A Hybrid Adaptive Feedback Based Prefetcher

Santhosh Verma, David M. Koppelman and Lu Peng

Department of Electrical and Computer Engineering
Louisiana State University, Baton Rouge, LA 70803
sverma3@lsu.edu, koppel@ece.lsu.edu, lpeng@.lsu.edu

Abstract

We present a hybrid adaptive prefetching scheme in this paper. The scheme consists of a hybrid stride/sequential prefetching mechanism which adapts its aggressiveness based on feedback metrics collected dynamically during program execution. Metrics such as prefetching accuracy and prefetching lateness are used to adjust aggressiveness in terms of prefetch distance (how far ahead of the current miss it fetches) and prefetch degree (the number of prefetches issued). The scheme is applied separately at both the L1 and L2 cache levels. For a set of 15 SPEC CPU 2006 benchmarks, our scheme achieves a geometric mean CPI improvement of 29% compared to no prefetching and 6% compared to a stride prefetcher. The performance improvement over a set of memory intensive benchmarks is 45% over no prefetching and 9% over stride.

1. Introduction

Memory misses are a significant limitation on performance in modern processors because of long latencies. In order to improve performance, predictable memory access patterns can be prefetched in an effort to reduce or hide misses.

Many techniques have been proposed in the literature for prefetching data. However, prefetching may not always be beneficial and can sometimes hurt performance. This will be the case if inaccurate prefetches (prefetched cache blocks that are never used) are issued, increasing both contention for memory bandwidth with demand requests and displacing useful lines that already exist in the cache (cache pollution). If the prefetching scheme is too aggressive, performance may be hurt due to contention and pollution, but being too conservative may result in lost opportunity if the prefetcher is very accurate.

We propose a dynamic adaptive hardware prefetcher which adjusts its aggressiveness based on information collected during execution. Information about prefetcher accuracy, lateness and memory bandwidth contention is collected over program segments and used to adjust the prefetcher's aggressiveness at the end of each segment. The aggressiveness of the prefetcher is defined in terms of how far ahead of the current memory access prefetching is done (prefetch distance) and the number of lines that are prefetched (prefetch degree). Our prefetching scheme shows an average (geometric mean) CPI improvement of 29% over a system with no prefetching and 6% over a stride prefetcher for the SPEC 2006 benchmarks that we simulated. We build on the work done by Srinath et. al [8] who proposed the Feedback Directed Prefetcher. Our work differs from their mechanism in several ways. We use both L1 and L2 prefetching, a hybrid prefetcher and a new feedback metric (see Section 2).

2. Related Work

Hardware prefetching schemes range from very simple schemes which prefetch the next line [7][4] to more complex schemes such as stride predictors that predict simple strides [1] and Markov predictors [3] which predict repeated and irregular address sequences. Global History Buffer [5][6] based schemes are able to effectively prefetch program regions because they do not track data on a per instruction basis.

Dahlgren et. al propose an adaptive prefetching [2] scheme which is limited to sequential access. Their scheme monitors the number of prefetches and the number of useful prefetches, and adapts aggressiveness if the number of useful prefetches is above a threshold. Nesbit et. al [5] propose an adaptive prefetcher in which the CZone (memory region) size and prefetch degree of their C/DC predictor is varied dynamically based on the detection of a change in program phase. Our work is most directly related to the adaptive Feedback Directed Prefetcher proposed by Srinath et. al [8] which we build on. Our scheme differs from their work in several ways. They limit their prefetching scheme to the L2 level while we extend the scheme so that it works for both L1 and L2. Using our scheme, we observed that in some benchmarks, many accurate L1 prefetches are issued which trigger beneficial L2 prefetches, resulting in significant performance gain. We use a combined stride-sequential scheme, while they

do separate evaluation of stride and sequential schemes. We also introduce a new metric, bandwidth contention, to evaluate prefetching effectiveness. Further, we do not use a cache pollution metric. [1]

3. Hybrid Feedback Based Prefetcher

In this section we first explain the basic Stride-sequential prefetching scheme that is used as our underlying prefetcher. This is followed by a description of the feedback based information that is used to enhance the basic prefetcher. Finally, we demonstrate how we collect and evaluate this information and use it to enhance prefetching.

3.1. Stride-Sequential Prefetching scheme

Stride prefetchers [1] detect sequences of memory address' which differ by a constant amount. For our PC-based stride prediction scheme, a Stride Prediction Table (SPT) is used to detect and monitor strides.

A single entry in the table is shown in Figure 1. The previous address field stores the lower PC bits of the previous data address that was observed for this instruction.

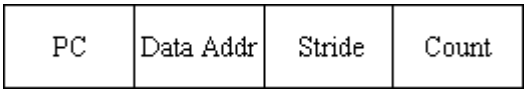

Figure 1 Stride Prediction Table Entry

When a memory instruction is first executed, an entry is allocated for this instruction in the table. On subsequent executions, a new stride is computed using the difference between the current and previous data addresses before being stored in the stride field. The count field is used to monitor the number of consecutive times that the same stride has occurred and the entry is considered trained when the value in this field is above a certain threshold. The stride table is fully associative and uses a LRU replacement policy, i.e., a new memory instruction will take the place of the least recently used table entry.

When either a cache miss occurs or there is hit to a prefetched line, the SPT is checked and prefetches may be issued if there is a valid and trained entry in the table. We combine this stride prediction scheme with a sequential prefetch policy. If a trained entry is not found in the SPT after either a demand miss or a demand hit to a prefetched line, a sequential prefetch may be issued if the previous line exists in the cache. Before a prefetch is issued in any of these cases, we check to determine if the line already exists in the cache or check the MSHR (see section 3.3) to determine if the line is either in queue or in flight. A prefetch is not issued in any of these cases.

3.2. Feedback Metrics

The aggressiveness of our prefetcher is dynamically adjusted based on feedback information collected during execution segments. The length of the segment is fixed in terms of clock cycles. For each segment we evaluate three metrics (prefetch accuracy, prefetch lateness and bandwidth contention) and use these metrics at the end of the segment to adjust prefetcher aggressiveness. These metrics are evaluated separately for the L1 and L2 caches.

Prefetch accuracy is the percentage of all issued prefetches which are eventually used by a demand request. *Prefetch lateness* is the percentage of useful prefetches which do not arrive before the demand miss occurs and are therefore late. *Bandwidth contention* is an estimate of the relative amount of outstanding memory requests during the execution segment. We include a bandwidth contention related metric because two of the three configurations that are simulated are very bandwidth limited. The formulas for each of these metrics are given below. Each metric is classified as high or low based on whether the computed value is above a certain threshold.

Prefetch accuracy = Num. of useful prefetches / Total Num. of prefetches issued
Prefetch lateness = Num. of late prefetches / Num. of useful prefetches
Bandwidth contention
= Num. of cycles during which outstanding memory requests are above a threshold / Num. of cycles

3.3. Implementation of Feedback Metrics

3.3.1 Prefetch Accuracy

For prefetch accuracy, a prefetch bit stored with every cache line is used to detect useful prefetches. When a prefetched line is brought into the cache, the prefetch bit for that line is set. If there is a demand access to a line, the prefetch bit is checked and the useful prefetch counter is incremented. Once a useful prefetch is detected, the prefetch bit is reset. The total counter is incremented on each issued prefetch.

3.3.2 Prefetch Lateness and the MSHR

For prefetch lateness, we use a fully associative set of Miss Status Hit Registers (MSHR's) to keep track of

[1] *Cache pollution is not used since the information needed is not readily available in the competition simulator.*

all in queue and in flight memory requests. A single MSHR is shown in Figure 2.

PC	Valid Bit	Cache Level	Pref. Bit 1	Pref. Bit 2

Figure 2. MSHR Entry

When a memory instruction is issued, an MSHR is allocated to it and the valid bit is set. Since we use the MSHR's to track both L1 and L2 requests, a two bit cache level field is used to indicate whether the outstanding request is either an L1 request, an L2 request or a combined L1/L2 request. Two prefetch bits (one each for L1 and L2) are also used to indicate whether an outstanding request is currently classified as a prefetch request. If there is a demand request for a prefetched line that is still in flight, the corresponding prefetch bit is checked. The late prefetches counter is incremented if the bit is set, before resetting the bit.

In addition to determining prefetch lateness, we use the MSHR to ensure that prefetch requests are not issued for an already in flight address and to keep track of the total outstanding L1 and L2 requests at any given time. The number of outstanding L1 and L2 requests is used to estimate the bandwidth contention in the system.

3.3.3 Bandwidth Contention

As mentioned in the previous subsection, we can use the MSHR to determine the total number of outstanding L1 and L2 requests. In each cycle, we check to see if this number is above a threshold and increment a counter if it is. In addition, we use this information to throttle new prefetch requests if the number of outstanding requests is too high.

3.4. Adjusting Prefetcher Aggressiveness

The metrics described in the previous section are evaluated at the end of each segment and used to determine if prefetcher aggressiveness should be increased or decreased. The table below shows the specific cases under which we adjust aggressiveness. The thresholds used for Accuracy, Lateness and Contention are 0.45, 0.07 and 0.07 respectively.

Accuracy	Lateness	Contention	Policy
High	Late	Low	Increase
Low	Late	Either	Decrease

Table 1 Aggressiveness Policy

We adjust aggressiveness of the prefetcher by varying two prefetch parameters, prefetch distance and prefetch degree. The prefetch distance indicates how many blocks ahead of the current block that the prefetcher should be prefetching. The higher the distance, the further down the address stream that prefetches are issued and the more aggressive the prefetcher. The prefetch degree indicates the number of blocks that should be prefetched when prefetching is triggered (either on a miss or access to a prefetched block). The higher the degree, the more blocks are prefetched and the more aggressive the prefetcher. The various aggressiveness levels are shown in Table 2 in increasing order of aggressiveness.

Prefetch Distance	Prefetch Degree
4	1
8	1
16	2
32	4
64	4

Table 2 Prefetcher Aggressiveness levels

4. Experimental Evaluation

4.1. Experimental Framework

Config	Prefetching	Bandwidth Limited	L2 Size(KB)
1	No	No	2048
1	Yes	No	2048
2	No	Yes	2048
2	Yes	Yes	2048
3	No	Yes	512
3	Yes	Yes	512

Table 3 Configurations for Simulations

The CMPSim simulator for the JILP-DPC Prefetching competition is used to do our simulations. The simulator models an out of order processor with a 15 stage, 4 wide pipeline and perfect branch prediction. The L1-cache is 32 KB, 8-way set associative with LRU replacement and a miss latency of 20 cycles for an L1 hit/L2 miss. The L2-cache is 16-way set associative with LRU replacement and a miss latency of 200 cycles (size of cache specified below). The trace generator provided with this simulator is used to create traces spanning 100 million instructions after skipping the first 40 billion instructions. We generate traces for a total of 15 benchmarks selected from the SPEC CPU 2006 suite. These include memory and CPU intensive benchmarks. For each prefetcher that we evaluate,

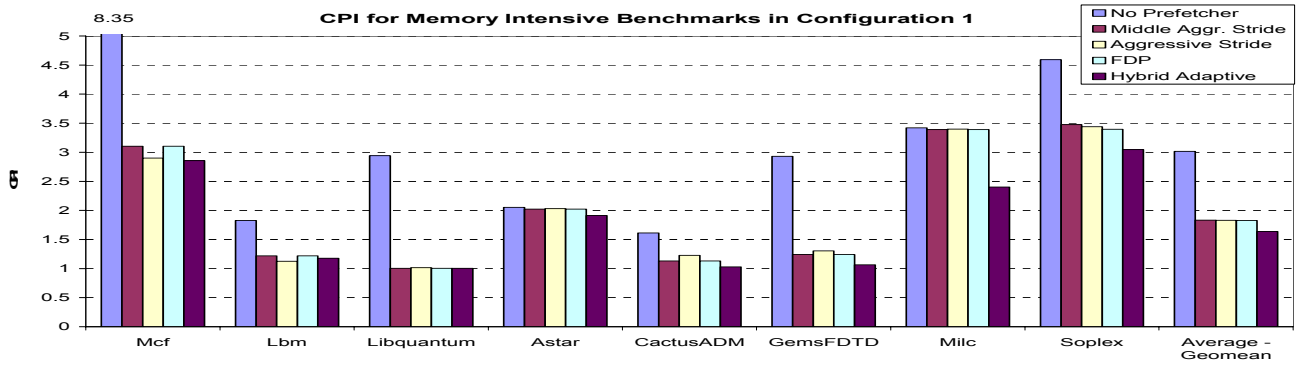

Figure 3. CPI Comparisons for Memory Intensive Benchmarks in Configuration 1

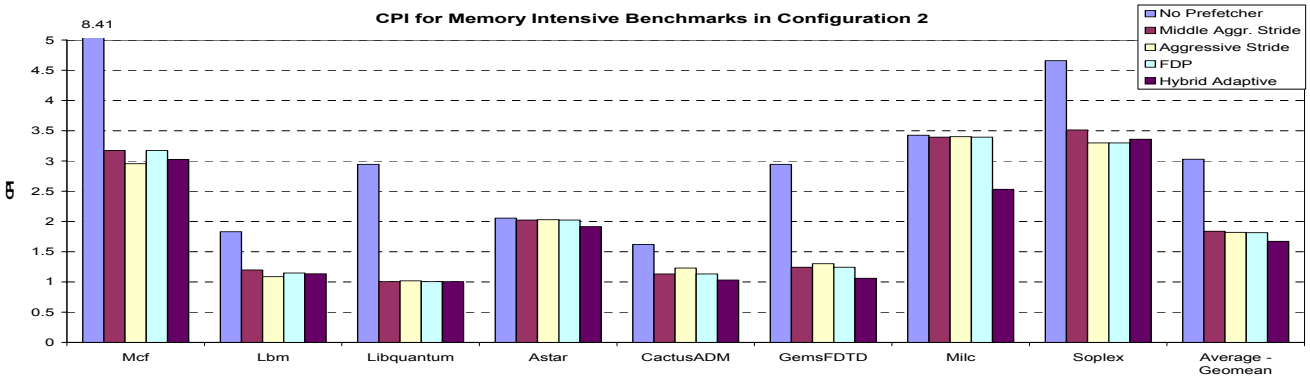

Figure 4. CPI Comparisons for Memory Intensive Benchmarks in Configuration 2

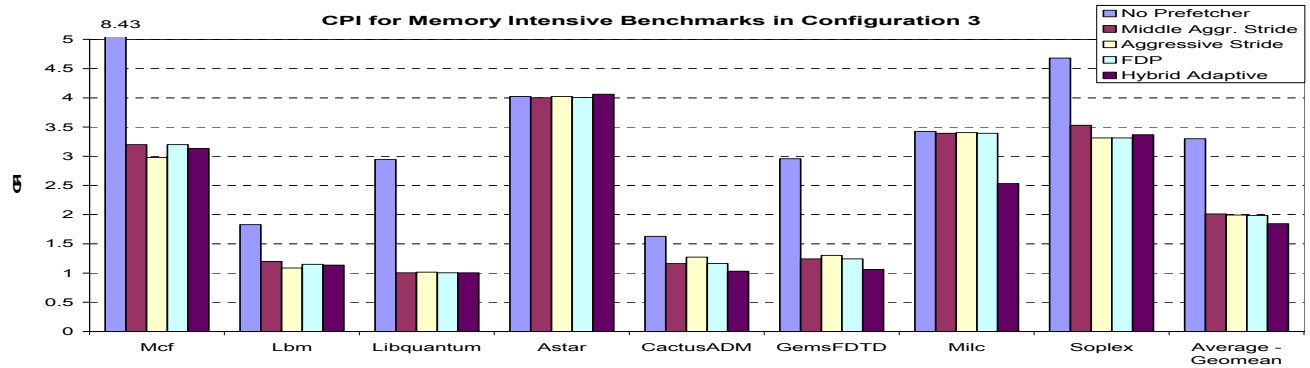

Figure 5. CPI Comparisons for Memory Intensive Benchmarks in Configuration 3

simulations are done using these benchmarks for the six competition configurations listed in Table 3.

4.2. Experimental Evaluation & Results

We evaluate our Hybrid Adaptive Prefetcher (HAP) over the three configurations listed above and compare its performance with two stride prefetching schemes, a version of the Feedback Directed Prefetcher and a system that does not use prefetching.

In figures 3-5, we show the CPI results over configurations 1 through 3 respectively for a subset of SPEC 2006 benchmarks which benefit from prefetching. The geometric mean CPI for HAP improves by 45% compared to no prefetching and 9% compared to the Middle-Aggressive stride prefetcher. This prefetcher has a fixed aggressiveness level of prefetcher distance 16 and prefetch degree 2. The Aggressive stride prefetcher (prefetch distance 64, prefetch degree 4) performs slightly better on average compared to the Middle-Aggressive prefetcher. The version of the Feedback Directed Prefetcher (FDP) that we implemented is on average equivalent to the Aggressive stride prefetcher. This version varies from FDP proposed in [8]

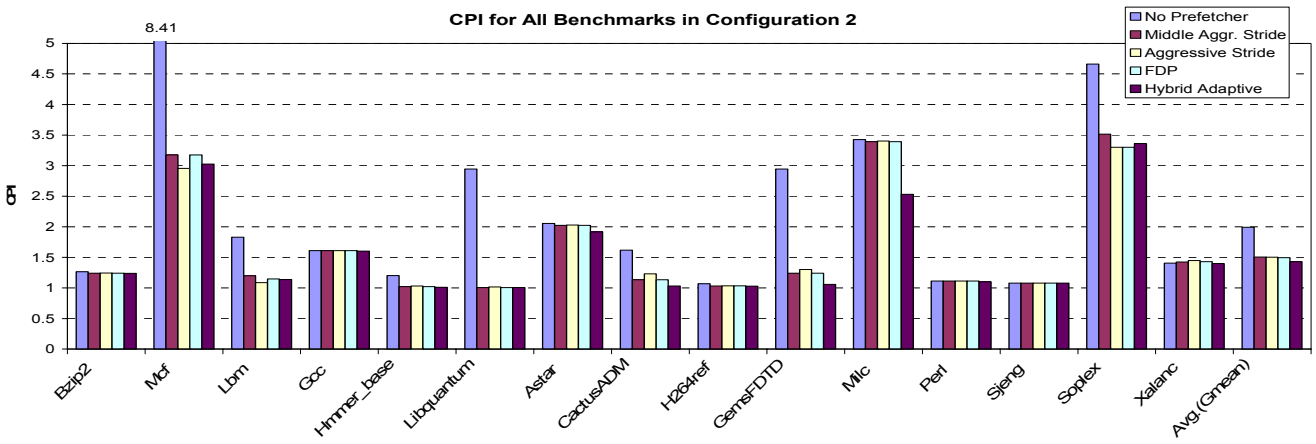

Figure 6. CPI Comparisons for All Benchmarks in Configuration 2

in both major and minor ways. Most importantly, it uses the combined L1 and L2 prefetching described here but does not use a pollution metric.

It can be seen that HAP outperforms the other schemes for most benchmarks. The exceptions are mcf and lbm which achieve higher performance with the Aggressive stride prefetcher. The mcf benchmark is very memory intensive and has a load miss rate greater than 90%. As such, it is not surprising that it does well with the most aggressive configuration. Figure 6 shows the CPI data over all 15 benchmarks that we simulated for configuration 3. While there are many benchmarks which are unaffected by prefetching, the HAP scheme still achieves an average improvement of 29% over no prefetching and 6% over Middle-Aggressive stride.

4.3. Prefetcher Size

We show in this section that our scheme fits into the competition committee's budget requirements.

*Stride Prefetcher L1 => 256 entries *(32 PC bits + 11 bit data address field + 10 bit stride + 2 bit state) = 14080 bits*
Stride Prefetcher L2 => 256 entries =>14080 bits
*MSHR => 16 * (2 cache level bits + 2 pref. bits) + 112 * (32 PC bits + 2 cache bits + 2 pref. bits) = 4096 bits*
For tracking counters useful_pref, late_pref, total_pref, cycles bandwidth above threshold
= > 4 counters each for L1 and L2 and each counter counts up to a million (20 bit counters)
=> 8 20 = 160 bits (trivial)*

Total: 32416 bits.

5. Conclusions

In this paper, we propose a Hybrid Adaptive prefetcher which dynamically adjusts the aggressiveness level of a hybrid prefetching scheme based on a set of feedback metrics. The prefetching scheme achieves significant performance gains compared to two stride prefetching schemes, a basic Feedback Directed Prefetcher and a scheme that does not use prefetching. Future work can try to incorporate the pollution metric that was proposed in [8] into the HAP scheme.

References

[1] J-L. Baer and T.-F Chen. Effective hardware based data prefetching for high performance processors. *IEEE Transactions on Computers*, 1995.

[2] F. Dahlgreen, M. Dubois and P Stenstrom. Sequential Hardware prefetching in shared–memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems,* 1995

[3] D. Joseph and D. Grunwald. Prefetching using markov predictors. *Proceedings of the24th Annual International Symposium on Computer Architecture*, 1997.

[4] N.P Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. *Proceedings of the 17th Annual International Symposium on Computer Architecture*, 1990.

[5] K. Nesbit, A. Dhodapkar, and J. Smith. AC/DC: An adative data cache prefetcher. In *PACT*, 2004.

[6] K. Nesbit and J. Smith. Prefetching with a global history buffer. *HPCA, 2004.*

[7] A. Smith. Sequential program prefetching in memory hierarchies. *IEEE Transactions on Computers*, December 1978.

[8] S. Srinath, Onur Multu, Hyesoon Kim, Yale N. Patt. Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers. *HPCA,* 2007.