Journal of Instruction-Level Parallelism 13 (2011) 1-14 Submitted 3/10; published 1/11

Efficient Prefetching with Hybrid Schemes and Use of Program Feedback to Adjust Prefetcher Aggressiveness

Santhosh Verma SVERMA3@LSU.EDU
David M. Koppelman KOPPEL@ECE.LSU.EDU
Lu Peng LPENG@LSU.EDU
Department of Electrical and Computer Engineering
Louisiana State University
Baton Rouge, LA 70803 USA

Abstract

A set of hybrid and adaptive prefetching schemes are considered in this paper. The prefetchers are hybrid in that they use combinations of Stride, Sequential and C/DC prefetching mechanisms. The schemes are adaptive in that their aggressiveness is adjusted based on feedback metrics collected dynamically during program execution. Metrics such as prefetching accuracy and prefetching timeliness are used to vary 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. We previously proposed a Hybrid Adaptive prefetcher for the Data Prefetching Competition (DPC-1) which uses a hybrid PC-based Stride/Sequential prefetcher.

In this work, we also consider hybrid schemes that use the CZone with Delta Correlations (C/DC) prefetcher. Further, we breakdown the components of hybrid adaptive prefetching by evaluating the individual benefits of using hybrid schemes and using adaptivity. While hybrid prefetching improves performance over a single prefetcher, we demonstrate that the main benefit of adaptivity is the reduction in the number of prefetches without hurting performance. For a set of 18 SPEC CPU 2006 benchmarks, the best hybrid adaptive scheme achieves an average (geometric mean) CPI improvement of 35% compared to no prefetching and 19% compared to a basic Stride prefetcher. Adaptivity makes it possible to reduce the number of prefetches issued by as much as 50% in some benchmarks without sacrificing performance.

1. Introduction

Cache misses are a significant limitation on performance in modern processors because of long memory latencies. Performance can be improved by prefetching predictable memory access patterns in an effort to reduce or hide misses [7]. The performance of some programs is dominated by cache misses and many of them can benefit from extremely aggressive prefetching. The aggressiveness of the prefetcher can be increased either by increasing the number of prefetches issued on a miss (prefetch degree) and/or increasing the distance from the current memory access where we start prefetching (prefetch distance). In benchmarks like mcf, the more aggressive we make the underlying prefetcher, the greater the performance improvement we can achieve. This motivated us to look at hybrid prefetching in order to exploit the maximum possible prefetching opportunities. On the other hand, prefetching may not always be beneficial and may even hurt performance. This will be the case if inaccurate prefetches of cache blocks that are never used are issued, as contention for memory bandwidth with demand requests increases and useful cache lines that already exist in the cache are displaced (cache pollution). As such, being

too aggressive may hurt performance and waste resources but being too conservative may result in lost opportunity for benchmarks dominated by cache misses. Therefore we also look at adjusting the aggressiveness of the prefetcher dynamically in order to achieve the maximum possible performance in the most efficient way possible.

A hybrid adaptive prefetching scheme [14] was selected as a finalist at the Data Prefetching Competition (DPC-1). This scheme uses a PC-based Stride prefetcher and a Hybrid Stride/Sequential prefetcher, the aggressiveness of which is adjusted dynamically based on information collected during program execution. Information about prefetcher accuracy, timeliness and memory bandwidth contention is collected over program segments and used to adjust the prefetcher's aggressiveness (in terms of prefetch distance and prefetch degree). The adaptive component of the prefetcher is built on the work done by Srinath, et al. [13] who proposed the Feedback Directed Prefetcher (FDP). FDP adjusts the aggressiveness of the prefetcher based on three metrics, accuracy, lateness (the inverse of timeliness) and pollution. However, the schemes discussed in this paper differ from FDP in several ways. Their prefetching was limited to the L2 level, while we do both L1 and L2 prefetching. We observed that L1 prefetching can significantly improve performance by prefetching useful data into both the L1 and L2 caches. We also consider hybrid prefetching. In addition to their accuracy and lateness metric, we use bandwidth based metrics to adjust prefetcher aggressiveness. In this paper, we evaluate CZone with Delta Correlations (C/DC) based prefetching schemes and a broader range of hybrid prefetching schemes in addition to the Stride and Stride/Sequential prefetching mechanisms originally considered in [14]. The best hybrid prefetching scheme shows an average CPI improvement of 35% over a system with no prefetching and 19% over a basic Stride prefetcher for the SPEC 2006 benchmarks that we simulated.

The rest of the paper is organized as follows. In section 2 we look at prior work. In section 3, we explain the Stride and C/DC prefetching schemes as well as the hybrid prefetching schemes we consider. In section 4, we look at how we adjust the aggressiveness of prefetching. In section 5 we look at Experimental results and we conclude in section 6.

2. Prior Work

The simplest hardware prefetching schemes are sequential schemes which prefetch the next cache lines [7, 12] if a sequential access pattern is detected. Stride prefetching schemes [1, 3] are more complex as they issue prefetches if a constant stride in the memory access is detected. A table stores the most recent stride as well as the most recent memory address that was issued for a load instruction. A stride is computed by calculating the difference between the memory address value of the current access and the previous address value that is stored in the table entry. If the stride is the same between consecutive accesses, a two-bit status field is incremented, indicating a repeating pattern of the same stride. The limitations of basic Stride prefetchers are their use of program counters which may not be easily accessible at the L2 cache level and their reliance on detecting per instruction patterns instead of global memory patterns. Stride Stream Buffer CZone prefetching [10] overcomes these issues by dividing the memory space into fixed size partitions (CZones). Instead of tracking per instruction access patterns by indexing a table with the program counter, the accesses in each CZone are monitored to determine stride patterns.

More complex address patterns are considered in correlation prefetching methods such as Markov predictors [6] which predict repeated and irregular address sequences. Instead of directly accessing a table, Global History Buffer based schemes [8, 9] utilize entries which store an address and a link to a previous entry. The GHB is a FIFO table of the most recent addresses and

can be used to create a time ordered linked list of addresses. In C/DC prefetching [8], an index table holds an initial pointer to an entry in the GHB. The index table is accessed using the tag of the CZone memory region. The address patterns are monitored to detect delta correlation patterns.

Nesbit, et al. [8] also propose an adaptive component in which the CZone size and prefetch degree of the C/DC predictor is varied dynamically based on the detection of a change in program phase. Dahlgreen, et al. [2] propose an adaptive prefetching 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.

Hur and Lin [4] propose Adaptive Stream Detection which adjusts the aggressiveness of a Stream prefetcher based on stream length histograms. They improve on this work in [5] by improving their stream detection technique, dynamically adjusting the evaluation period for collecting data and supporting variable length prefetching of multiple cache lines. Ramos, et al. [11] propose low cost solutions for adjusting the aggressiveness of sequential prefetching. The adaptive component of the prefetching schemes considered in this paper is most directly related to the adaptive Feedback Directed Prefetcher proposed by Srinath, et al. [13]. 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 also consider hybrid Stride and C/DC prefetching schemes, while they do separate evaluation of Stride, Sequential and C/DC schemes. We also use bandwidth usage based metrics to adjust prefetcher aggressiveness. Further, we do not use a cache pollution metric in this paper (this is due to limitations of the competition simulator).

3. Prefetching Schemes

We explain basic Stride prefetching and our Stride/Sequential prefetching mechanism followed by an explanation of the C/DC predictor and C/DC based hybrid prefetching schemes.

3.1. Stride and Stride/Sequential Prefetching

Stride prefetchers [1] detect sequences of memory address' which differ by a constant amount. 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. 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 a saturating counter 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 the 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. The aggressiveness of stride prefetching can be increased by increasing both prefetch distance (how far from the current miss that we prefetch) and prefetch degree (how many prefetches are issued per miss).

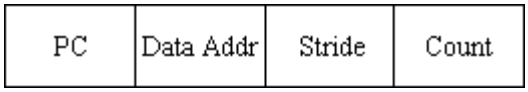

Figure 1: Stride Prediction Table (SPT) entry.

In general, the hybrid schemes in this paper consider secondary prefetching schemes in case the primary prefetcher does not find a prefetch pattern. The Stride/Sequential prediction scheme considers sequential options if there is no stride option. 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. The sequential prefetcher will issue prefetches for the next few cache lines (up to a maximum prefetch degree) if the previous line exists in the cache and has been accessed by a demand request. Before a prefetch is issued at any point (for any of the prefetching schemes), 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 these cases.

3.2. C/DC and C/DC Based Hybrid Prefetching Schemes

The CZone with Delta Correlations (C/DC) prefetching mechanism was proposed by Nesbit, et al. [8]. The memory is divided into fixed size regions called CZone and memory access patterns are determined for each CZone. A CZone index table stores an entry for each CZone of the memory and each entry has a pointer to an element in a global history buffer (GHB). The entries in the GHB store a delta value and a pointer to a previous entry in the GHB. As such, for each CZone, it is possible to create a linked list of its most recent delta values. The delta values can be analyzed to determine delta correlation access patterns (such as 1, 2, 5, 1, 2, 5 ...) in the address stream. The prefetcher accesses the index table (using higher order bits of the miss address) to find its CZone table entry, and follows the pointers to obtain the most recent delta values. It uses two pairs of registers to compare pairs of deltas in the address stream and determine a delta correlation. Deltas are shifted into these registers and when they match, a correlation has been detected and prefetches may be issued. Computed deltas are also stored in a delta buffer which is used to compute prefetch addresses. In addition to finding delta patterns, this scheme can also predict constant stride patterns.

The C/DC predictor is more powerful than the Stride predictor for several reasons. Firstly, it can predict more complex patterns than simple stride accesses. Secondly, C/DC looks at global access patterns in a region of memory and is not limited to the local per instruction access patterns that PC-based stride prediction schemes are limited to. In addition, C/DC does not need access to the program counters of instructions which is beneficial because PC's may not be readily available especially at the L2 cache level.

We extend the C/DC mechanism in this paper by exploring hybrid C/DC-Sequential and C/DC-Sequential-Stride mechanisms. The C/DC-sequential predictor explores deltas to find sequential or almost sequential patterns. Deltas are compared, and if they are below a threshold value, a sequential pattern is detected. Sequential prefetching is considered if the main predictor does not detect a pattern. Similarly, the C/DC-Sequential-Stride predictor will consider PC-based stride prefetching if neither the main predictor nor the sequential component detects a pattern.

4. Feedback Metrics

The aggressiveness of prefetching 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. The timeliness of prefetching is measured by *Prefetch lateness*, 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 bandwidth based metrics because the simulator configuration we use limits the number of L1 and L2 requests that can be issued in each cycle. 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 usage = Num. of cycles with outstanding memory requests above threshold / Num. of cycles

4.1. Implementation of Feedback Metrics

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.

Prefetch Lateness and the MSHR - For prefetch lateness, we use a fully associative set of Miss Status Handling Registers (MSHR's) to keep track of all in queue and in flight memory requests. A single MSHR is shown in Figure 2.

When a memory instruction is issued, an MSHR is allocated to it and the valid bit is set. Prefetch lateness is also measured in [13] but only L2 prefetching is considered. 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 both the 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 following which the bit is reset. The lateness is measure as a percentage of the number of useful prefetches. In addition to determining prefetch lateness, the MSHR is used to ensure that prefetch requests are not issued for already in flight addresses 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.

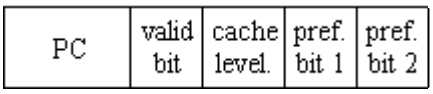

Figure 2: A single MSHR entry.

Bandwidth Usage - As mentioned above, 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. Bandwidth is also used to control the aggressiveness of prefetching by limiting the number of outstanding L1 and L2 requests at any given time. If there are too many outstanding requests, prefetches are not issued. Since the number of requests that can be issued per cycle is limited, the goal of this mechanism is to prevent queuing prefetches which may be too late by the time they reach the head of the queue and are issued.

4.2. 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 tables below show the specific cases under which we adjust aggressiveness. Since the level of prefetching varies significantly between Stride and C/DC prefetching, we use a slightly different set of rules and thresholds for each group of prefetchers. These rules are shown in Table 1 and 2. For each metric, its value is compared to a threshold and is determined to be either high or low. These threshold values are shown in brackets. For example, if the accuracy of prefetching is greater than 0.45 for Stride (Table 1), then the accuracy is considered high. If a combination is not listed in the table then there is no change in aggressiveness for that combination. In general, we increase aggressiveness if prefetching is accurate and either late in bringing in data or if the overall bandwidth usage is low. Aggressiveness is decreased for inaccurate prefetches which are also late, or for inaccurate prefetches which consume too much bandwidth. Note that there is one fewer decrease aggressiveness configuration for Stride. We will see in the next section that PC-based Stride prefetching schemes are less robust and issue far fewer prefetches than C/DC. As such, there is less reason to decrease their aggressiveness in general.

For stride based prefetching schemes, we adjust aggressiveness of the prefetcher by varying the prefetch distance and prefetch degree. The higher the distance, the further down the address stream that prefetches are issued and the more aggressive the prefetcher. The higher the degree, the more blocks are prefetched and the more aggressive the prefetcher. For C/DC based schemes, we adjust the prefetcher degree.

Accuracy (0.45)	Lateness (0.07)	Bandwidth Usage (0.10)	Action
High	Late	Low	Increase aggressiveness
High	Not Late	Low	Increase aggressiveness
Low	Late	Any	Decrease aggressiveness

Table 1: Rules for adjusting stride prefetcher aggressiveness.

Accuracy (0.45)	Lateness (0.07)	Bandwidth Usage (0.10)	Action
High	Late	Low	Increase aggressiveness
High	Not Late	Low	Increase aggressiveness
Low	Late	Any	Decrease aggressiveness
Low	Not Late	High	Decrease aggressiveness

Table 2: Rules for adjusting C/DC prefetcher aggressiveness.

5. Experimental Evaluation

5.1. Experimental Framework

The CMPSim simulator provided for the JILP Data Prefetching competition (DPC-1) 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. For the configuration used in this paper, the L2-cache is 16-way set associative with LRU replacement and a miss latency of 200 cycles. 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 18 benchmarks selected from the SPEC CPU 2006 suite. These include memory and CPU intensive benchmarks. We measure the performance in terms of CPI for a set of Stride, C/DC, Hybrid and Adaptive schemes. We also look at the number of prefetches that prefetching schemes need to issue in order to achieve their performance gains.

Mechanism	Number of entries	Approximate cost
Stride Table	256	2 KB
C/DC Index Table and GHB	256 + 256	2 KB
MSHR's	128	0.5 KB
Prefetch bits L1	512	0.1 KB
Prefetch bits L2	32768	4 KB

Table 3: Prefetching table sizes and other costs.

Table 3 shows the table sizes and other costs for our prefetching system. We also use a 128 entry MSHR to keep track of in queue and inflight requests at both the L1 and L2 cache levels. The basic prediction tables (such as the Stride Prediction Table and the C/DC Index table) are duplicated at both the L1 and L2 levels. In addition to the basic prediction tables described in section 3, the scheme also uses the one additional prefetch bit per line provided by the simulator to keep track of useful prefetches. While the amount of storage required for the prefetch bit at the L2 level is high, it is not as high as a percentage of the total cache size (2 MB). We use this bit since it was provided "free of cost" in the original competition framework.

5.2. Performance of Hybrid Prefetching Mechanisms

Figures 3 and 4 show the CPI for Stride and C/DC based hybrid prefetching schemes respectively for a subset of benchmarks which are sensitive to prefetching. The benefit of Hybrid prefetching

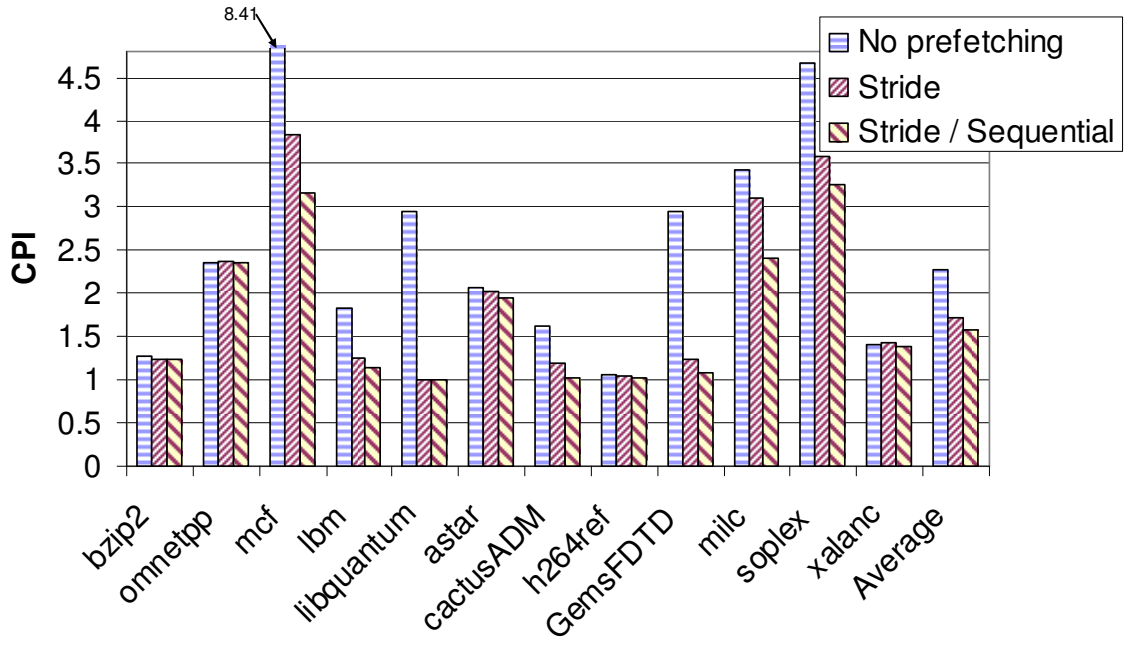

Figure 3: CPI for Stride and Stride Hybrid prefetching schemes.

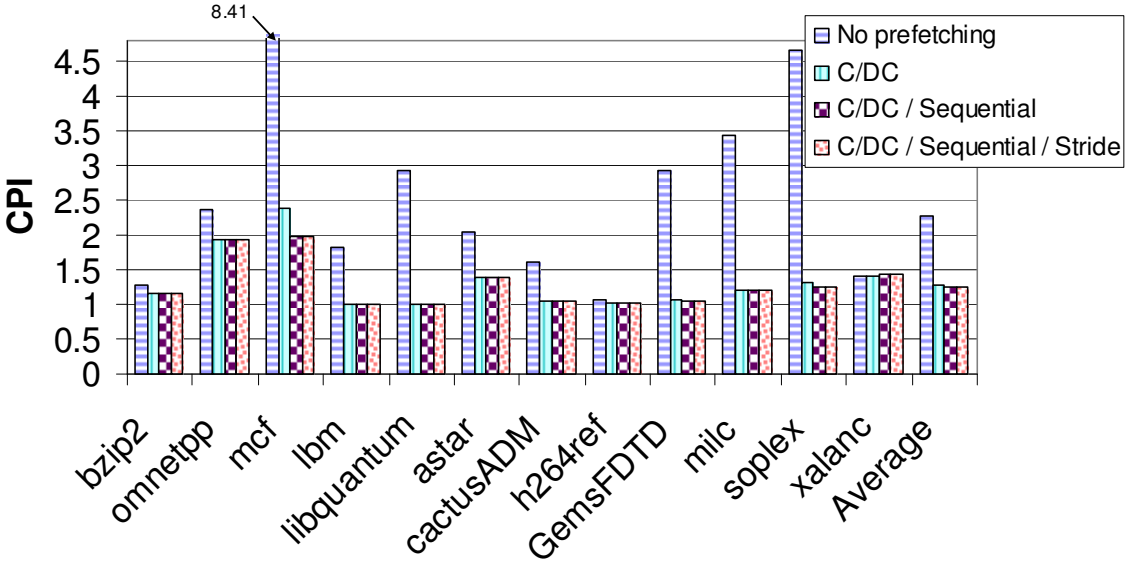

Figure 4: CPI for C/DC and C/DC Hybrid prefetching schemes.

is clearly very significant for the Stride prefetcher. The Stride/Sequential prefetcher shows significant performance gains compared to Stride prefetching for memory intensive benchmarks such as mcf, milc and soplex. The average (geometric mean) improvement in CPI for Stride/Sequential prefetching for these benchmarks compared to Stride only prefetching is 9% and 33% compared to no prefetching. The performance benefit of hybrid prefetching is less significant for the C/DC / Sequential prefetcher compared to C/DC prefetching. While benchmarks like mcf and soplex benefit significantly from the sequential prefetching component,

the average CPI improvement for these benchmarks is 2%, significantly lower than the improvement in Stride prefetching. We also implemented a C/DC / Sequential / Stride prefetcher that considers a PC-based stride prediction option if neither a delta correlation nor a sequential pattern is found. Since the basic C/DC predictor can also detect stride patterns (the difference is that C/DC looks at the global address stream), it is not surprising that this configuration did not yield any performance improvement.

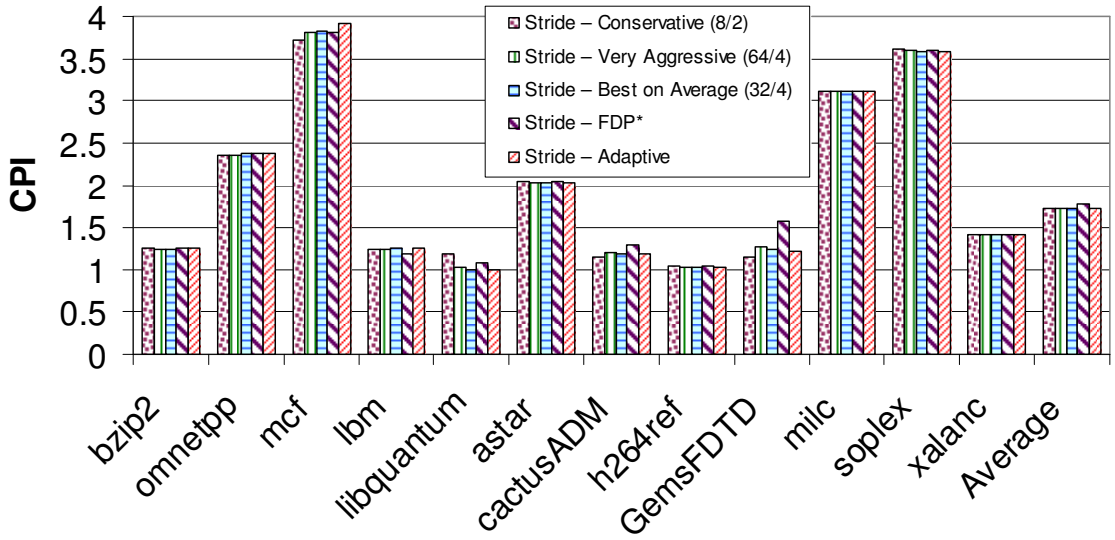

Figure 5: CPI for Stride prefetcher configurations with varying aggressiveness.

5.3. Impact of Using Adaptive Mechanisms on CPI

Figure 5 and Figure 6 show the changes in CPI for varying levels of prefetcher aggressiveness for Stride and C/DC based prefetchers respectively. For each configuration, we show the prefetch distance and degree in brackets. For example, the conservative stride prefetcher in Figure 5 uses distance of 8 and a degree of 2 while the very aggressive configuration uses a distance of 64 and a degree of 4. The best on average configuration is the configuration which provides the best results on average over all benchmarks. We can observe in both Figures 5 and 6 that the best configurations for individual benchmarks can vary. For example with stride based mechanisms, benchmarks like mcf and cactusADM do best with conservative prefetching schemes while libquantum and soplex do best with more aggressive schemes. In the C/DC based configurations in Figure 6, conservative prefetching is adequate for benchmarks like lbm and libquantum, while benchmarks like mcf, milc and soplex do significantly better with aggressive prefetching schemes. Since the best aggressiveness configuration can vary significantly across benchmarks, these results demonstrate the case for adaptive prefetching. The results for both Figure 5 and Figure 6 show that our adaptive mechanism does not fluctuate much from the best individual configurations for any of the benchmarks. In fact, our adaptive mechanism gets better performance than the best average configuration for mcf, astar and xalanc in Figure 6.

We also included an FDP configuration in these two figures. This configuration is our approximation of the Feedback Directed Prefetching [13] although there are some major and minor differences. While they only did L2 prefetching in their work, we extend FDP to cover both L1 and L2 prefetching. Our mechanism also differs from FDP in the use of bandwidth based metrics to adjust aggressiveness. On average, our modifications provide a performance benefit of

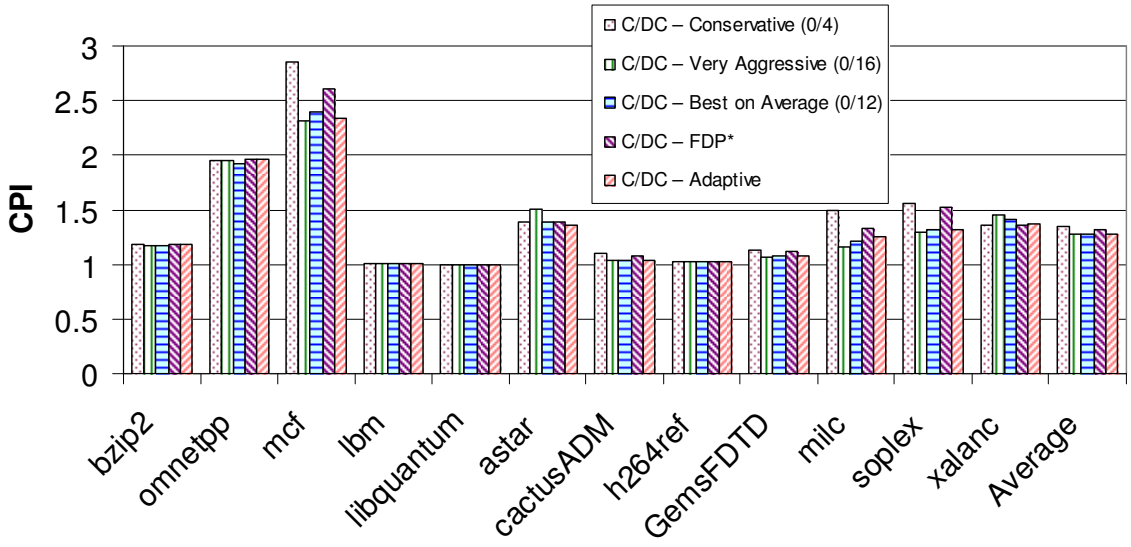

Figure 6: CPI for C/DC prefetcher configurations with varying aggressiveness.

metrics to adjust aggressiveness. On average, our modifications provide a performance benefit of about 3% compared to this version of FDP.

Figure 7 shows the impact of adaptivity on hybrid C/DC based prefetching mechanisms. These results are not significantly different from the results for the non-hybrid schemes. The adaptive scheme here seems to do slightly worse on average, but this result may be because we use the same rules and thresholds that were used for the non-hybrid mechanisms. Since the behavior of the hybrid prefetcher will differ, it would make sense to find a suitable set of parameters and rules specifically for these configurations.

5.4. Impact of Using Adaptive Mechanisms on Prefetching Bandwidth

While we saw in the previous section that adaptive mechanisms can provide performance benefits equivalent to or better than the best individual configurations for benchmarks, we will see in this section that the main benefit of adaptivity is to achieve the maximum performance by issuing as few prefetches as possible. Issuing too many prefetches may lead to a waste of bandwidth and increased contention in more complex systems and also wastes energy. Figure 8 and Figure 9 show the number of prefetches issued (in millions) at both the L1 and L2 cache levels for C/DC and Stride based prefetching schemes respectively. The number of prefetches issued for adaptive schemes is significantly lower for many benchmarks, particularly in the C/DC based schemes. Adaptivity can reduce the number of prefetches issued by as much as 50% for some benchmarks compared to the best average configuration without sacrificing accuracy. On average, the total number of prefetches is reduced from 5 million to 4 million for the hybrid C/DC based mechanisms. The decrease in prefetches is less significant for Stride prefetchers, but this is largely because Stride prefetching does not find as many prefetching opportunities in the first place. The average number of prefetches issued for the Adaptive Stride scheme is 0.51 million compared to 0.59 million for the non-adaptive Stride.

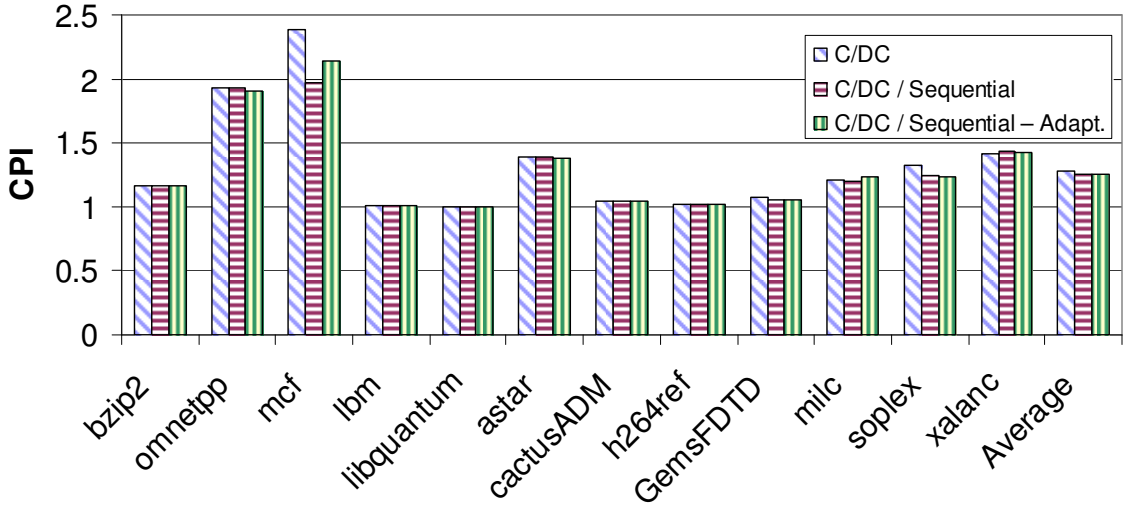

Figure 7: CPI for C/DC based Hybrid Adaptive Schemes.

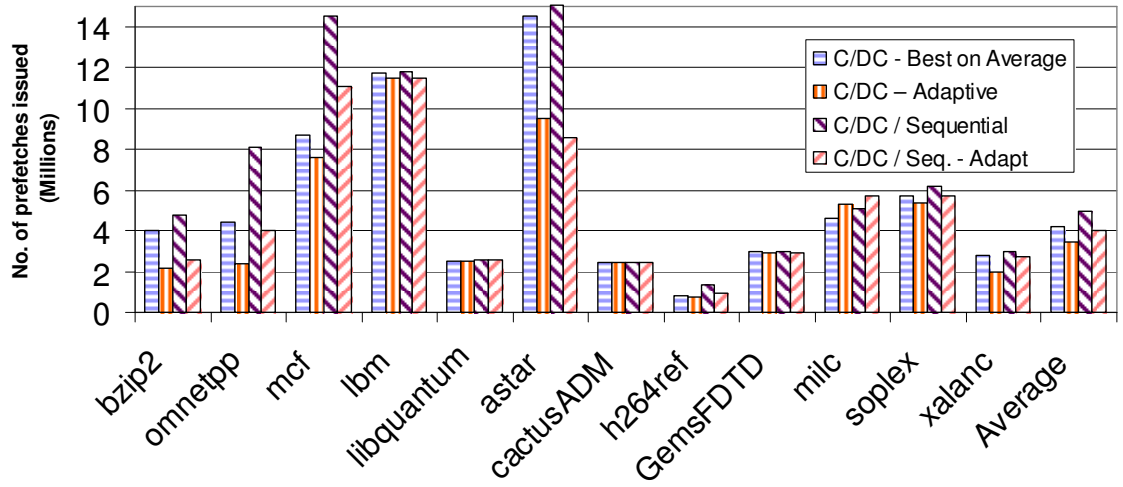

Figure 8: Number of prefetches issued for C/DC based prefetchers.

5.5. Results Over All Benchmarks

For completeness, Figure 10 and Figure 11 show the CPI for some of the schemes discussed in this paper for all 18 benchmarks that we simulated. The average CPI improvement over a system with no prefetching is 20% for Stride, 24% for Stride/Sequential, 34% for C/DC and 35% for C/DC / Sequential.

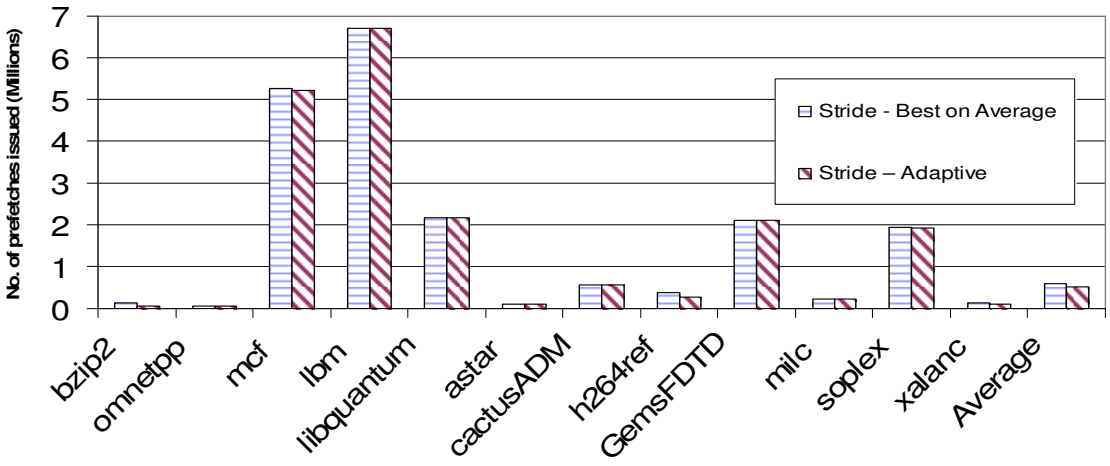

Figure 9: Number of prefetches issued for Stride based prefetchers.

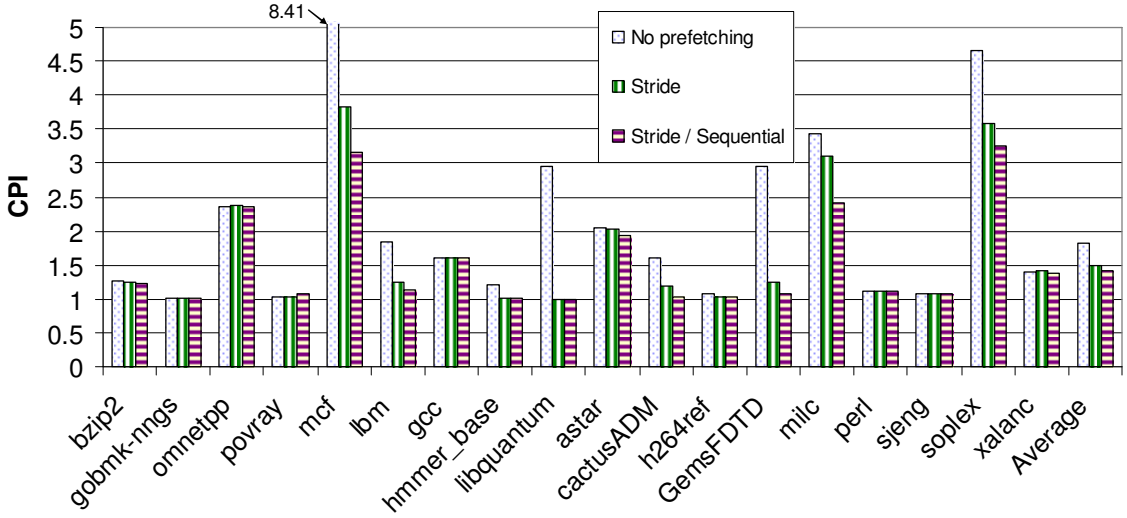

Figure 10: CPI for Stride based prefetching schemes over all simulated benchmarks.

6. Conclusions

In this paper, we consider a broad set of hybrid and adaptive prefetching schemes. Since some benchmarks can benefit significantly from more aggressive prefetching, hybrid prefetching mechanisms have the ability to improve performance by exploring a greater set of prefetch possibilities. On the other hand, inaccurate and late prefetches can waste bandwidth and energy, cause cache pollution, and increase memory contention in more complex systems. This motivates the case for adaptive prefetching mechanisms which try to achieve the maximum possible performance in the most efficient way possible.

Therefore, we take a more detailed look at the benefits of both Hybrid and Adaptive prefetching schemes. We explore a set of Stride and C/DC based hybrid prefetching schemes and

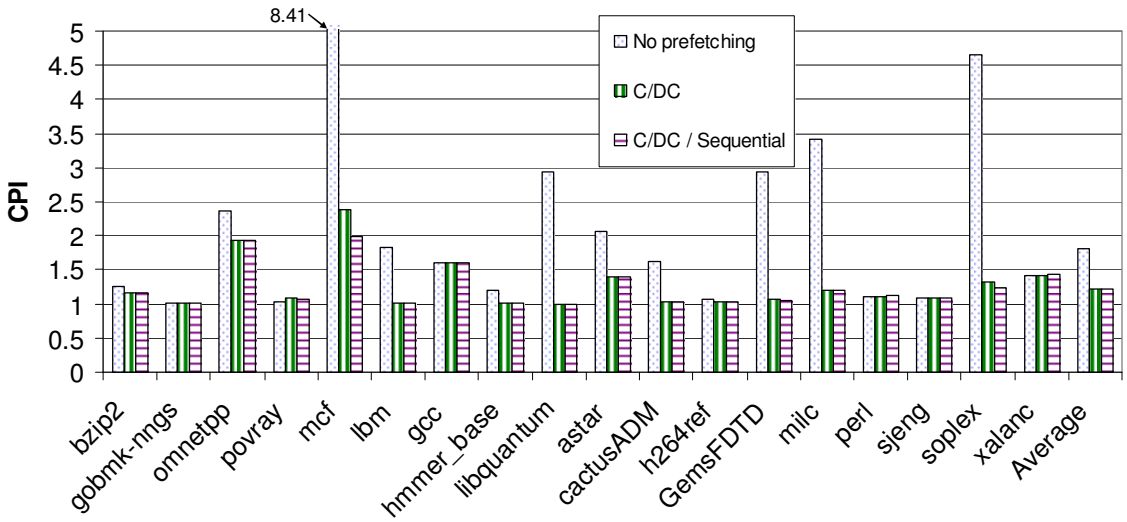

Figure 11: CPI for C/DC based prefetching schemes over all simulated benchmarks.

evaluate the impact of adaptive prefetching. We show that Hybrid prefetching schemes can improve performance of the underlying prefetcher by exploring more prefetch opportunities. The benefit of hybrid prefetching is more significant for Stride based mechanisms than C/DC based mechanisms. We also show that adaptive mechanisms can match and even outperform the best single configuration mechanisms. However, the more important benefit of adaptivity is achieving the maximum possible performance by issuing as few prefetches as possible. In some benchmarks, performance is not lost even though the number of prefetches is reduced by half.

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] J.W.C. Fu and J.H. Patel, "Stride directed prefetching in scalar processors," In *Proceedings of the 25th International Symposium on Microarchitecture*, 1992.

[4] Ibrahim Hur and Calvin Lin, "Memory Prefetching Using Adaptive Stream Detection," In *Proceedings of the 39th International Symposium on Microarchitecture*, 2006.

[5] Ibrahim Hur and Calvin Lin, "Feedback Mechanisms for Improving Probabilistic Memory Prefetching," *In Proceedings of the 15th International Symposium on High Performance Computer Architecture* , 2009.

[6] D. Joseph and D. Grunwald, "Prefetching using markov Predictors," In *Proceedings of the 24th International Symposium on Computer Architecture*, 1997.

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

[8] K. Nesbit, A. Dhodapkar, and J. Smith, "AC/DC: An adaptive data cache prefetcher," In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, 2004.

[9] K. Nesbit and J. Smith, "Prefetching with a global history buffer," In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, 2004.

[10] S. Palacharla and R. Kessler. "Evaluating stream buffers as a secondary cache replacement," In *Proceedings of the 27th International Symposium on Microarchitecture*, 1994.

[11] Luis M. Ramos, José Luis Briz, Pablo E. Ibáñez and Víctor Viñals, "Low-Cost Adaptive Data Prefetching," *Lecture Notes in Computer Science; Springer Berlin / Heidelberg*, 2008.

[12] A. Smith, "Sequential program prefetching in memory Hierarchies," *IEEE Transactions on Computers*, December 1978.

[13] S. Srinath, O. Multu, H. Kim, Y. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, 2007.

[14] S. Verma, D. Koppelman and L. Peng, "A Hybrid Adaptive Feedback Based Prefetcher," In *Proceedings of the 1st JILP/Intel Data Prefetching Championship (DPC-1) in conjunction with HPCA-15*, February 2009.