An Effective Instruction Cache Prefetch Policy by Exploiting Cache History Information

Soong Hyun Shin, Cheol Hong Kim, Chu Shik Jhon

Department of Electrical Engineering and Computer Science,
Seoul National University, Shilim-dong, Kwanak-gu, Seoul, Korea

shordan@panda.snu.ac.kr, kimch@panda.snu.ac.kr,
csjhon@panda.snu.ac.kr

Abstract. The hit ratio of the first level cache is one of the most important factors in determining the performance of embedded computer systems. Prefetching from lower level memory structure is one of the techniques for improving the hit ratio of the first level cache. This paper proposes an effective prefetch scheme for the first level instruction cache by exploiting cache history information. The proposed scheme utilizes two factors to improve the prefetch efficiency: the disparity of block size between memory hierarchies and continuous same page hits. According to our simulations, the proposed prefetching scheme improves the performance by up to 6.3%.

Keywords: computer architecture, embedded processor, instruction cache, cache prefetching

1 Introduction

Minimizing the average data access time is crucial in designing embedded computer systems. As the gap between processor cycle speed and memory access time grows, access time to lower level memory structure has increased dramatically. Accordingly, first level cache plays one of the most important roles in determining the performance of computer systems.

Many researchers have examined how to improve the hit ratio of the first level cache. Prefetching from lower level memory structure is one of the techniques to improve the hit ratio of the first level cache. Prefetching can be implemented by software or hardware. Software-based prefetching needs compiler supports to insert prefetch instructions selectively by analyzing the program. However, it is very difficult to predict the block to be prefetched precisely at compile time, resulting in passing up many prefetching opportunities. Contrary to the software-based prefetching, hardware-based prefetching can use run-time information to predict the block to be prefetched. However, hardware-based prefetching schemes require additional hardware overhead. In this paper, we propose a hardware-based prefetching scheme to improve the hit ratio of the first level instruction cache (iL1) with little hardware overhead.

The hardware-based prefetching schemes are classified into sequential prefetching schemes and non-sequential prefetching schemes. Smith proposed sequential prefetching scheme for the first time[1]. OBL (One Block Lookahead) scheme[1] initiates prefetching only for the sequential block. There are three ways in prefetching by OBL scheme: always-prefetch, prefetch-on-miss and tagged-prefetch. The always-prefetch scheme prefetches the next block on each reference, so it increases the number of cache lookups significantly. The prefetch-on-miss scheme simply prefetches the next block only when one block is required due to a cache miss. The tagged-prefetch scheme is a modified version of the prefetch-on-miss scheme. It associates a tag bit with every memory block, and the tag bit is used to avoid prefetching same block again. Dahlgren et al. proposed an adaptive sequential prefetching scheme, varying the prefetch degree (the number of blocks to prefetch) dynamically[2]. In this scheme, the proportion of useful prefetches to total prefetches is periodically calculated. The prefetch degree is controlled according to the proportion. Jouppi suggested a FIFO stream buffer, which contains prefetched blocks before they are brought into the cache, to avoid cache pollution due to prefetching[3]. Each block in the buffer is brought into the cache if it is referenced, then a new block is pushed into the FIFO stream buffer. The cache pollution is avoided by using this buffer. However, if a cache miss occurs and the requested block is not located in the head entry of the buffer, the buffer is flushed and new sequences of blocks are fetched to the buffer, resulting in no use of prefetched blocks. The fetch directed instruction prefetching scheme[4] proposed by Reinman et al., one of the non-sequential prefetch schemes, utilizes the branch predictor for predicting program stream. In this scheme, the prefetch unit prefetches blocks according to the predicted program stream. Similar to the fetch directed instruction prefetching scheme, the execution history guided instruction prefetching scheme[5] proposed by Zhang et al. utilizes execution history. It is done by correlating execution history with cache miss history. Batcher et al. proposed the cluster miss prediction scheme for embedded cpu instruction caches[6]. The scheme is for real-time networking applications, so they profiles the characters of the dedicated application traces.

In most prefetch schemes, there are two types of references to the lower level memory structure; actual lookup and prefetch lookup. The actual lookup, caused by a cache miss in the first level cache, services normal memory reference. The prefetch lookup is caused by a predicted memory reference. If the prefetch prediction is incorrect, the prefetch lookup is turned out to be unsuccessful. These unsuccessful prefetch lookups increase the contention to the lower level memory structure without any contribution to the hit ratio of the first level cache.

For the iL1, utilizing sequential prefetch is more effective than non-sequential prefetch[7], but sequential prefetches increase the traffic to the lower level memory structure significantly. To reduce the traffic, we propose a prefetch scheme which doesn't request the prefetch lookup to the lower level memory structure separately, but requests the missed block and the next block simultaneously when a cache miss occurs. It is done by simply doubling the block fetch size.

Moreover, the proposed scheme improves the prefetch accuracy by tracing the memory page access pattern of executed instruction sequence.

The rest of this paper is organized as follows. Section 2 describes the motivation of this paper. Section 3 presents the proposed prefetching scheme. Section 4 describes our evaluation methodology and shows detailed simulation results. Section 5 concludes this paper.

2 Motivation

It is commonly regarded that the program control flows sequentially. Generally, branch instructions occupy only $0 - 30\%$ of total instructions[8][9]. Therefore, most instructions are expected to be accessed sequentially in the iL1. We try to exploit benefits from this characteristic of the program. Fig.1 shows the proba-

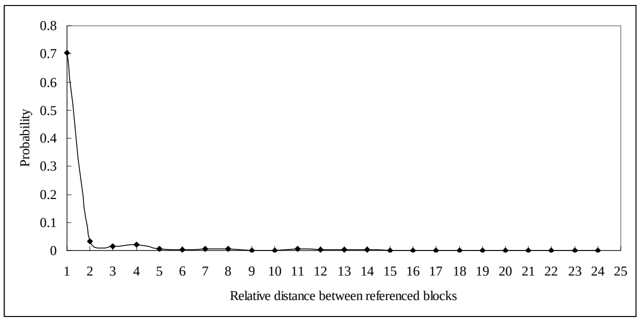


Fig. 1. The probability according to the relative distance between instructions executed.(Instruction cache block size=32B, Memory page size=4KB)

bility according to the relative distance between continuously referenced cache blocks in the iL1, obtained from our simulations using SimpleScalar[10]. We simulated 22 benchmarks from SPEC CPU2000[9]. The relative distance in the graph denotes the distance between continuously referenced cache blocks in the iL1. For example, in case that $n + 1$th instruction is referenced after nth instruction in the iL1, the relative distance is one if the $n + 1$th instruction is within the next block to the block containing nth instruction. In this graph, we excluded the probability of accessing same cache block (the relative distance is zero). The probability (the nth and the $n + 1$th instruction are within the same block in the iL1) is meaningless with respect to prefetching, because the needed block is already located in the iL1. As shown in Fig.1, the probability that the relative distance is one is over 70%. From this result, we realized that the next block is referenced in the iL1 with very high probability after one cache block is referenced. Therefore, the hit ratio of the iL1 is expected to increase if the next block is prefetched from lower level memory structure when one block is requested due to iL1 cache miss. However, prefetching every next cache block is very expensive, because it may increase the bus contention significantly.

To find the way to maximize the prefetch efficiency, we also observed the relative distance between continuously referenced blocks in case that two blocks are within the same page. As shown in Fig.2, the probability of accessing the next block is 84% when executed instructions are within the same page. As shown in Fig.2, if prefetch operations are permitted only when the same page is accessed, the probability of accessing the next cache block is increased by 14% (from 70% to 84%). We propose a prefetch scheme to utilize these characteristics of applications.

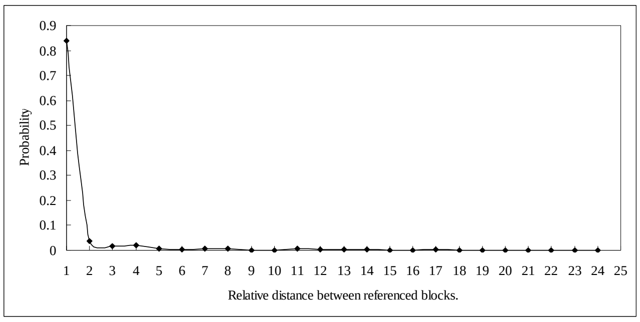
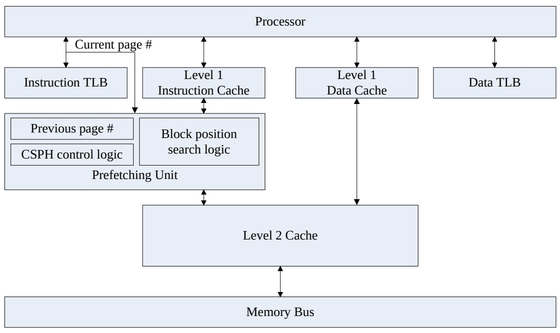


Fig. 2. The probability according to the relative distance between instructions within the same page. (Instruction cache block size=32 B, Memory page size=4KB)

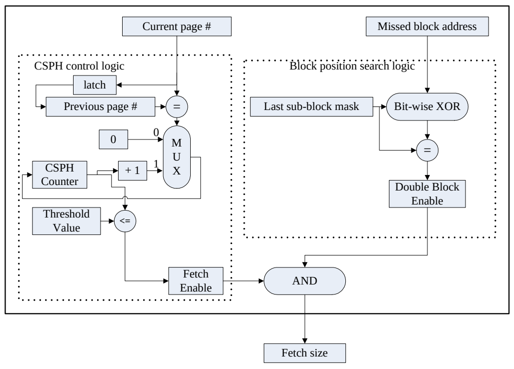
3 Prefetch on Continuous Same Page Access

The *Prefetch on Continuous Same Page Access* (CSPA) scheme, proposed in this paper, initializes prefetch on misses if two conditions are satisfied: (1) the number of continuous same page hits (CSPH) reaches the threshold value, and (2) the block to prefetch isn't the last sub-block in the block of the lower level memory structure. Fig.3 depicts the proposed architecture. We assume two level cache architecture.

The simplest way to predict same page hit is to count the CSPH. If the CSPH exceeds the threshold value which we decide after simulations, we predict that the next instruction is within the page containing the previous instruction. The prefetching unit, placed between iL1 and L2, counts the CSPH to decide whether to prefetch or not. To count CSPH, the *previous page #* is recorded and it is compared with *current page #*. If these are equal, the *CSPH counter* is increased. The *fetch enable* signal is set when the *CSPH counter* reaches the *threshold value*. (Fig.3(b)) This condition raises the probability of successful prefetch (Fig.1, Fig.2).



(a) Overview of the architecture



(b) Precise depiction of the prefetching unit

Fig. 3. Proposed architecture and the prefetching unit

Most prefetch schemes have two types of references; actual lookup and prefetch lookup. For example, if the $A00$ (Fig.4(a)) block is required due to a cache miss in the iL1, the $A00$ is transferred to the iL1 and the $A01$, the next block of the missed block ($A00$), is prefetched by the prefetch lookup. For all prefetches, the L2 cache is looked up separately to the actual lookup. If prefetching the block is unsuccessful(the prefetched block is not referenced), the average memory access time increases by the cost of the prefetch. Even if the prefetch is successful, the memory traffic is not decreased because the prefetch lookup also looks up the L2. To remove this disadvantage, we initialize prefetch requests only when the missed block and the prefetched block are in the same block of lower level memory structure. It is because the L2 block has several iL1 blocks(generally, 4 sub-blocks) and it can service more quickly when requested iL1 blocks are sat

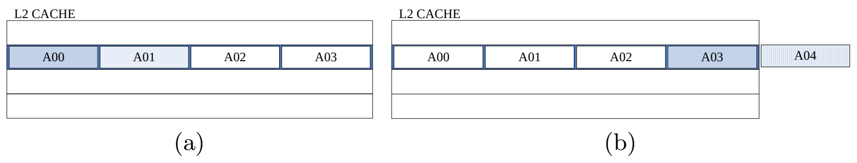


Fig. 4. Examples of allocation of a missed block and a prefetched block

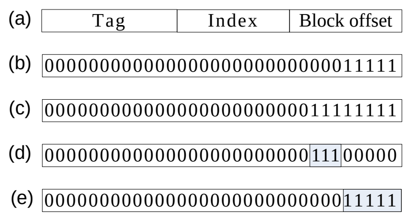

(a) Tag Index Block offset
(b) 00000000000000000000000000011111
(c) 00000000000000000000000011111111
(d) 000000000000000000000000 111 00000
(e) 000000000000000000000000000 11111

Fig. 5. Example of block offset masks and a last sub-block mask

in the same L2 block. Were it not so, the L2 must be accessed again to find the next sub-block. In CSPA scheme, two sub-blocks, $A00$ and $A01$, are requested simultaneously by the iL1 when the $A00$ is requested due to a miss in the iL1. If the prefetching block is successful, the prefetch lookup cost disappears and the memory traffic is decreased by removing prefetch lookup. In this scheme, if these two blocks aren't located in the same block (Fig.4(b)), fetching isn't initialized. In this case, only $A03$ block (without $A04$) is transferred to the iL1, because L2 has to be looked up again to find the block containing the $A04$.

As described, the CSPA scheme requests the missed block and the next block simultaneously when a cache miss occurs and two conditions(as described above) are satisfied. It is done by simply doubling the block fetch size. To make it possible, we assume that the $fetch\ size$ signal(Fig.3(b)) controls whether the L2 returns one sub-block or two sub-blocks.

It is easily noticed whether missed block is the last sub-block of the L2 cache block. The block offset of the L2 cache is divided into two fields: the lower field is a part overlapped by the block offset of the iL1, the upper field is the other part that indicates the sub-block in the L2 cache. Fig.5 shows an example. Memory address is divided into a tag, an index, and a block offset (Fig.5(a)). Fig.5(b) and Fig.5(c) depict the block offset masks of iL1 and L2, respectively. In this example, the iL1 cache block size is 32 bytes and the L2 cache block size is 256 bytes, that is, the bit widths of block offset are 5 and 8, respectively. The lower field of L2 block offset mask, shown in the Fig.5(e), is not used in the L2. L2 cache is searched by the upper field (Fig.5(d)) to find the sub-block which is requested by the first level cache. If the higher field is filled with '1', the requested block is the last sub-block of the L2 cache block. As shown in the Fig.5(d), if we apply XOR to the block offset of the iL1 and L2, the result is the $last\ sub-block\ mask$, indicated the upper field.

Consequently, through this operation, if the missed block is not the last sub-block, the *double block enable*(Fig.3(b)) is set. If all two enable signals, the $fetch$ *enable* and the *double block enable*, are set, $fetch\ size$ is set and the prefetch unit requests two blocks from the L2 cache(Fig.3(b)).

4 Performance Evaluation

4.1 Simulation Model

In order to determine the performance of the proposed prefetching scheme, we simulatated various benchmarks using SimpleScalar simulator[10]. Simulated applications are selected from SPEC2000 suite[9]. Simulated system parameters are shown in Table 1.

Table 1. System Parameters

System Parameter	**Value**
Processor	2-way superscalar processor
Level 1 instruction cache	16KB, 4way, 32bytes cache line, 1cycle latency
Level 1 data cache	16KB, 4way, 32bytes cache line, 1cycle latency, write-back
Level 2 cache	unified, 256KB, 8way, 256bytes cache line, 8cycle latency
Memory latency	32cycle latency + 1 cycles/8bytes

4.2 Simulation Results

At first, we simulated to determine the threshold value for the CSPH. Fig.6 shows the iL1 cache miss ratio and normalized execution time for various threshold values. The execution time is normalized to no-prefetch scheme. In the graph, 'CSPA-n' means that the threshold value is n. As shown in Fig.6, the CSPA-1, CSPA-2, and CSPA-4, show better performance than the others. We choose '4' for the threshold value among 1, 2, and 4 because the total number of prefetched instructions in the CSPA-4 scheme is less than the others. If the threshold value is too high, the performance degrades. As shown in Fig.6, as the threshold value increases, the iL1 miss ratio and the execution time increase. It is because, the successful prefetch ratio increases slightly as the threshold value increases, but the un-duplicated ratio decreases conspicuously as shown in Fig.7. The successful prefetch ratio is the proportion that prefetched block is used in the iL1 before it is evicted from the iL1, and the duplicated prefetch ratio is the proportion that prefetched block is useless because the block is already located in the iL1.

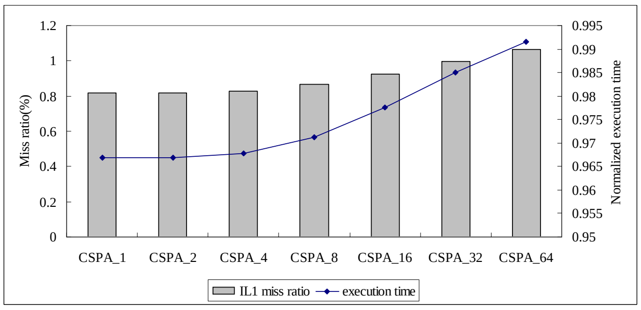


Fig. 6. Level 1 instruction cache miss ratio and normalized execution time. The execution time is normalized to no-prefetch scheme

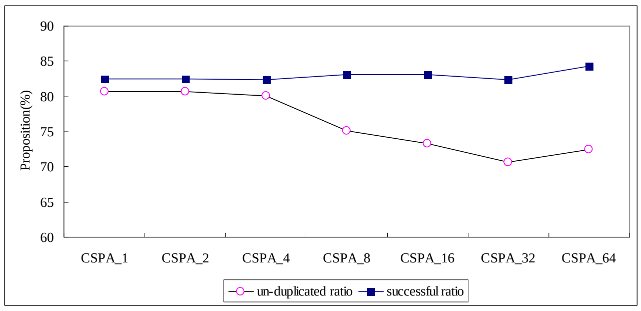


Fig. 7. The un-duplicated ratio and the successful ratio of prefetch for various prefetch configurations.

It is because while the CSPH increases (instructions within the same page are referenced continuously), the probability increases that the prefetched blocks are already referenced. Besides we found the probability of accessing the next block is 84% when executed instructions are within the same page, as shown in the Fig.2, and the successful prefetch ratio of the CSPA is $82 - 84\%$. It means that the CSPA utilizes the sequential characteristic of benchmarks so effectively to maximize the prefetch efficiency and the successful prefetch ratio doesn't increase any more.

We compare the CSPA with the no-prefetch scheme and the prefetch-on-miss scheme, because the CSPA needs hardware overhead comparable to that of these two schemes, and other schemes like[4], [5], and [6] uses tables to keep informations which are heavier than the hardware cost of CSPA. In the iL1, the sequential prefetch scheme outperforms the non-sequential scheme[7] and hardware cost of sequential prefetch is less than that of non-sequential prefetch.

Fig.8 and Fig.9 show the execution time and the miss ratio in the iL1. As shown in Fig.8, the performance of the CSPA is superior to other schemes. The

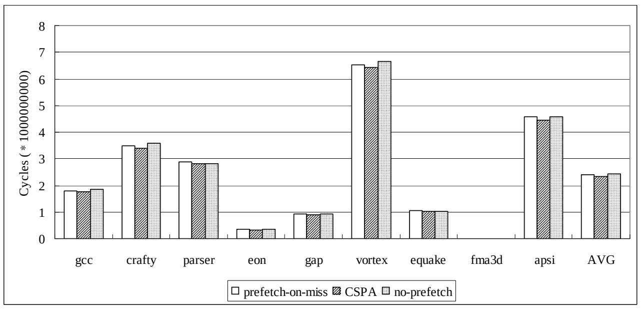


Fig. 8. The execution time for SPEC2000 benchmarks

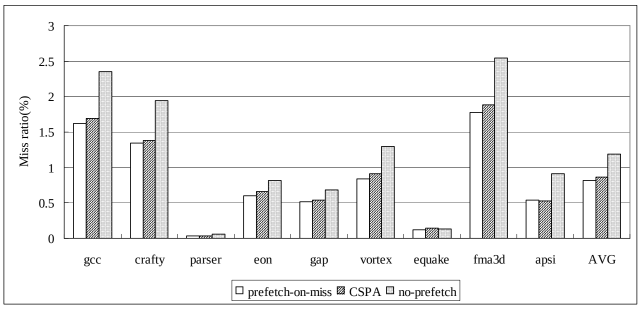


Fig. 9. The miss ratio for SPEC2000 benchmarks

CSPA outperforms the no-prefetch by 3.3% on the average and especially in the benchmarks, crafty and equake, the performances increase 5.3% and 6.3%,respectively. The CSPA also outperforms the prefetch-on-miss by 2.4% on the average. The miss ratio of the no-prefetch is 1.14% and that of the CSPA is 0.82%. The CSPA reduces miss ratio by 0.3%. Though the miss ratio of the prefetch-on-miss is less than that of the CSPA, the execution time of the CSPA is better. It is because the prefetch-on-miss performs prefetches 14% more than the CSPA, as shown in Fig.10. It is shown well in the benchmarks, gcc, crafty and vortex. As compared with the prefetch-on-miss scheme, the miss ratio of the CSPA is higher by 0.1% and the execution time is less by 2%. The reason of this conflicting result is that the CSPA reduces the total number of prefetch by 15.6%.

5 Conclusions

We have proposed the prefetch on continuous same page access scheme, which utilizes the disparity of block size between iL1 and the lower level memory structure and continuous same page hits. The simulation shows that the CSPA scheme

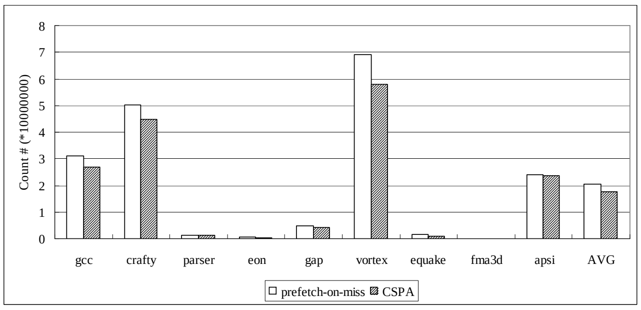


Fig. 10. The total number of prefetch for SPEC2000 benchmarks

outperforms the no-prefetch policy by 3.3% on the average. The CSPA outperforms the prefetch-on-miss, and moreover it reduces the total number of prefetch by 14%. Simulation results show that the CSPA scheme is the most effective iL1 prefetching scheme among three compared schemes.

References

1. Smith, A.: Cache memories. ACM Computing Surveys **14** (1982) 473–530
2. Dahlgren, F., Dubois, M., Stenström, P.: Fixed and adaptive sequential prefetching in shared memory multiprocessors. In: International Conference on Parallel Processing. (1993) 56–63
3. Jouppi, N.P.: Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In: International Symposium on Computer Architecture. (1990) 364–373
4. Reinman, G., Calder, B., Austin, T.: Fetch directed instruction prefetching. In: International Symposium on Microarchitecture. (1999) 16–27
5. Zhang, Y., Haga, S., Barua, R.: Execution history guided instruction prefetching. In: International Conference on Supercomputing. (2002) 199–208
6. Batcher, K., Walker, R.: Cluster miss prediction with prefetch on miss for embedded cpu instruction caches. In: International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. (2004) 24–34
7. Hsu, W.C., Smith, J.E.: A performance study of instruction cache prefetching methods. IEEE Transactions on Computers **47** (1998) 497–508
8. Lee, C., Potkonjak, M., Mangione-Smith, W.H.: Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In: International Symposium on Microarchitecture. (1997) 330–335
9. SPEC: (SPEC CPU2000 Benchmarks) http://www.specbench.org.
10. Burger, D., Austin, T.M., Bennett, S.: Evaluating future microprocessors: the simplescalar tool set. Technical Report TR-1308, Univ. of Wisconsin-Madison Computer Sciences Dept. (1997)