Instruction cache prefetching directed by branch prediction

J.-C.Chiu, R.-M.Shiu, S.-A.Chi and C.-P.Chung

Abstract: As the gap between processor speed and memory speed grow, so the performance penalty of instruction cache misses gets higher. Instruction cache prefetching is a technique to reduce this penalty. The prefetching methods determine the target line to be prefetched generally based on the current fetched line address. However, as the cache line becomes wider, it may contain multiple branches. This is a hurdle which must be overcome. The authors have developed a new instruction cache prefetching method in which the prefetch is directed by the prediction on branches, called branch instruction based (BIB) prefetching; in which the prefetch information is recorded in an extended BTB. Simulation results show that for commercial benchmarks, BIB prefetching outperforms traditional sequential prefetching by 7% and other prediction table based prefetching methods by 17% on average. As BTB designs become more sophisticated and achieve higher hit and accuracy ratios, BIB prefetching can achieve a higher level of performance.

1 Introduction

As the gap between processor speed and memory speed grows, the performance penalty of instruction cache misses gets higher. Furthermore, modern microprocessor designs increase the instruction issue rate by employing techniques such as superscalar processing and superpipelining. The performance degradation caused by instruction cache misses becomes more vital. To achieve a high clock rate, the on-chip instruction cache usually has a limited size, ranging from 4Kbytes to 32Kbytes, and low associativity. This increases the cache miss rate.

Instruction cache prefetching is one of the methods that significantly reduces the penalty caused by instruction cache misses. The method attempts to prefetch an absent line that the CPU may use in the near future, from the lower memory hierarchy. Prefetching can be done simultaneously with normal program execution, avoiding unnecessary idling of the CPU [1].

Several instruction cache prefetching techniques have been proposed in the past. Smith [1] explored sequential prefetching, which prefetches the sequentially next line of the current instruction. Smith and Hsu [2] proposed target prefetching, which prefetches the line predicted by a target line prediction table. They further proposed a combined form of sequential prefetching and target prefetching known as hybrid prefetching, which prefetches both the sequential line and the target line predicted by the target line prediction table. Park and his colleagues [3] extended hybrid prefetching to become PBN prefetching by introducing a non-referenced prefetch (NRP) cache to buffer the prefetched lines that were not referenced by the CPU. Target prefetching, hybrid prefetching, and PBN prefetching all use a target line prediction table. They can be categorised as prediction table based prefetching methods. Chen and his colleagues [4] had proposed the BP-based prefetching scheme which uses the program counter of a branch instruction to predict the target line. The BP-based prefetching scheme is similar to the prediction-table-based prefetching methods, but uses the branch predictor to compute the target line address.

The prediction-table-based prefetching methods determine the target line to be prefetched based on the current line address. However, as the cache line becomes wider, there may be multiple branches, confusing the decision made by the prediction-table-based prefetching methods and leading to heavy hardware complexity when seeking the branch instruction. In this paper, we develop a different instruction cache prefetching method in which the prefetch is directed by the prediction of the branches. We call it branch instruction based (BIB) prefetching. BIB prefetching works according to the prefetch information saved in an extended branch target buffer (EBTB). Simulation results show that, even with the traditional small cache line size, BIB prefetching achieves higher performance than sequential prefetching and other prediction-table-based-prefetching methods.

2 Previous research

Sequential prefetching prefetches the sequentially next cache line upon the first reference to the current line. It exploits the code sequentiality, which has been found to be an important characteristic of typical programs [1]. To prevent the cache from being polluted by the prefetched line, which may never be referenced, a stream buffer [5] can be introduced to buffer the lines prefetched on a cache miss. The stream buffer works as a FIFO queue. When a miss occurs, the lines next to the missed line are prefetched and placed in the stream buffer until the stream buffer is full or another miss occurs. Upon fetching an instruction,

IEE Proceedings online no. 19990310

DOI: 10.1049/IP-CDT:19990310

Paper first received 19th May 1998 and in revised form 18th January 1999

The authors are with the Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, R.O.C.

E-mail: cpchung@arch3.csie.nctu.edu.tw

only the tag in the first entry of the buffer need be compared against the fetch address. A miss will cause the buffer to be flushed even if the requested line is already present in the stream buffer, and the prefetching will be restarted at the miss address.

Target prefetching overcomes the inability of sequential prefetching, i.e. only the sequentially-next line can be prefetched. It uses a target line prediction table to predict the address of the target line to be prefetched [2]. The table contains two fields: the *current line address*, and the *target line address*. The target line address field stores the fetch address to be used in the next cycle, which was the previous next line address of the current line. When the program counter changes to a new line, the line address of the program counter is used to search the table. If an entry matches, the target line address of the entry is used for prefetching. No prefetching request will be issued if the search is a miss.

In hybrid prefetching [2], both the target line and the sequentially-next line are prefetched, providing better protection against a cache line miss. Hybrid prefetching increases memory traffic because it has two candidate lines to prefetch.

Park and his colleagues [3] extended hybrid prefetching to PBN prefetching by adding a non-referenced prefetch (NRP) cache to store the prefetched lines that were not referenced by the CPU. Because these prefetched lines may be referenced by the CPU later, storing these lines in the NRP cache can reduce both later possible cache misses and memory traffic. As in hybrid prefetching, PBN prefetching prefetches both the target line and the sequentially-next line. Moreover, it considers both the line next to the sequentially next line, and the line preceding the target line, as candidate lines. According to an additional history bit in the target line prediction table, these four candidate lines may be prefetched according to priority.

In PBN prefetching, the stream buffer is extended to become a prefetch buffer. Every line coming from the lower level memory is placed in the prefetch buffer first, regardless of whether or not it is a prefetched line. Upon an instruction fetch, the tags of all entries in the prefetch buffer are compared against the fetch address. If there is a hit, the hit line can be supplied by the prefetch buffer without delay and then transferred to the instruction cache in the next cycle. A line can be present in either the cache or the prefetch buffer but not both. A T bit is appended to each entry to determine if the corresponding prefetch buffer entry is to be transferred to the NRP cache.

3 Branch instruction based (BIB) prefetching

The prediction-table-based prefetching, such as target, hybrid or PBN prefetching, has two significant drawbacks. Firstly, it uses the cache line as a whole to predict the next accessed line, thus incorrect prediction may be made when there is more than one control transfer instruction in a single cache line. Secondly, it requires significant additional die size to accommodate the target line prediction table. To overcome these drawbacks, we develop a new prefetching method, called branch instruction based (BIB) prefetching.

The block diagram showing the implementation of BIB prefetching is shown in Fig. 1. A traditional branch target buffer (BTB) [6] is extended to provide additional prefetching information, and is known as extended BTB (EBTB). Similar to BTB, EBTB is associated with the instruction fetch stage, and is searched with the addresses

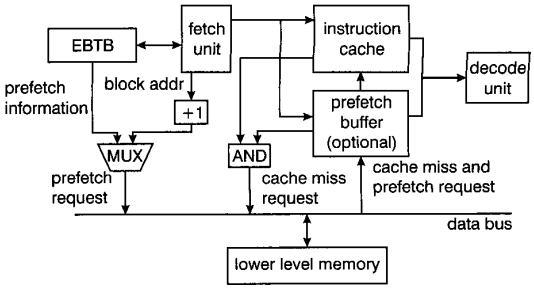


Fig. 1 *Block diagram for implementing BIB prefetching*

of the instructions fetched each cycle to ensure the fetched instruction is a branch. If it is, it will predict the instruction to fetch in the next cycle. Moreover, if the search is a hit, the prefetching information of the hit entry will be used for choosing the candidate instruction line to be prefetched. If the search is a miss, then the sequentially-next instruction line will be chosen as the prefetch candidate. An optional prefetch buffer, which is similar to that in PBN prefetching [3], can be added to avoid pollution of the instruction cache by the prefetched lines. The T bit in the prefetch buffer of PBN prefetching is not needed here because there is no NRP

3.1 Extended BTB

BIB prefetching uses the information of previous branch history stored in EBTB. The organisation of EBTB is shown in Fig. 2. The prefetch line address in EBTB is recorded according to the previous execution trace of this branch instruction.

This may be designed to prefetch the dth line ahead which is expected to be referenced by the CPU. d is the degree of lookahead prefetching. We apply this concept to BIB prefetching with some modification and call the result lookahead basic-block prefetching. Lookahead basic-block prefetching prefetches the first line of the basic block previously executed following the EBTB predicted basic block. We do not prefetch the first line of the EBTB predicted basic block because this prefetch is often unnecessary and the performance gain is too small. We do not prefetch the first line of the third or further, basic block either, as those prefetched lines are not often used.

In Fig. 3, a binary-tree-like structure gives an abstract representation of a program execution flow. Each square in this figure stands for a basic block. Inside each square, 'Branch x' stands for the last, control transfer instruction of the basic block, and '*inst n*' stands for the first instruction of that basic block. When branch A hits an entry in EBTB, the actions that the EBTB takes are similar to those of a conventional BTB, except that it will start to prefetch according to the line address contained in the prefetch line address field. Suppose that branch A is taken, and the execution result of branch B will determine which of the two instructions, *inst* 1 or *inst* 2, will be executed next. The result of branch B will write a line address-the line address

tag	taken address	history	prefetch line address
	⋮		

Fig. 2 *EBTB of BIB prefetching*

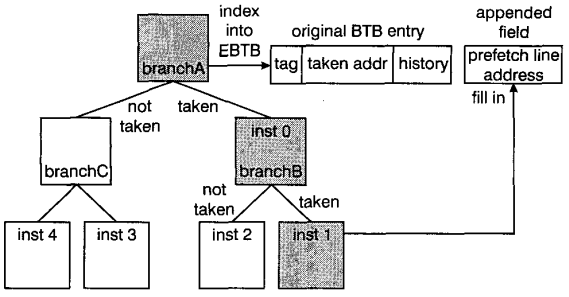


Fig. 3 *Abstract representation of a program execution flow and how it affects the EBTB*

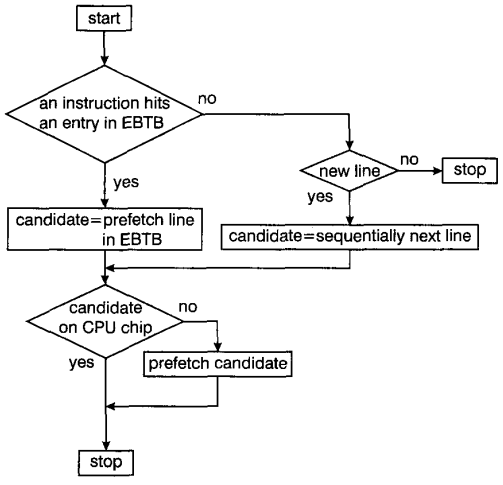


Fig. 4 *Flowchart of BIB prefetching*

of either *inst* 1 or *inst* 2-into the prefetch line address field of the EBTB entry of branch *A*. If branch *A* does not hit an entry in EBTB, BIB prefetching uses the next sequential line address instead.

In BIB prefetching, because there are two instruction storage areas: the instruction cache and the prefetch buffer, prefetching is invoked only when the requested line does not exist in either of the two storage areas. One machine cycle is assumed for EBTB and on-chip storage lookup. The BIB prefetching flow chart is shown in Fig. 4. Prefetching is initiated when an instruction hits an entry in EBTB, or when a new line is used by the fetch unit. There is only one prefetching candidate in each case. The target line has higher priority over the sequentially-next line.

3.2 Maintaining prefetching information

We use a two-entry FIFO queue mechanism to implement the lookahead basic-block prefetching nature of BIB prefetching. In the instruction decode stage; if an instruction is a branch, its EBTB index value is inserted into the FIFO queue. After the execution stage of the branch instruction, the line address of the upcoming basic-block will be written into the EBTB entry indexed by the top entry of the FIFO queue. To simplify the two-entry FIFO queue design, a modified maintaining mechanism of the FIFO queue is shown in Fig. 5. The T and T' bits are toggles. The value of T' is the inverse of T. C is a check bit that can be set to 1 or reset to 0. 'EBTB index 0' and 'EBTB index 1' are the two entries of the FIFO queue.

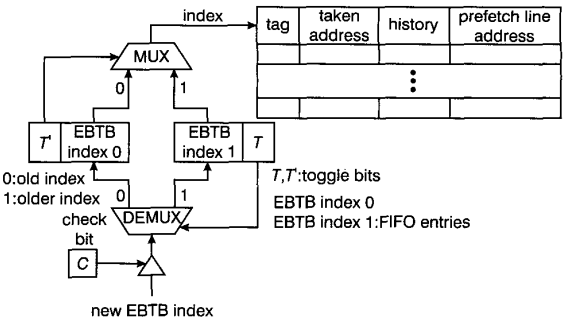


Fig. 5 *Maintaining mechanism of the target line address with modified FIFO queue*

The prefetch line address of the EBTB entry of a previous branch instruction is updated only when the target address of the current branch instruction is known. Whenever a branch instruction is recognised, the index value of the corresponding EBTB entry is inserted in the FIFO queue entry selected by the T bit, which is the inserted entry of the FIFO queue, if the check bit C is set. Each time a new EBTB index value is inserted into a FIFO entry, the check bit is reset. The check bit is used to prevent the FIFO queue from being written by the second or further BTB index values. There will not be a serious performance loss when two or more branch instructions appear in a row, because there is high possibility that these instructions reside on the same cache line. Prefetching according to the first branch instruction will also bring the other contiguous branch instructions into the on-chip storage.

After the execution stage of the current branch instruction, the next instruction address of the current branch instruction is known. The index value selected by the T' bit, which is in the top entry of the FIFO queue, is then used to update the prefetch line address of the previous branch instruction. After this update, the two toggle bits, T and T', are toggled, and the check bit C is set.

4 Experiments

To study the efficiency of the various instruction cache prefetching methods, we have built a trace-driven simulator for experiments.

4.1 Simulation method

A trace-driven simulator on DEC-station 5133 was built to evaluate the performance of the various instruction cache prefetching methods. The benchmark program traces are generated by the pixie utility on the DEC-station [8].

Six benchmark programs are selected from the SPEC95 integer benchmarks. Table 1 lists these benchmark programs. For each benchmark trace, only the first twenty millon instructions are used to measure the performance of each prefetching method. This is a feasible reduction in workload: the performance behavior of every prefetching method reaches a stable state at an instruction count of twenty million.

The most frequently used instruction cache performance metric is the miss ratio. However, it is not suitable for instruction cache prefetching studies [3]. If a line to be prefetched cannot be returned from lower level memory before it is needed by the CPU, it is a miss, but a few cycles are saved due to this prefetching request. The miss ratio

Table 1: Benchmark programs

Benchmark	Short description
go	artificial intelligence; plays the game of "Go"
gcc	new version of GCC; build SPARC code
compress	compress and decompress file
li	LISP interpreter
ijpeg	graphic compression and decompression
vortex	a database program

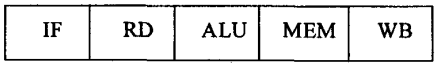
cannot reflect this situation. Another metric, called MCPI Memory Accesse Cycles Per Instruction, is used as the performance metric [9]. We assume that the processor can execute each instruction in one cycle (perfect pipelining), and the only extra contributor of CPI is due to instruction cache access penalty. The MCPI is defined as:

$$\text{MCPI} = \frac{\text{total instruction access penalty}}{\text{number of instructions executed}}$$

4.2 Machine model specification

The machine model we use in this study is a pipelined microprocessor with five stages, similar to the R3000 processor [10], as shown in Fig. 6. Some necessary assumptions are made on this machine model to make it suited to the experiments:

1. The processor is perfectly pipelined and executes each instruction in one cycle.
2. A fetch miss request has higher priority than a prefetch request.
3. There is an on-chip instruction cache with an access time of one cycle, and a line size of 16 bytes. The size and the associativity of the instruction cache are input parameters in the experiments. We examined the sizes of 2, 4, 8, and 16KB; and the associativities of direct-mapping, two-way and four-way.
4. There is a unified level-two cache which has a hit rate of 100%. The latency of the L2 cache is three cycles, and the L2 cache is non-pipelined and non-pre-emptive.
5. The bus width between L1 and L2 caches is the same as the line size.
6. BTB is associated with the IF stage to predict if the instruction fetched is a branch and to fetch the next instruction address. The BTB has 1024 entries and is 4-way set-associative with the LRU replacement. The history field of the BTB uses a two-bit saturation counter.
7. One cycle is assumed for the instruction fetch request and the prefetch lookup in on-chip instruction cache.
8. The prefetch buffers of the PBN and the BIB prefetching have eight entries and are fully-associative.
9. The NRP cache of PBN prefetching is four-way set-associative and has 1024 bytes. The prediction table of the PBN prefetching is a 1024 entry four-way set associative branch target buffer-like cache.



- **IF : fetch the instruction (I-cache).**
- **RD : read any required operands from CPU registers while decoding the instruction.**
- **ALU : perform the required operation on instruction operands.**
- **MEM : access memory (D-cache).**
- **WB : write back results to register file.**

Fig. 6 *Pipeline stages of the proposed machine model*

4.3 Simulation results

In this section, we examine the performance of four instruction cache prefetching methods including no prefetching, sequential prefetching, PBN prefetching, and BIB prefetching. PBN prefetching is the best known case of the prediction-table-based prefetching methods.

Some abbreviations are used in the simulator. The four instruction cache prefetching methods are denoted 'normal', 'sequential', 'PBN', and 'BIB', respectively. The symbol 'associativity size' is used to indicate the associativity and size of the instruction cache. The associativity part is denoted 'D' for direct-mapping and '*n*-way' for *n*-way set-associativity; and the size part is denoted '*m*' for *m*Kbytes cache. For example, 'D_2' means the cache is direct-mapped and 2K bytes, and '4-way_16' means the cache is 4-way set-associative and 16K bytes.

The performance of these methods according to MCPI against each benchmark are shown in Figs. 7–12. BIB prefetching is the most effective in most cases. The average MCPI of the six benchmarks are shown in Fig 13. The BIB prefetching method outperforms sequential prefetching by 7%, and PBN prefetching by 17%, on average.

The average memory bus traffic for each prefetching method is shown in Fig. 14. The method that results in the heaviest traffic is PBN prefetching, and BIB prefetching generates about 28% more memory traffic than the direct

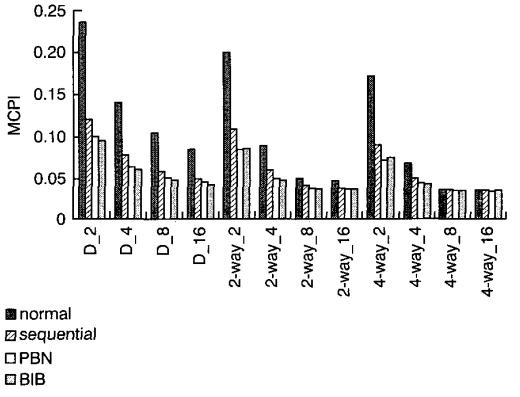


Fig. 7 *MCPI of 'li' benchmark program*

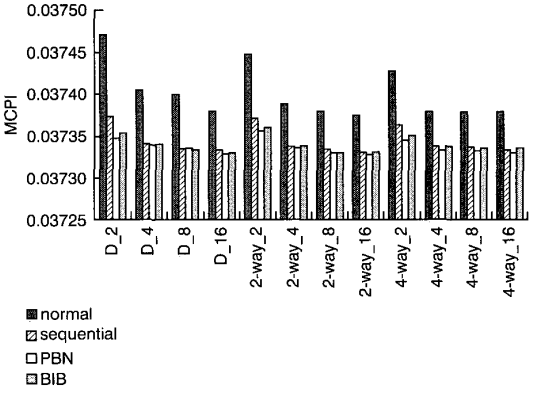


Fig. 8 *MCPI of 'compress' benchmark program*

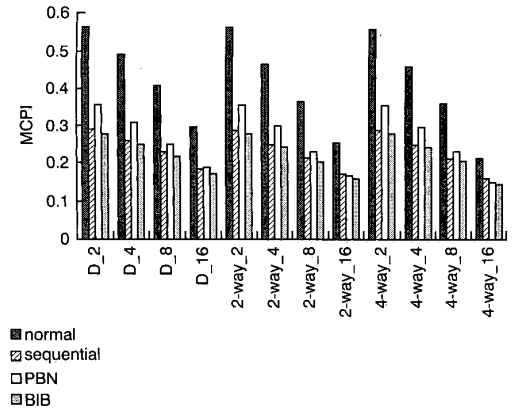


Fig. 9 *MCPI of 'gcc' benchmark program*

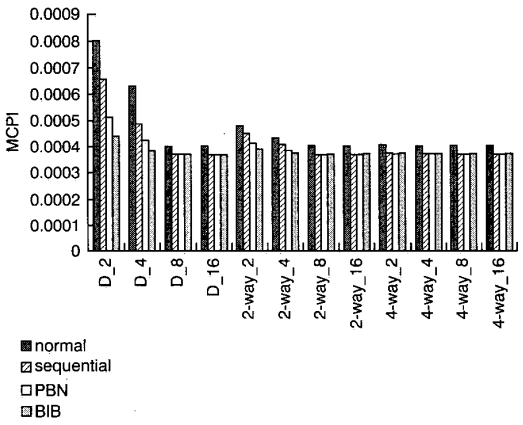


Fig. 10 *MCPI of 'ijpeg' benchmark program*

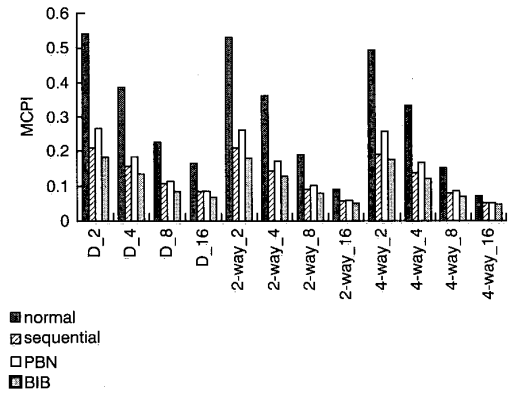


Fig. 11 *MCPI of 'vortex' benchmark program*

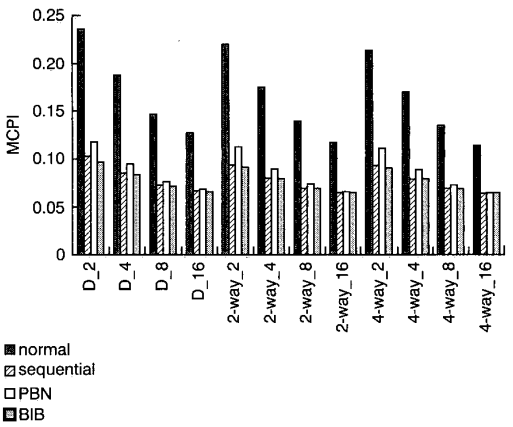


Fig. 12 *MCPI of 'go' benchmark program*

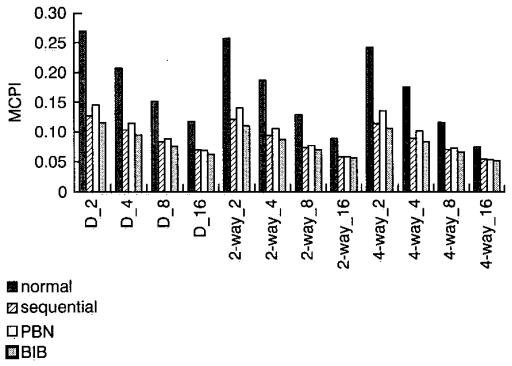


Fig. 13 *Average MCPI*

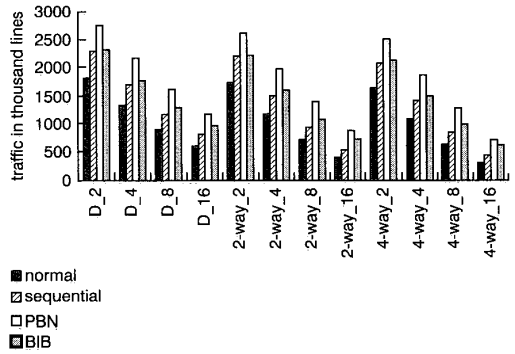


Fig 14 *Average memory traffic*

mapped normal cache with a cache size of 2K. The amount of traffic which BIB prefetching generates is closely correlated to BTB accuracy. High BTB accuracy will result in high prefetching accuracy in the BIB prefetching method, and as a consequence the memory traffic overhead will be lowered or even completely eliminated.

Note that sequential prefetching is very effective, and the BIB approach only shows a small improvement relative to sequential prefetching. Two factors have caused this:

1. The frequency of not-taken conditional branches is 45.98% in SPECint95 [11]. Sequential prefetching prefetches contiguous lines without being affected by cache miss penalties, and no additional memory traffic or instruction cache pollution will result.

2. The targets of more than 50% of branches in SPECint95 are only 2 to 15 instructions away [11]. This useful characteristic of spatial locality makes sequential prefetching perform well without suffering from the penalties contributed by the high ratio of branch instructions in benchmarks. Because the target of a branch instruction is close to the branch instruction, there is a good possibility that the target instruction already exists in the instruction cache.

Besides the reasons stated above, which will affect the performance of all prefetching methods, other benchmark

behaviours deserve attention. If the ratio of branch instructions in a benchmark is too small, sequential prefetching works well because of the good sequentiality characteristic in benchmark programs. In the MCPI diagrams for benchmarks 'vortex' and 'go', which have a smaller branch instruction ratio, the sequential prefetching performance is comparable to that of the BIB prefetching method.

The BIB prefetching method uses a lookahead basic-block prefetching mechanism which starts to prefetch whenever EBTB recognises a branch instruction. It works well if (both) EBTB hits and the prediction ratios are high. The EBTB used in our simulation is based on a conventional BTB that has relatively low hit and accuracy ratios. For this reason, in the MCPI diagrams for benchmarks 'vortex', 'gcc', and 'go', which have lower BTB hit and accuracy ratios, the performance of the BIB prefetching method is not significantly better.

From the average MCPI diagram shown in Fig. 14, it is seen that there is less need to employ any prefetching mechanism if the on-chip instruction cache is large and the associativity is high, while a smaller cache with BIB prefetching can do just as well as a large cache. For example, a 2KB, direct mapped cache, with BIB prefetching can achieve better performance than that of a 16KB, direct mapped cache without prefetching, or an 8KB, four-way set associative cache without prefetching.

Note that PBN prefetching does not perform well in all cases. The main reason for this is that PBN prefetching uses the cache line address as the key to search the prediction table. The resulting prefetching prediction accuracy is too low, because there may be more than one branch instruction on a cache line. The lines prefetched from the L2 cache are mostly not used by the CPU; as a consequence, PBN prefetching cannot achieve the anticipated performance even with an NRP cache.

5 Conclusions

We have studied the problem of on-chip instruction cache miss. Instruction cache prefetching is a method that can significantly reduce the penalty caused by instruction cache misses. Many prefetching methods have previously been proposed. In this paper, we have put forward a new mechanism, known as BIB prefetching that, in benchmarks, prefetches instruction cache lines based on a branch instruction one basic block ahead. Simulation results show that our method achieves better performance than other prefetching methods. The BIB prefetching method outperforms sequential prefetching by 7%, and other prediction-table-based prefetching methods by 17% on average.

The simulation results also show that there is no need to prefetch if the on-chip instruction cache is large and the associativity is high. However, the BIB prefetching method with a small cache size can achieve performance equal to that of a larger cache. It requires only a small increase in die size to achieve better performance than other prefetching methods.

The prediction-table-based prefetching methods do not perform highly enough because they use cache line addresses as the key for searching the prediction table. The prefetching prediction accuracy is low because there may be more than one branch instruction in a line. Even with an NRP cache, the prefetched lines are rarely used by the CPU.

The BIB prefetching method is highly dependent on the BTB hit and accuracy ratios. In this research, we used only a simple BTB. As the BTB designs become more sophisticated and achieve higher hit and accuracy ratios, the performance of BIB prefetching can also be enhanced.

The instruction cache design is a major performance factor in modern microprocessor designs. Furthermore, the trends of ever-increasing clock rate and system-on-a-chip both restrict the size and associativity of on-chip caches. Instruction cache prefetching will become increasingly important. Currently, we are planning to extend the BIB prefetching method to other sophisticated hardware branch prediction mechanisms, and use software to assist the prefetching.

6 Acknowledgments

This paper presents partial result of a long-term research project financed by both NSC of R.O.C. under contract no. NSC 85-2622-E-009-010R, and by the industry.

7 References

1 SMITH, A.J.: 'Sequential program prefetching in memory hierarchies', *IEEE Comput.*, 1978, 11, (12), pp. 7–21
2 SMITH, J.E., and HSU, W.C.: 'Prefetching in supercomputer instruction caches', *Supercomputing* '92, 1992, pp. 588–597
3 PARK, G.H., KNOW, O.Y., HAN, T.D., and KIM, S.D.: 'Non-referenced prefetch (NRP) cache for instruction prefetching', *IEE Proc. Comput. Digit. Tech.*, 1996, 143, (1), pp. 37–43
4 CHEN, I.-C.K., LEE, C.-C., and MUDGE, T.N.: 'Instruction Prefetching Using Branch Prediction Information', International Conference on *Computer Design*, October 1997, Austin, Texas
5 JOUPPI, N.: 'Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers', 17th ISCA, 1990, Seattle, WA, pp. 364–373
6 LEE, J.K., and SMITH, A.J.: 'Branch prediction strategies and branch target buffer design', *Computer*, 1984, 7, (1), pp. 6–22
7 KIM, S.B., PARK, S.H., PARK, M.S., KIM, J., MIN, S.L., JEONG, D.K., SHIN, H., and KIM, C.S.: 'Threaded prefetching: An adaptive instruction prefetch mechanism', *Microprocess. and Microprogr.*, 1993, **39**, (1), pp. 1–15
8 SMITH, M.D.: 'Tracing with pixie', Technical Report of Stanford CA 94305-4070, 1991
9 CHEN, T.F.: 'An effective programmable prefetch engine for on-chip caches', Proceedings of MICRO-28, 1995, pp. 237–242
10 KANE, G.: 'MIPS RISC Architecture' (Prentice Hall, 1988)
11 CHANG, S.E., and CHANG, Y.R.: 'A study of SPEC CPU95 benchmarks'. Technical Report of Chung Yuan University, Taiwan, 1996