5 [1] full system simulator to simulate a single core system with 2 levels of cache. Table 1 shows the baseline parameters of our simulated system. We use the region of interest (ROI) of selected SPEC 2000 and SPEC 2006 benchmarks. We simulate each application for 1 billion instructions after a fast-forward and warm-up of 500 million instructions within the ROI.

We compare the effectiveness of our technique in terms of prefetch accuracy, prefetch coverage and improvement in IPC with three baseline prefetchers such as AMPM [3], SMS [9], and GHB-PC/DC [6] that do not use any filter. We provide detailed results only for AMPM because on an average, it outperforms both SMS and GHB-PC/DC. We also compare our filter with the individual-best prefetch-address filter (a filter that performs best for an individual application).

Figure 3 shows the prefetch accuracy provided by the WM filter. On an average, for AMPM, SMS, and GHB-PC/DC prefetchers, individual-best provides accuracies of 0.68, 0.51, and 0.43 respectively whereas, the proposed WM-filter provides accuracies of 0.85, 0.81, and 0.76 respectively. Our filter provides a prefetch-accuracy of more than 0.85 for 12 out of 22 applications.

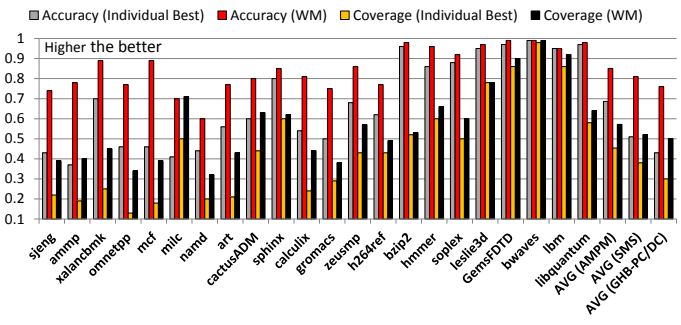

Fig. 3. Prefetch-accuracy/coverage with the WM-filter.

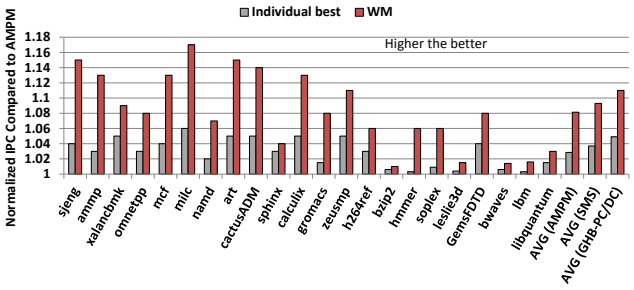

Fig. 4. Performance with the WM-filter. Note, Gmean (SMS) and Gmean (GHB-PC/DC) are the geomean of IPCs normalized to baseline SMS and GHB-PC/DC prefetchers respectively.

Figure 4 shows the performance improvement in terms of IPC. Compared to a system with baseline prefetchers without filters, WM-filter provides IPC improvements of 8.1%, 9.3%, and 11% for AMPM, SMS, and GHB-PC/DC respectively with maximum improvement of 17%. On the other hand, the individual-best provides IPC improvements of 2.8%, 3.6% and 4.9% for the same set of prefetchers.

Why WM performs better than the individual-best? We analyze the reasons behind the performance gain with the WM-filter, as follows: (i) WM-filter adapts to the phase changes of applications. For example, `milc` changes its phase once in 50M cycles causing sudden decrease in its prefetch accuracy (from 0.52 to 0.21). (ii) The approach of combining the weights of multiple filters holds the key. In case of the individual-best, if the best-filter that has been predicting correctly in the past predicts wrongly then the individual-best will make an incorrect prediction, whereas with WM-filter, it can happen that sum of the weights of other filters, that predict oppositely is higher than the weight of the best-filter. On an average, for AMPM, this happens for 23% of the total predictions.

To exemplify further, Figure 5 shows how summation of weights from the experts that predict '1'(w_y) and summation of weights from the experts that predict '0'(w_n) affect the WM-filter's prediction process. We select 20 predictions from the ROI of `omnetpp`, which spends a large part of the execution time traversing a non-linear data structure (heap), and present w_n and w_y.

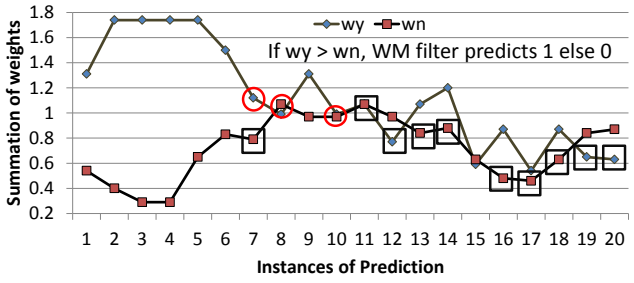

Fig. 5. Variation in weights for `omnetpp` with the WM-filter. Circle shows the mis-predictions of WM-filter and squared ones show predictions of the individual-best that are different from the WM-filter.

Out of the 20 predictions, WM-filter predicts 17 correctly. The instances of 3 mis-predictions are circled in Figure 5 and the instances where individual-best differs from the WM-filter are shown in squared boxes.

During the first 6 instances, both individual-best and WM-filter predict 1. At the 7^{th} instance, the WM-filter mis-predicts whereas the individual-best predicts correctly. From 11^{th} to 14^{th}, and 16^{th} to 20^{th} instances, WM-filter predicts correctly but individual-best mis-predicts. During these instances, PC+ADD, which is the best-filter till the 7th instance starts mis-predicting and affects the decisions of the individual-best. On the other hand, the WM-filter adapts efficiently, predicting correctly.

Effect of learning rate on performance: The learning rate of a filter depends on α (rate of update). When α becomes less than 0.35, the WM-filter provides a performance improvement of 3.9%. When α is close to 1, the WM-filter delivers better performance achieving the best when α is 0.75.

Other insights: ϵ and γ play an important role in determining the effectiveness of the WM-filter. A WM-filter without ϵ and γ provides an IPC improvement of 3.7%, 2.8%, and 2.7% for SMS, AMPM, and GHB-PC/DC respectively. Further, the number of experts also play an important role. More the number of experts, the better would be the accuracy of the WM-filter.

6 Related Work

The most closely related work is Pugsley et al.'s sandbox prefetching [7] that uses a bloom filter to test the effectiveness of a prefetch request by using 16 candidate prefetchers. Every time a prefetch request is issued, the filter predicts whether the prefetch-address would be useful, and by checking the accuracy of the corresponding candidate, it determines the number of prefetch requests to be issued. One of the limitations of the sandbox prefetching is that it predicts the usefulness of a prefetch address by observing only the past history of the prefetch address (one expert, in contrast to four experts of WM-filter).

7 Conclusions

In this manuscript, we have presented an effective WM-filter, a prefetch-address filtering technique that uses weighted-majority algorithm to predict the usefulness of prefetch addresses. The main advantage of using the WM-filter is its adaptive nature in making predictions. Our results show that WM-filter performs well across a wide variety of applications.

References

[1] Binkert et al.The gem5 simulator, SIGARCH CAN, 2011.
[2] Bianchi et al., How to use expert advice, Journal of ACM, 1997.
[3] Ishii et al., Access map pattern matching for high performance data cache prefetch. In Journal of Instruction-Level Parallelism, 2011.
[4] Littlestone et al., The Weighted Majority Algorithm. Information and Computation, 108(2):212-261, Feb.1994.
[5] Muralimanohar et al., CACTI 6.5.
[6] Nesbit et al., Data cache prefetching using a global history buffer, IEEE Micro, 2005.
[7] Pugsley et al., Sandbox Prefetching: Safe run-time evaluation of aggressive prefetchers, in HPCA, 2014.

[8] Qureshi et al., Adaptive Insertion Policies for High Performance Caching, in ISCA, 2007.
[9] Somogyi et al., Spatial Memory Streaming, in ISCA 2006.
[10] SPEC CPU2006, SPEC CPU2000, http://www.spec.org.
[11] Zhuang et al., A hardware-based cache pollution filtering mechanism for aggressive prefetches, In ICPP, 2003.