Extended Histories: Improving Regularity and Performance in Correlation Prefetchers

R Manikantan†, R Govindarajan†, Kaushik Rajan‡
†Indian Institute of Science, Bangalore, India
‡Microsoft Research India
{rmani,govind}@csa.iisc.ernet.in, krajan@microsoft.com

ABSTRACT

Data Prefetchers identify and make use of any regularity present in the history/training stream to predict future references and prefetch them into the cache. The training information used is typically the primary misses seen at a particular cache level, which is a filtered version of the accesses seen by the cache. In this work we demonstrate that extending the training information to include secondary misses and hits along with primary misses helps improve the performance of prefetchers. In addition to empirical evaluation, we use the information theoretic metric entropy, to quantify the regularity present in extended histories. Entropy measurements indicate that extended histories are more regular than the default **primary miss only** training stream. Entropy measurements also help corroborate our empirical findings.

With extended histories, further benefits can be achieved by triggering prefetches during secondary misses also. In this paper we explore the design space of extended prefetch histories and alternative prefetch trigger points for delta correlation prefetchers. We observe that different prefetch schemes benefit to a different extent with extended histories and alternative trigger points. Also the best performing design point varies on a per-benchmark basis. To meet these requirements, we propose a simple adaptive scheme that identifies the best performing design point for a benchmark-prefetcher combination at runtime.

In SPEC2000 benchmarks, using all the L2 accesses as history for prefetcher improves the performance in terms of both IPC and misses reduced over techniques that use only primary misses as history. The adaptive scheme improves the performance of CZone prefetcher over Baseline by 4.6% on an average. These performance gains are accompanied by a moderate reduction in the memory traffic requirements.

Categories and Subject Descriptors

C.1 [**Computer Systems Organization**]: Processor Architectures

HiPEAC 2011 Heraklion, Crete, Greece

1. INTRODUCTION

Timely access to data is a key performance bottleneck in modern processors. The performance of memory subsystem can have a significant impact on the overall performance of the system [14]. Prefetching [2, 6, 8, 9, 11, 12] is an effective mechanism to hide the memory access latency.

The goal of any prefetcher is to get data that might be accessed in the near future into the cache before the actual request is made by the processor. Prefetchers accomplish this by learning from the history of memory accesses[1] to predict future references. At a high level, all hardware prefetchers can be conceptually viewed as shown in Figure 1. The prefetcher tries to learn from the input stream (past access stream) and looks for certain repetitive behavior that it can exploit. For instance, a prefetcher can identify and exploit a constant stride in memory accesses [6]. More sophisticated prefetchers such as the delta correlation prefetchers [11, 12] look for Markov correlation between two consecutive entries of the input stream. In other words, prefetchers primarily look for any regularity — some repeating pattern — that might exist in its training stream.

The input stream presented to the prefetcher is a subset of the complete memory access stream, as higher level caches filter out the hits. In addition, prefetchers typically use *primary misses*[2], lines that are requested from the next-level in the memory hierarchy, as training information. Thus, prefetchers generally do not exploit any information provided by secondary misses or cache hits at a particular cache level.

In this work, we study the impact of extending the history to include secondary misses and cache hits on the performance of prefetchers. From a performance point of view, extending the history eliminates more misses compared to using only primary misses. Extending the history can improve the effectiveness of a prefetcher only if it improves the regularity present in the training stream.

We use entropy [15], a measure of information content, to quantify the regularity present in extended histories. Ideally, a more regular history, characterized by a lower entropy, should lead to improved prefetcher performance. For all the delta-correlation prefetchers [12] studied by us, entropy measurements show that extended histories in general are

[1]Prefetchers in a last level cache.
[2]A primary miss initiates a request to the data from the cache/memory below it, while a secondary miss is a request to a line that has already been requested by a primary miss but the data has not arrived yet from the lower level of the memory hierarchy.

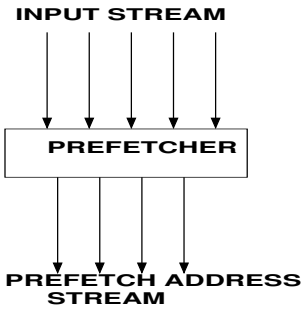

Figure 1: Conceptual View of a Prefetcher

more regular than history comprised only of primary misses. The improved regularity as identified by entropy explains the performance improvements provided by the extended histories.

Existing prefetch schemes generate new prefetches whenever a primary miss takes place[3]. Extending the history to include secondary misses and hits allows the choice of additional trigger points. As secondary misses and hits happen later in time compared to primary misses, considering them as trigger points can provide the prefetcher with a improved history compared to what was available when the primary miss happened. We observe that triggering prefetches on secondary misses along with primary misses helps in improving the performance of the prefetchers.

We study the performance of five design points, defined as a combination of *History/Trigger* for three correlation prefetchers, viz., PerPC, CZone and Global [12][4]. The history and trigger points can be a valid combination of primary misses(P), secondary misses(S) and cache hits(H). The design points studied are P/P, PS/P, PS/PS, PSH/P, PSH/PS[5]. Among these, P/P, is the default design followed currently by most if not all prefetchers. It should be noted here that extending the history of the prefetch scheme or introducing alternative prefetch trigger points requires minimal hardware modifications. Further the modifications required are agnostic to the underlying prefetch scheme.

The evaluation of the various design alternatives in terms of history and trigger reveals the following trends:

1. Extended histories provide better performance in terms of IPC and reduced number of misses compared to using only primary misses.

2. The performance of extended histories improves further by triggering prefetches on secondary misses also.

3. The best performing design point varies across the prefetchers.

4. For any particular prefetcher, the best performing design point can vary across benchmarks.

Item 3 and 4 above warrant an adaptive scheme that can choose the appropriate history/trigger point for a given prefetch scheme and a benchmark. As an initial step, we propose a simple adaptive scheme that measures the effectiveness of the various design points and picks the best performing configuration at runtime. We show that the adaptive mechanism is able to bridge a significant portion of the performance gap between P/P and the best performing design point. For the CZone prefetchers, the adaptive scheme performs better than any static design point and improves the performance on an average by 4.6% over the P/P configuration. A secondary benefit is that the improved performance is accompanied by a moderate reduction in memory traffic of 4.1%. Similar behaviour, gains in performance along with memory traffic reduction, is observed in the case of PerPC and Global delta correlation prefetchers too.

[3]Prefetch hits are treated at par with primary misses.

[4]Stride prefetchers and stream buffers can be thought of as special cases of PerPC and CZone prefetchers respectively

[5]Hits as trigger generates too many prefetch requests and degrades the performance of the prefetcher. Hence, we do not study it in detail in this paper.

2. BACKGROUND AND RELATED WORK

In this section we review some of the common prefetching strategies proposed earlier and discuss in detail the Global History Buffer *GHB* and the delta correlation prefetchers.

2.1 Prefetch Strategies

Next line prefetching [2] relies on spatial locality and a prefetch is issued to the next block in memory on a cache miss. Simple stride prefetching [6] identifies and exploits any constant stride that appears in the access stream of load instructions. In the context of stride prefetching, it is a well-known practice to consider all the accesses to learn the stride rather than using only misses. The decision to use all the accesses was validated primarily through experimental results showing improved performance. No attempt was made to quantify as to whether the extended histories improved the regularity of the training information, thereby improving the predictability of the training information used by the stride prefetcher.

Markov prefetching [8] (Address correlation) scheme records streams of addresses and uses Markov correlation to predict the most likely addresses to be accessed next, given the addresses of the past few misses. However, address correlation cannot cover the first miss to any address. The distance prefetching [8] scheme computes and records the difference between the addresses of consecutive misses and looks to identify Markov correlation patterns in it. Stream buffers [9] allow multiple streams of misses to be targeted simultaneously. It has been shown that using a correlation method, rather than next use or stride, to predict the prefetch addresses in the streams can improve the performance of stream buffer based prefetching [17].

2.2 Global History Buffer

2.2.1 Hardware

Global History Buffer (*GHB*) [12] is a hardware structure which stores prefetch history and enables a variety of prefetchers to be implemented on top of it. It consists of two key structures, the first being a *GHB*, a FIFO structure, which stores the most recent misses. Each entry of GHB also has a forward and backward pointer that can be used to point to other *GHB* entries. The second structure is an index table that along with the pointers in the *GHB* entries is used to link together misses sharing the same characteristic. Each entry in the index table stores a key and a pointer to the most recent miss suffered by that key in the GHB. For instance if the key used is the PC suffering the

cache miss, the index table entry points to the most recent miss suffered by that PC.

2.2.2 Delta Correlation Prefetching

Delta correlation prefetching [12] is one of the schemes built on top of *GHB*. Here, delta is the difference between two consecutive addresses in a stream of addresses. Thus, a stream of addresses is translated into a stream of deltas. When a new miss address is added to the prefetcher, delta correlation computes the two[6] most recent deltas in the miss stream. It then goes back in time along the stored miss stream and checks for previous occurrences of the two most recent deltas. The stream of deltas is expected to show regularity i.e. the likely sequence of deltas that are expected to follow the most recent pair of deltas will be the same as the last time. It can be observed that delta correlation prefetchers rely heavily on the regularity exhibited in their training stream.

Three variations of delta correlation prefetching have been studied and they differ in their of choice of the criterion used to link the misses together into streams. The schemes and the criteria used in them are: (i) *PerPC*(P/DC) [12]: Misses caused by the same load instruction are linked together. The key into the index table is the PC of the load experiencing the miss. (ii) *Global*(G/DC) [12]: All the misses are considered as a single stream. It is like distance prefetching without using a table to store the correlation information. The index table contains a single entry that points to the head of the GHB where the next insertion will take place. (iii) *CZone*(C/DC) [11]: A CZone is a consecutive region of memory. Addresses missing in the same CZone are linked together in C/DC. The delta correlation prefetchers have been shown to give high performance and are one of the best performing prefetchers studied so far. Hence we use these prefetchers in our study.

2.3 Prefetcher Enhancements

Aggressive prefetching [20] can fetch many memory blocks into the cache in anticipation that these lines will be accessed in the future. However, some of these lines may never be accessed or get replaced from the cache before they are accessed. These are referred to as wasted prefetches. As these unused prefetches waste valuable memory bandwidth, recent works on prefetchers are directed towards filtering the prefetch requests once they are generated [10, 18, 19, 21]. It is normally done taking into account the accuracy of prefetches. More recent methods [18] take into account a variety of parameters – accuracy, pollution and timeliness into account to throttle the prefetchers. All these schemes are agnostic to the history used by the prefetcher.

3. DESIGN SPACE EXPLORATION

3.1 Design Space of Prefetchers

We enumerate the design space of prefetchers along two dimensions, history (Hist) and trigger (Trig). As cache access results in a primary miss(P), secondary miss(S) or a hit(H), the history for the prefetcher can be any combination of these three. Earlier works typically use primary misses as history for the prefetcher. Similarly earlier works used only primary misses as trigger points to issue prefetches. However, the presence of secondary miss and/or hits along with primary miss in the history allows other trigger points as well (any combination of primary misses, secondary misses and cache hits triggering the prefetches). Among the various possible combinations, we study the following interesting design alternatives in this paper:

- *P/P:* Only primary misses are used as history to the prefetcher, and they trigger prefetches. This is the most widely used design point and we refer to it as baseline in the paper.
- *PS/P:* The history is composed of both primary and secondary misses but only primary misses are used to trigger the prefetches. This configuration allows us to identify the impact of including secondary misses in the history.
- *PSH/P*[7]*:* All the accesses seen by the cache are used as history to the prefetcher, but only primary misses trigger prefetches.
- *PS/PS*: Primary and secondary misses form the prefetch history and both of them can trigger prefetches.
- *PSH/PS*: The history is composed of primary and secondary misses and hits at the cache level. Primary and secondary misses can trigger new prefetches in this configuration.

We do not consider configurations with cache hits triggering prefetches as it results in a large number of prefetches being generated, thereby affecting the performance adversely. In all these configurations, prefetch hits[8]are always treated at par with primary misses. This is in-line with the standard practice of prefetch hits triggering further prefetches, as prefetch hits are essentially primary misses avoided due to the effectiveness of the prefetcher.

From a hardware perspective implementing extended history and primary or secondary misses as trigger requires only minimal changes to the baseline hardware. It involves modifying the input to the prefetcher and ignoring the prefetch requests generated by non-trigger points. No change is required to be carried out to the cache or the prefetcher itself. Further it can be seen that the implementation of extended history and trigger points is agnostic to the underlying prefetcher used.

3.2 Simulation Methodology

We use the validated alpha simulator sim-alpha[4] for studying the impact of history on delta correlation prefetchers – PerPC, Global and CZone. The delta correlation prefetchers are considered as they provide significant gains in performance at acceptable levels of increase in the memory traffic. We implemented the delta correlation prefetchers on top of *GHB*. We used simpoint[16] methodology to identify representative intervals of length 500M committed instructions. The benchmark suite considered is SPEC2000. We discuss results in detail for a set of 14 benchmarks which

[6]Experimental results in [12] demonstrate that using two deltas gives the best performance.

[7]PH/P behaved similar to PSH/P. Hence we do not consider PH/P as a separate design point in this study.

[8]Even prefetches that are not timely, ones that cover a part of the memory access latency only, are treated as primary misses.

Fetch/Map/Commit width	8
INT/FP Issue width	6
ROB/LQ/SQ	128/64/64 entries
INT/FP Issue Queue	96 Entries each
Branch predictor	21264's predictor, 32 Entry RAS
Caches	L1 ICache 16KB, 4way, 64B line size, 1 cycle access L1 DCache 16KB, 4way, 64B line size, 1 cycle access L2 Unified, 1MB, 32Way, 64B line size, 12 cycle access MSHR - 32 for Miss, 32 for Prefetches at each cache Minimum 400 cycles memory latency 16 Byte wide memory bus
GHB	512 entries to track the misses 512 entry Index table CZone size – 16K Prefetch Degree – 16

Table 1: Machine Configuration

have an L2 Misses Per Kilo Instruction(MPKI) greater than 2.5. We also provide summary results for the entire benchmark suite[9] wherever applicable. The machine configuration used for this study is shown in Table 1. The timing requirements of *GHB* and the associated prefetch mechanisms are also modeled faithfully. All the design points used a GHB that can store 512 addresses. The prefetch degree used in all the simulations is 16.

4. DESIGN SPACE EXPLORATION

In this section we explore the five design points for the three delta correlation prefetchers. While it is easier to evaluate the performance potential of the design points using simulation, it does not provide any insight as to whether the extended history indeed improves the regularity/predictability of the training stream observed by the prefetchers. To address this, we attempt to characterize the various histories, *P*, *PS* and *PSH*, based on their regularity.

4.1 Impact of Extending the History

We use entropy [15] as a measure for summarizing the regularity of the various histories. We demonstrate that entropy is a suitable measure by studying the correlation between the actual performance and the trends identified by entropy.

Entropy Basics

Entropy in simple terms is a measure of the information contained in a stream of symbols [15]. It is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) log_b(p(x_i)) \quad (1)$$

Here X is the entire message (or a stream of symbols) and H(X) is its entropy. Each x_i is a unique symbol in X and $p(x_i)$ is the probability of occurrence of x_i in X. We use base 2 for the logarithm. In practical terms, entropy is a measure of how hard it is to predict the next symbol that is likely to be seen in a stream of symbols. The maximum value is reached when all x_is are equally likely to occur. A high value for entropy indicates low predictability. When one of the x_is occurs with high probability, it makes the stream more predictable and it is reflected in terms of a reduced entropy value. To summarize, lower the value of entropy, more predictable the stream is. The various training streams, P, PS and PSH, can be converted to a stream of deltas[10] and their entropy can be estimated using Equation 1.

The above computation treats the entire training stream as one single entity and tries to estimate how regular it is. This is similar to the way Global delta correlation prefetcher treats the entire training stream as one single entity. But in the case of PerPC and CZone prefetchers, the single training stream is effectively split into multiple training streams, all accesses/misses from a particular PC/CZone are treated as belonging to one stream and there are as many streams as there are unique PCs/CZones. When the training stream is split into multiple streams, say one per PC, the entropy can be computed as

$$H(X) = \sum_{p \in PC} P(p).H(X_p) \quad (2)$$

Here X is the complete stream, PC is the set of all unique PCs that have contributed to at the least one access in X. X_p is the set of all accesses in X that are caused by the instruction p. P(p) is the probability that a given entry in X is from p ∈ PC. P(p) is actually computed as ($|X_p|/|X|$).

While entropy is a simple measure of regularity present in a stream of symbols, it is only an approximation of the regularity captured by the prefetchers. This is primarily because entropy operates after seeing the entire stream and it ignores the relative ordering of symbols in the stream. On the other hand the correlation prefetchers that we study rely on deltas repeating in order. Despite this limitation, we show that the regularity captured by entropy correlates well with the observed effectiveness of the various prefetchers.

Regularity Vs Performance

The entropy computations are carried out in an off-line fashion. A trace consisting of all the L2 accesses along with their hit/miss information is processed to compute the entropies of the various histories. As the prefetchers used are delta correlation prefetchers, we convert the stream of addresses (at cache block level) into a stream of deltas before carrying out the entropy computation. Here delta is the difference between successive entries of the stream. The trace also has the information regarding the load/store instruction that caused the access. This permits the history to be split according to the PC causing the access which is required for measuring the entropy for PerPC prefetcher.

Figure 2 shows the entropy of the extended histories PS and PSH used by the PerPC prefetcher normalized to the entropy of the baseline history(P). The entropies are computed for each stream of accesses/misses caused by a particular PC and they are weighted together as shown in Equation 2. As lower entropies imply more regularity, lower normalized values, less than 1.0, means extending the history makes it more predictable. Also shown (along the Y2 axis) are the number of L2 primary misses for PS/P and PSH/P normalized to that of P/P. We present the IPC and memory traffic results in Section 6.

The key motivation behind the extended histories is to improve the regularity observed in the training stream of

[9]Benchmark *fma3d* did not run in our simulation framework.

[10]Difference between the block level address of two successive entries.

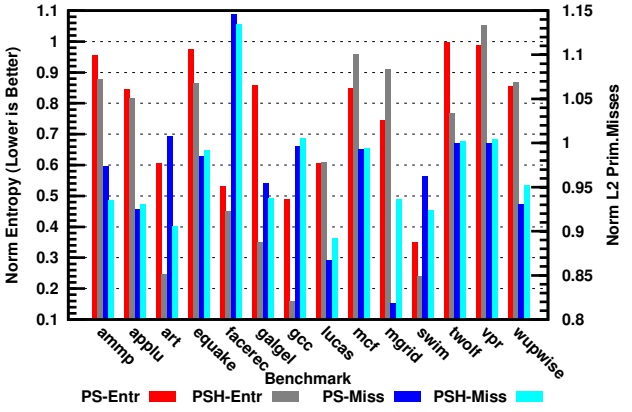

Figure 2: Entropy Vs L2 Misses for PerPC Prefetcher

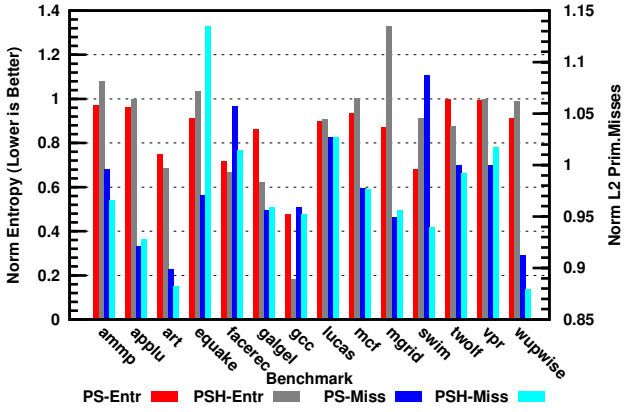

Figure 3: Entropy Vs L2 Misses for CZone Prefetcher

prefetchers and thus help eliminate more misses compared to the baseline configuration. In the case of PerPC the prefetchers using extended histories, PS/P and PSH/P, reduce the primary misses at L2 by 3.8% and 3.7% respectively over P/P. In 13 of the 14 benchmarks studied in detail, there is at the least one extended history that experiences lower misses compared to P/P, with *facerec* being the only exception. Even in *facerec*, PS/P and PSH/P suffer fewer L2 misses compared to a baseline with no prefetcher.

Ideally a reduction in the entropy should lead to a reduction in the misses, and lower the entropy more the reduction for a given benchmark. This kind of a correlation between entropy of the history and misses observed by the prefetcher is seen in a majority of the benchmarks. In benchmarks like *ammp*, *art*, *galgel* and *swim*, PSH has lower entropy than PS and both of them have entropy lower than that of P. This translates into PS/P and PSH/P suffering fewer misses compared to P/P, with PSH/P doing better than PS/P. The interesting case is in benchmarks like *lucas*, *mcf*, *mgrid*, *vpr* and *wupwise*, where entropy characterizes the PSH stream as less regular than PS which correlates well with the result that PSH/P suffers more misses compared to PS/P. Minor differences in the relative performance of PSH/P and PS/P, different to that suggested by entropy, is noted in the case of benchmarks *applu* and *equake*. This can primarily be attributed to the approximate nature with which entropy captures the behaviour of prefetchers.

Facerec and *gcc* are the only benchmarks where entropy varies from the observed behaviour significantly. In *facerec*, despite a reduction in entropy for extended histories compared to using only primary misses, PS/P and PSH/P exhibit more misses than P/P. In *gcc*, the difference with measured entropy shows up in the relative behaviour of PS/P and PSH/P. We noticed that these differences are in line with the number of prefetches issued by PS/P and PSH/P. In *facerec*, the number of prefetches issued by PS/P and PSH/P is 1.8% and 1.3% less respectively compared to the number of prefetches issued by P/P. This translates into both the design points suffering more misses than P/P with PSH/P performing marginally better than PS/P. In gcc, PS/P generates 1.1% more prefetches compared to PSH/P, and hence succeeds in eliminating more misses compared to it. In fact, PSH/P generates 0.3% prefetches less than P/P

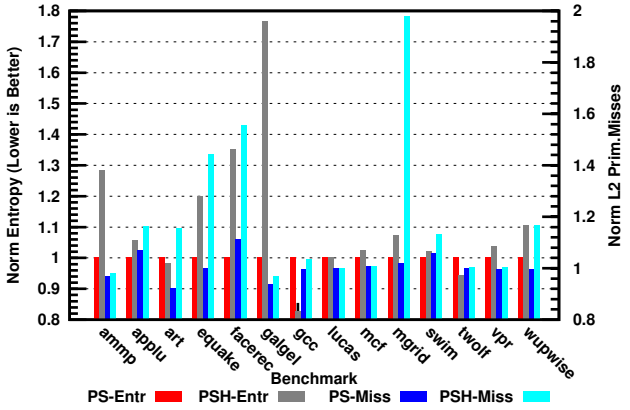

Figure 4: Entropy Vs L2 Misses for Global DC Prefetcher

and as an effect experiences marginally more misses compared to it.

Figure 3 shows the normalized entropy and misses at L2 for the CZone prefetcher. The results observed are similar to that of PerPC. PS/P reduces the misses experienced by 2.4% (2.1% for the entire suite) on average compared to P/P. PSH/P experiences on an average 3.2% less L2 misses (1.9% for the entire suite) compared to P/P. In terms of entropy and regularity, a correlation is observed in *applu*, *art*, *equake*, *facerec*, *gcc*, *lucas*, *mgrid*, *twolf* and *vpr*. Benchmark *swim* is an interesting case where despite the poor regularity exhibited by PSH, PSH/P performs better than PS/P. On analyzing this further, we realized that PSH/P issues more prefetches compared to PS/P with only a minor drop in prefetch accuracy. Differences between the relative entropies and performance is observable in more cases here but in general the trend of reduced entropy leading to better performance holds well enough. This is especially encouraging since entropy is a high-level abstract metric which only takes into account the frequency of occurrences of deltas and not the order in which they appear.

The correlation between entropy of the various histories and their performance can also be observed in the case of

Global delta correlation prefetchers as can be seen from Figure 4. On an average, PS/P suffers 0.2% more L2 misses than P/P. But PS/P suffer fewer misses in benchmarks like *ammp*, *art* and *galgel* compared to P/P. PSH/P suffers 10% more misses on average compared to P/P. Entropy does especially well in explaining the poor performance of PSH/P as the history PSH consistently exhibits higher entropy compared to others. This shows that poor performance in the case of Global history prefetchers with PSH history is primarily due to the harmful effect of all the interleaved accesses being seen as one single stream and is not due to reasons like pressure on the *GHB* structures. We verified this by studying the performance of the extended histories for Global delta correlation prefetchers using a *GHB* containing 2048 entries. Even after increasing the size of the structures by a factor 4, the difference in performance compared to the prefetchers using a smaller sized *GHB* (of 512 entries) was almost non-existent. The poor performance of PSH/P and PS/P is only in comparison with P/P. Both these design points actually performed better than no prefetching.

In general, extending the history to include secondary misses and hits results in a stream with improved regularity. The nature of the prefetcher plays a great role in determining whether extending the history is useful or not. For instance while PSH is more regular for PerPC prefetchers, they have poor regularity in the case of Global prefetchers. Also for a given prefetcher, the impact of extending the history can vary across the benchmarks. For instance, PSH history improves the regularity of *ammp* in PerPC prefetchers while it exhibits poor regularity in *mgrid*.

4.2 Triggers Vs Performance

Off-chip memory access normally takes hundreds of processor cycles. Further, primary and secondary misses are separated in time. Hence it is possible that the cache had experienced further misses since the primary miss took place and there can be an improved history present by the time the secondary miss takes place. This is one of the reasons to consider secondary misses as trigger points. In PerPC prefetchers, the secondary miss can be experienced by a different PC compared to the one that caused the primary miss. If the PC experiencing the secondary miss has a more regular history, it might be beneficial to trigger prefetches when the secondary miss happens.

Figure 5 shows the L2 misses of PS/P, PS/PS, PSH/P, PSH/PS normalized with respect to P/P for the PerPC prefetcher. The positive impact of triggering prefetches on secondary misses can be seen from the improved performance of PS/PS and PSH/PS compared to PS/P and PSH/P respectively. PS/PS reduces the misses seen in P/P by 8.8%. PSH/PS, the best performing configuration, experiences 10% less misses compared to P/P. In all the benchmarks except *facerec*, triggering prefetches also during secondary misses helps in eliminating more misses. The poor performance in *facerec* is primarily due to the lower prefetch accuracy[11] exhibited by PS/PS and PSH/PS compared to PS/P and PSH/P respectively. Overall, though PSH/PS eliminates more misses compared to the other design points, the best performing design point varies on a benchmark by benchmark basis. For instance, PS/PS is the best performing configuration for *applu*, *equake* and *mgrid*. And P/P happens to be the best performing configuration for *facerec*.

[11]Fraction of useful prefetches among the total prefetches.

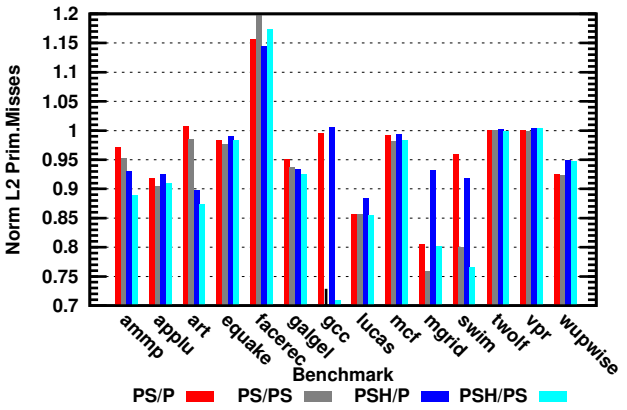

Figure 5: Normalized L2 Misses for PerPC Prefetcher

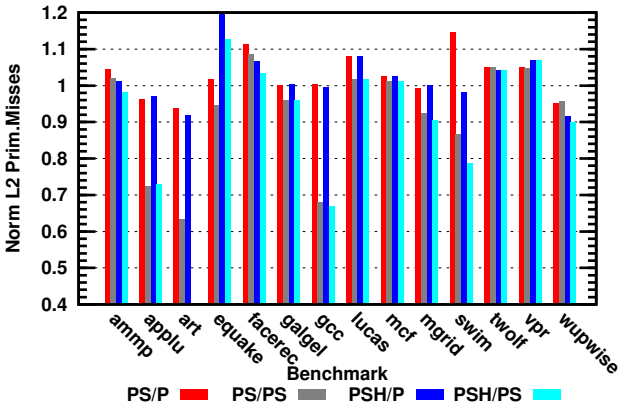

Figure 6: Normalized L2 Misses for CZone Prefetcher

The performance of the various design points in terms of the L2 misses for the CZone and Global prefetchers are shown in Figures 6 and 7 respectively. In CZone, PS/PS and PSH/PS experience 15.5% and 22% fewer misses compared to P/P. For the Global prefetchers, PS/PS eliminates 13% more misses compared to P/P. Even PSH/PS unlike PSH/P suffers fewer misses, 2.7% less, compared to P/P.

In general triggering prefetches on secondary misses proves beneficial in all the prefetchers studied by us. But the best performing configuration varied on a per benchmark basis for any given prefetcher.

5. ADAPTIVE SCHEME

From the previous sections, it can be observed that extending the history to include secondary misses and hits can improve the regularity observed in it, thereby eliminating more misses compared to using only primary misses. But the right history to use varies from prefetcher to prefetcher. Even for the same prefetcher, different benchmarks favoured different histories. Similar trends are observed in the case of secondary misses triggering prefetches too. In effect, the observed trends make a case for attempting to identify and use the right configuration, in terms of history and trigger

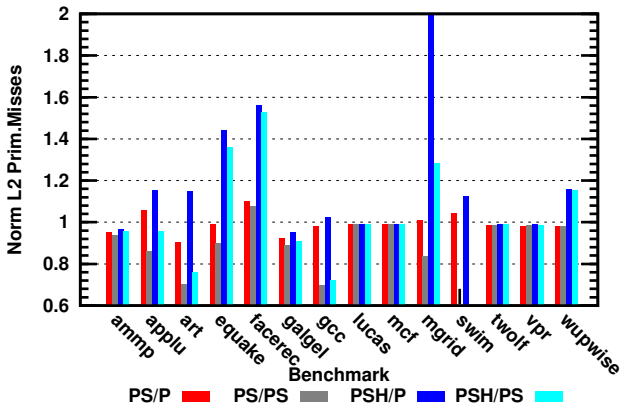

Figure 7: Normalized L2 Misses for Global DC Prefetcher

points, at runtime. We propose and evaluate one such attempt in this section.

The primary requirement of any adaptive scheme is to be able to store the three different histories – P, PS and PSH. In this work, we use three separate *GHB*s (each containing 512 entries) to store the three different histories used. Though this increases the storage requirements of *GHB*, they do not have any adverse impact on timing requirements as they are situated off the critical path. It is possible to have one single monolithic *GHB* with a set of backward and forward pointers, one set of pointers for each history. As this would require significant hardware changes to the well understood and easy to implement *GHB* structure, we plan to pursue this aspect as part of future work.

The best design point for a benchmark-prefetcher combination is one that is able to eliminate more misses. In other words, on a primary miss, the adaptive scheme should have the ability to identify the design points that could have avoided the miss. The problem can be viewed as set membership testing. Whenever a primary miss happens, we need to ascertain as to whether any of the design points generated a prefetch for it in the past. We use a set of bloom filters [3], one per design point, to approximately represent the set of prefetch addresses generated by each of the design points. Bloom filters have been used for similar purposes in many earlier studies [1].

A bloom filter is essentially a bit vector and a set of hash functions that converts set elements into indices that can be used to index the bloom filter. When a new element is to be added to the set, the hash functions are applied to the new element and a set of index locations are arrived at. The bits at the computed index locations of the bloom filter are set to 1 to mark the insertion of the new element. Checking for set membership involves index computation and verifying as to whether all the index locations are set. As bloom filter is an approximate representation, they can sometimes say that an element has been inserted while it has not been. Such instances are called false positives. Multiple hash functions are used to reduce the false positive rates. It is important to note that a bloom filter does not provide false negatives – when it says an element has not been inserted, it is indeed the case.

In our proposed adaptive scheme, we use one bloom filter per design point to keep track of the set of addresses that would have been prefetched by that design point. Each bloom filter is made up of 2048 bits (256 Bytes in size). Three hash functions are used to index the bloom filter. The hash functions used in this study are simple ones, which pick the first 11, last 11 and middle 11 bits of any prefetch address[12]. As the hash functions just pick consecutive chunks of bits, they are inexpensive to implement. There are two functional aspects to the adaptive scheme, (i)Update – which deals with the mechanisms behind the storage of the set of prefetches generated by the various design points and (ii)Analysis – which makes use of the stored information to identify the best performing prefetcher design point.

Analysis: Whenever a primary miss happens, the bloom filters for all design points are queried for membership using the address (at block level). A positive response indicates that the miss could have been covered by the corresponding design point. Note that this is an over-estimation of the benefits as bloom filters can give some false positives. A set of counters, one per design point, is used to keep track of the number of misses covered by the various design points. On a successful response from the bloom filter, the corresponding counter is incremented by one. We observe the behaviour of the design points over a period of time (typically 16K misses at L2 in our simulation), and at the end of the interval, identify the best performing prefetcher in that interval. This is achieved by checking the counter values and choosing the design point that covered the maximum number of misses. All the bloom filters and the counters are cleared and the next observation interval is started. In case if all the bits in any bloom filter are set before the end of the interval, the interval is terminated immediately, the best performing prefetcher is identified, the bloom filters are cleared and the next observation interval is started.

Update: On a primary miss, all the three *GHB*s are to be updated. While only two, corresponding to PS and PSH, are to be updated on a secondary miss. A hit updates only the *GHB* corresponding to PSH. Primary misses and secondary misses also act as prefetch trigger points for one or more of the design points. When a design point generates prefetches, it uses the generated addresses to update the corresponding bloom filter. Also if the design point had been identified as the best performing design point in the previous interval (as explained in *Analysis* above), then the prefetch requests are also sent to the main memory to start the data access.

Instead of using a bloom filter, it is possible to use other alternatives like duplicate tags coupled along with mechanisms like SBAR [13]. The principal behind all these approaches is the same and comparing the merits of alternative ways to identify the best performing design point is beyond the scope of this paper and is left to future work.

6. PERFORMANCE RESULTS

In this section, we look at the performance in terms of IPC and memory traffic of the various design points and the adaptive scheme. The behaviour of the various design points in terms of their ability to eliminate misses has been presented earlier. We study the behaviour of the adaptive scheme in detail in the context of CZone prefetchers.

The P/P configuration of the CZone prefetcher improves

[12]The addresses are at cache block size i.e. the bits used to index into a block are ignored.

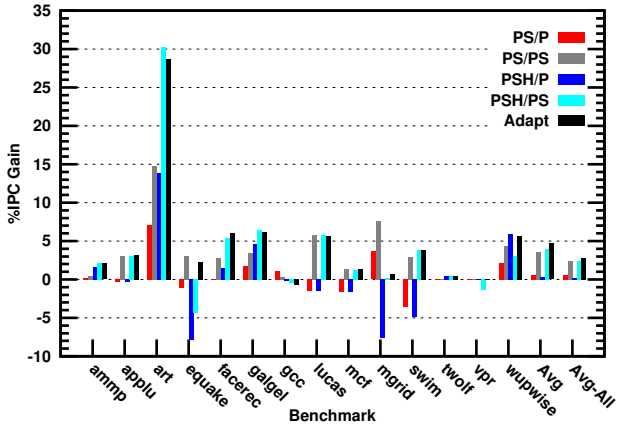

Figure 8: IPC Gain over P/P for CZone Prefetcher

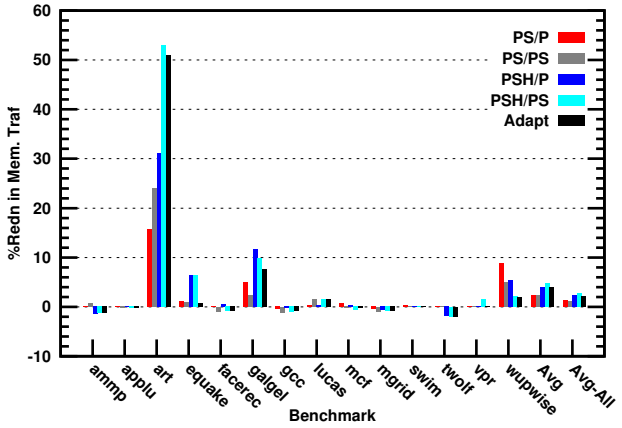

Figure 9: CZone: Memory Traffic Reduction Over P/P

the performance in terms of IPC by 81% (53% for the entire suite) on an average over no prefetching for the 14 benchmarks studied in detail. The corresponding gains for PerPC and Global prefetchers are 88% (58% for the entire suite) and 48% (33% for the suite) respectively. These significant gains are in line with earlier studies observing the high performance provided by the delta correlation prefetchers. Figure 8 shows the IPC gains obtained by PS/P, PS/PS, PSH/P, PSH/PS and the adaptive design over P/P prefetching for CZone prefetchers. The design points PS/P, PS/PS, PSH/P and PSH/PS show on an average [13] 0.5%, 3.6%, 0.3% and 4.0% IPC gains over P/P. The performance trend observed in the various benchmarks is in line with the reduction in misses observed earlier for the various design points. Significant reduction in misses observed in cases like *art*, translates to higher gains in performance. PSH/P shows a performance loss in *equake*, *mgrid* and *swim*. Entropy identified the PSH stream to be less regular in these benchmarks. While *equake* suffers due to more misses at L2, *mgrid* and *swim* lose performance due to a drop in the accuracy of prefetches generated by PSH/P compared to P/P.

[13]We use arithmetic mean in this paper to compute the average.

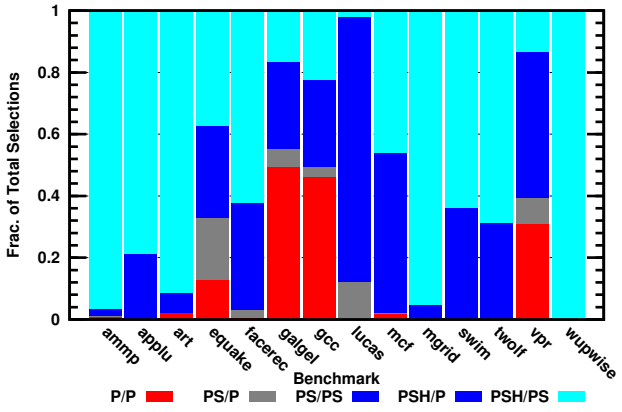

Figure 10: CZone Adaptive Scheme: Fraction of times various designs are selected

In 13 out of the 14 benchmarks, the adaptive scheme always performs better than P/P even though some of the design points lose performance compared to P/P. The adaptive scheme improves the IPC on an average by 4.6% over P/P(2.8% for the entire benchmark suite). The best performing static design point PSH/PS managed to improve the performance by 4.0% (2.3% for the entire suite). The adaptive scheme also reduces the L2 misses seen in the P/P by 22.3% (38% is the average for the entire suite). All of these leads to a secondary benefit of a moderate reduction in the memory traffic requirements compared to P/P by 4.1% (2.1% for the entire suite). The reduction in memory traffic is shown in Figure 9.

In most of the cases, the performance of adaptive scheme is close to that of the best performing design. In benchmarks like *facerec*, the adaptive scheme manages to outperform all the individual design points. Figure 10 shows the fraction of times the various designs are identified and selected as best performing configuration for the benchmarks studied in detail by us. It can be seen that where there is one clear best performing configuration like *art*, the adaptive scheme also picks the best performing configuration over 90% of the time. In *facerec*, alternating between the two best performing configurations, PS/PS and PSH/PS, helped the adaptive scheme to perform marginally better than the best performing design point. In future work, we plan to explore proactive adaptive schemes that can capture the performance impact of the prefetchers along with estimating the reduction in misses.

Figure 11 shows the IPC gain obtained by the various design points and the adaptive scheme for the PerPC prefetcher over P/P. PS/P, PS/PS, PSH/P and PSH/PS improve IPC on an average by 0.7% (0.2% for the entire suite), 1.9% (1.2%), 1.0%(0.3%), 2.5%(1.5%) respectively over P/P. For a majority of the benchmarks, the performance trends observed are in line with the reduction in misses observed earlier. The only exceptions being *mcf*, *swim* and *wupwise*, where some of the prefetch opportunities are lost due to non-availability of free MSHRs. The adaptive scheme provides an IPC gain of 1.7% (1.8% for the suite) over P/P. The L2 misses are reduced by 9.7% (32.8% for the entire suite) compared to P/P. As a positive side-effect, memory traffic reduces by 1.4% (0.6% for the suite) compared to

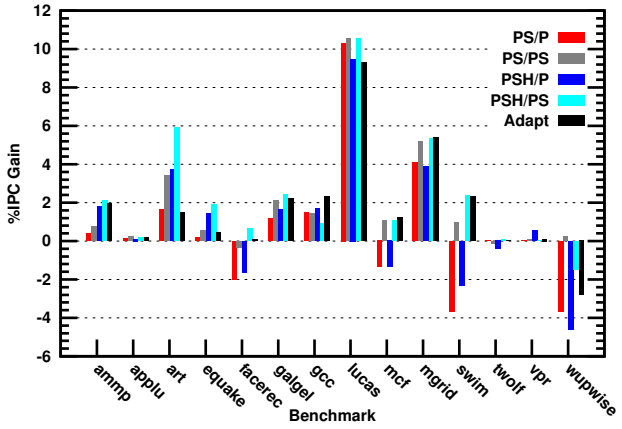

Figure 11: PerPC Prefetcher: IPC Gain over P/P

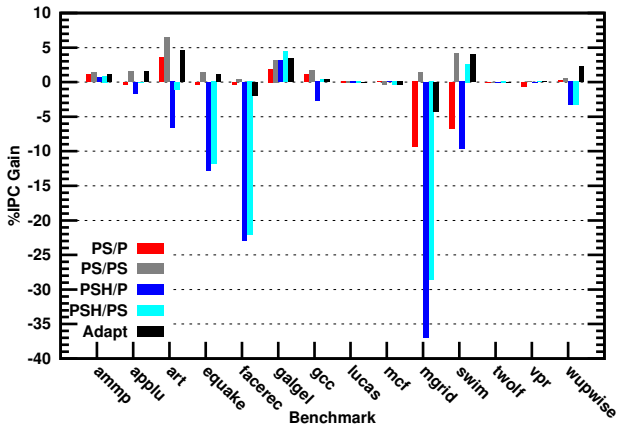

Figure 12: Global: IPC gain over P/P

P/P. The trend of adaptive scheme performing better than any design point (in *gcc*, *mcf* and *mgrid*) or as close to the best performing design holds for all the benchmarks except *wupwise*.

The results for the Global prefetcher are shown in Figure 12. While PS/P shows a minor loss of 0.7% on an average compared to P/P, PS/PS manages to improve the performance of P/P by 1.5%. This behaviour is similar to the other prefetchers where triggering prefetches on secondary misses helped improve the performance further. PSH/P suffers a 6.7% performance loss compared to P/P. PSH/PS reduces the losses to 4.2% over P/P. This is primarily due to the poor regularity, measured in terms of entropy, of the history comprising all the accesses(PSH). This is most evident in case of benchmarks *art*, *equake*, *facerec*, *mgrid* and *swim*. The significant performance losses suffered by PSH/P in these benchmarks compared to P/P indicates the importance of the design space exploration carried out by us. It shows that though extended histories can provide better performance, it need not be so for all cases. And it also shows that the best performing extended history might vary from one prefetcher to another, even among prefetchers sharing much in common like the delta correlation prefetchers. In these benchmarks, the adaptive scheme performs well in picking the right configuration and avoiding ones harmful to the performance,

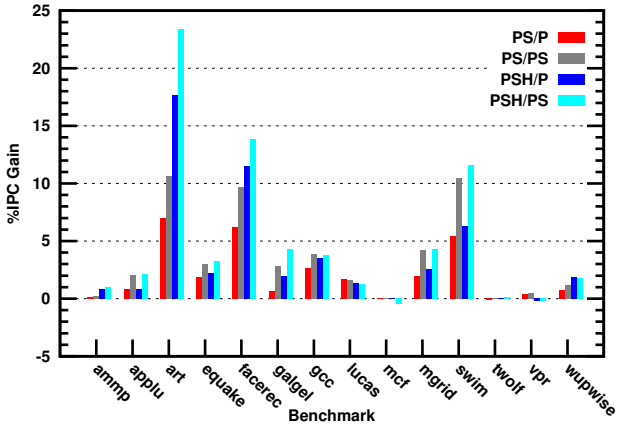

Figure 13: Access Map Prefetching – IPC gain over P/P

thereby reducing the losses seen in comparison to P/P. It is to be noted that PSH/P and PSH/PS perform badly only in comparison with P/P. Over a machine with no prefetching PSH/P and PSH/PS provide 32% and 36% gain in IPC respectively. The general trend of adaptive scheme performing as close to the best performing design holds in this case too. The adaptive scheme even manages to perform better than P/P in benchmarks like *art*, *equake* and *swim*.

Data Prefetching Championship

We study the impact of extended history and trigger points in two of the top performing prefetchers proposed in the JILP Championship Data Prefetching Contest – DPC-1.

Access Map Prefetch

Access map prefetching [7] is a prefetching technique that can overcome the negative side effects – reordering of memory accesses – of compiler and micro-architectural optimizations. Figure 13, shows the performance improvement in terms of IPC experienced by the various design points over P/P for access map prefetching[14]. P/P improves performance by 62.6% over no prefetching. The general trend is that the alternative design points provide better performance than P/P in a majority of the benchmarks. Extending the history proved to be beneficial with PS/P performing better than P/P and PSH/P providing further gains over PS/P. Secondary misses as prefetch trigger points improved the performance of the corresponding design points further. The best performing scheme PSH/PS provides a 5% gain in performance over P/P. This performance improvement is accompanied by an 1.8% reduction in memory traffic requirements.

Local Delta Buffer

The prefetching scheme proposed in [5] extends the *GHB* to include a small local delta buffer(LDB) to hold the most recent deltas of loads exhibiting predictable behaviour. The scheme also exploits simple correlation and global stride along with PerPC delta correlation. For the purpose of our study, we extended the *GHB* to include a 32 entry LDB[15]. P/P improved IPC by 85.3% compared to no prefetching. Similar to PerPC, as can be seen from Figure 14, extended history and trigger points are key to improving the perfor-

[14]A 64 entry fully-associative access map is used in this study.
[15]No prefetching was used at L1.

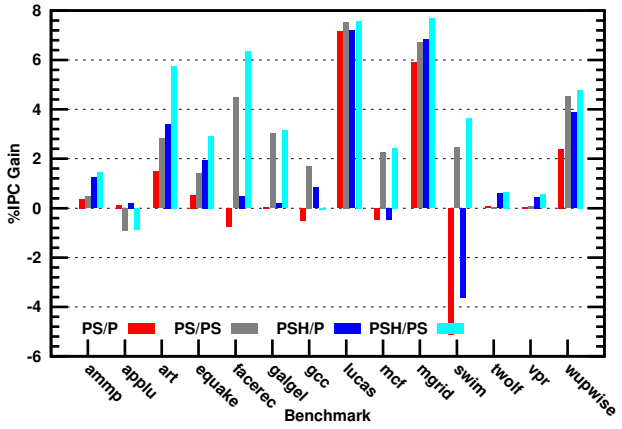

Figure 14: Local Delta Buffer – IPC gain over P/P

mance of this scheme. In fact, for most of the benchmarks, the behaviour of the various design points is similar to that observed in the case of PerPC. The best performing configuration is PSH/PS which improves the performance over P/P by 3.3%. This gain is accompanied with a traffic reduction of 2.2%.

7. CONCLUSIONS

Data prefetchers exploit regularity in their training information. Typically, the training stream is comprised of primary misses at a cache level. In this work we studied the effects of extending the training information to include secondary misses and hits. Extended histories performed better than using primary misses only and were able to eliminate more misses at the last level cache (L2 in our study). We measured the regularity present in the extended histories using the information theoretic metric entropy. Entropy measurements indicated that extended histories are more regular compared to history comprised of primary misses only and this correlated well with the reduction in misses provided by the extended histories in various prefetchers.

Extended histories also permitted triggering prefetches at points other than primary misses. In this work we identified that triggering prefetches at secondary misses along with primary misses can further improve the performance of extended histories. We studied the design space comprised of extended histories and trigger points using SPEC2000 benchmarks. The studies indicated that the best performing design point in terms of extended history and trigger point can vary from benchmark to benchmark and from prefetcher to prefetcher. The best performing design point improved the performance in terms IPC by 4.0% on an average compared to the baseline prefetch mechanism that relied only on primary misses.

We proposed and evaluated a simple adaptive scheme to identify the best performing design point for a given benchmark and adapt dynamically. The adaptive scheme provided performance improvement in terms of IPC by 4.6% on an average compared to the baseline prefetch mechanism.

8. REFERENCES

[1] A. Basu, N. Kirman, M. Kirman, M. Chaudhuri, J.F. Martinez, Scavenger: A New Last Level Cache Architecture With Global Block Priority.In *Proc. of Int. Symp. on Microarchitecture-40*, MICRO 2007.

[2] J. Baer and T. Chen, An effective on-chip preloading scheme to reduce data access penalty.*In Proc. of Supercomputing'91, 1991.*

[3] B. Bloom, Space/Time Trade-offs in Hash Coding with Allowable Errors, In Communications of the ACM, July 1970.

[4] R.Desikan, D.C. Burger, S.W. Keckler and T. Austin, Sim-alpha: a Validated, Execution-Driven Alpha 21264 Simulator. *The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-01-23,2001.*

[5] M. Dimitrov and H. Zhou, Combining Local and Global History for High Performance Data Prefetching. *In 1st JILP Data Prefetching Championship, DPC-1.*

[6] J.W.C. Fu and J.H. Patel, Stride directed prefetching in scalar processors.*In proceeding of Int. Symp. on Microarchitecture-25,1992.*

[7] Y. Ishii, M. Inaba and K. Hiraki, Access Map Pattern Matching Prefetch: Optimization Friendly Method. *In 1st JILP Data Prefetching Championship, DPC-1.*

[8] D. Joseph and D. Grunwald, Prefetching Using Markov Predictors.*In IEEE Transactions on Computer Systems, 1999.*

[9] N.P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of Intl. Symp. on Computer Architecture*, ISCA 1990.

[10] W.F. Lin, S.K. Reinhardt, D. Burger and T.R. Puzak, Filtering superfluous prefetches using density vectors.*In Proc. of ICCD,2001.*

[11] K.J. Nesbit, A.S. Dhodapkar and J.E. Smith, AC/DC: An adaptive data cache prefetcher.*In Proc. of PACT,2004.*

[12] K.J. Nesbit and J.E. Smith, Data Cache Prefetching Using a Global History Buffer.*In Proc. of Int. Symp. on High Performance Computer Architecture-10, 2004.*

[13] M.K. Qureshi, D.N. Lynch, O. Mutlu, Y.N. Patt, A Case for MLP-Aware Cache Replacement.In *Proc. of Int. Symp. Computer Architecture-33*, 2006.

[14] B.M. Rogers, A. Krisha, G.B. Bell, K. Vu, X. Jiang and Y. Solihin, Scaling the bandwidth wall: challenges in and avenues for CMP scaling. In *Proc. of Int. Symp. Computer Architecture*, ISCA 2009.

[15] C.E. Shannon, A Mathematical Theory of Communication, Bell System Technical Journal, vol. 27, pp. 379-423, 623-656, July, October, 1948

[16] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, Automatically Characterizing Large Scale Program Behaviour. In *Proc. of ASPLOS-X*, 2002.

[17] T. Sherwood, S. Sair and B. Calder, Predictor-Directed Stream Buffers.*In Proc. of Int. Symp. on Microarchitecture-33, 2000.*

[18] S. Srinath, O. Mutlu, H. Kim, Y.N. Patt, Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers.*In Proc. of Int. Symp. on High Performance Computer Architecture-13, 2007.*

[19] V. Srinivasan, G.S. Tyson and E.S. Davidson, A static filter for reducing prefetch traffic.*CSE-TR-400-99, University of Michigan Technical Report, 1999.*

[20] Z. Wang, D. Burger, K. McKinley, S. Reinhardt and C. Weems, Guided Region Prefetching: A Cooperative Hardware/Software Approach.*In Proc. of Int. Symp. Computer Architecture-30,2003.*

[21] X. Zhuang and H.H.S. Lee, A hardware based cache pollution filtering mechanism for aggressive prefetches. *In Proc. of ICCP-32,2003.*