Access Map Pattern Matching for Data Cache Prefetch

Yasuo Ishii
NEC Corporation
1-10, Nisshin-cho
Fuchu-shi, Tokyo, Japna
y-ishii@bc.jp.nec.com

Mary Inaba
The University of Tokyo
7-3-1, Hongo
Bunkyo-ku, Tokyo, Japan
mary@is.s.u-tokyo.ac.jp

Kei Hiraki
The University of Tokyo
7-3-1, Hongo
Bunkyo-ku, Tokyo, Japan
hiraki@is.s.u-tokyo.ac.jp

ABSTRACT

A novel data prefetching method — access map pattern matching (AMPM) — that uses "memory access map" is proposed. The AMPM prefetching concentrates hardware resources on collecting the access footprint of the frequently accessed area which we called "hot zones". 2-bit state is associated with each cache line of hot zone. A set of these states is called "memory access map". Prefetch requests are generated from the pattern matching of the memory access map. The pattern matching detects multiple memory access patterns in parallel and generates more prefetch requests than conventional prefetchers. The evaluation result shows that the AMPM prefetcher improves performance by 42.0% in FP benchmarks.

Categories and Subject Descriptors

C.1.0 [**Processor Architectures**]: General

General Terms

Design, Experimentation, Performance

Keywords

Cache memory, Data prefetch

1. INTRODUCTION

With the rapid advances in semiconductor process technology and microarchitecture, the speed gap between the clock cycle time of processor cores and that of memory systems have significantly increased. The large speed gap often degrades the performance. Cache memory and data prefetching is a mechanism used to increase the performance of the system with such large speed gap.

2. ACCESS MAP PATTERN MATCHING

In this paper, we propose a novel data prefetching method — access map pattern matching (AMPM) prefetch. The AMPM prefetcher divides the memory address space into fixed size areas as a management unit which we call "zone". Recently accessed zones are detected as "hot zones". The concept of hot zone is similar to the concentrated zone (*Czone*) [4]. It stores the access states of each cache line of corresponding zone (figure 1). Each hot zone has its own "memory access map". The memory access map stores the access footprint but does not store any access order nor instruction address. The number of hot zones and memory access maps is fixed. These maps are stored in a memory access map table. The maps are replaced by least recently used (LRU) policy. The AMPM prefetcher detects the prefetch candidates by parallel pattern matching of the memory access map and generates prefetch requests.

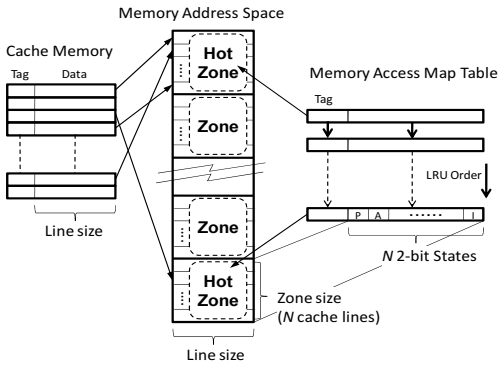

Figure 1: Overview of the AMPM Prefetch

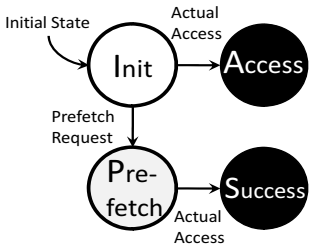

Figure 2: State Diagram of the Access Map

The memory access map is a 2-bit state map that holds the previous access information of all cache lines in the corresponding zone. The state diagram of a memory access map is shown in figure 2.

The AMPM prefetcher searches prefetch candidates by pattern matching. The basic concept of the pattern matching is based on stride detection. Prefetch requests are generated in the following steps. The example of forward prefetching is shown in figure 3. First, the memory access map (A) is shifted for aligning the map, and an actual accessed address is aligned at the edge of the access map (B).

ICS'09, June 8–12, 2009, Yorktown Heights, New York, USA.
ACM 978-1-60558-498-0/09/06.

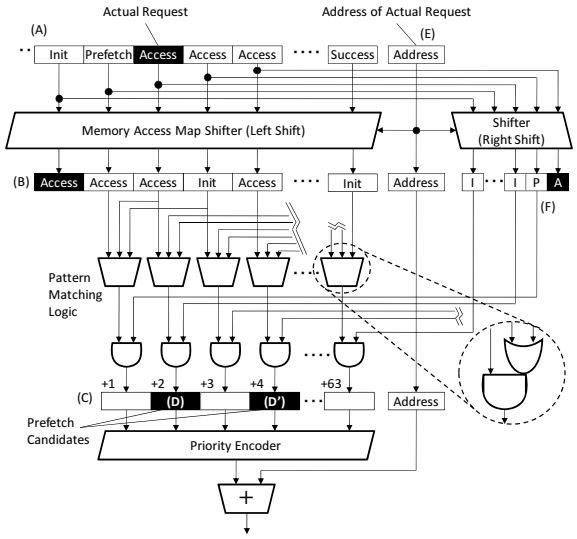

Figure 3: Pattern Matching Logic of the AMPM Prefetch

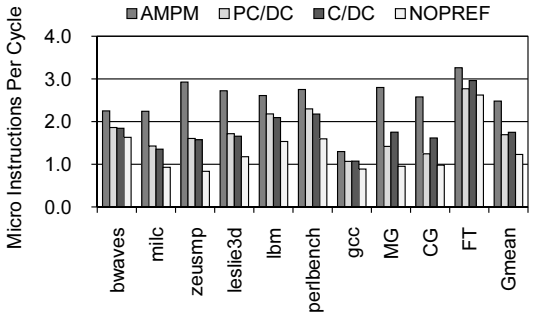

Figure 4: Performance Measurement

The (F) is a shifted map for backward prefetching. Seconds, the parallel pattern matching logic produces candidates in bitmap (C). The N_{th} request is generated by pattern matching of the entry at N, $2N$, and $2N+1$ of the (B). The output is filtered by an N_{th} entry of the (F). This filter checks a target state is "Init" or not and prevents redundant prefetch requests. (D) and (D') are the prefetch candidates. The nearest candidate bit (D) is selected as a prefetch request. Then, the encoded address offset is added to the actual accessed address. Finally, the address is issued as prefetch requests.

3. EVALUATION

We evaluate the AMPM prefetcher in x86-base cycle accurate processor simulator. We compare the AMPM prefetcher with PC/DC [3] and C/DC [2]. Each configuration uses 4KB budget. We use SPEC CPU2006 and NAS Parallel Benchmark for evaluation.

Figure 4 shows the performance obtained by using each benchmark. In figure 4, AMPM, PC/DC, and C/DC stand for each evaluated configuration. The NOPREF means no prefetching. On an average, the AMPM prefetcher improves performance by 42.0%,

Figure 5 shows the frequency of the prefetch requests. The AMPM prefetching increases the number of prefetch requests in all benchmarks. Figure 6 shows the coverage ratio of the prefetch requests. The result shows that the prefetch coverage of AMPM prefetching achieves more than 90 % except in gcc. These results show the aggressiveness of the AMPM prefetching. Especially, the aggressive prefetch improves the performance in FP benchmarks.

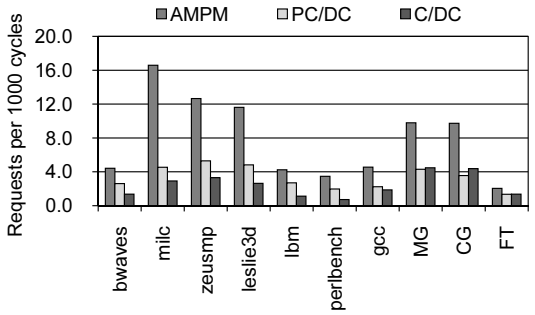

Figure 5: Frequency of Prefetch Requests

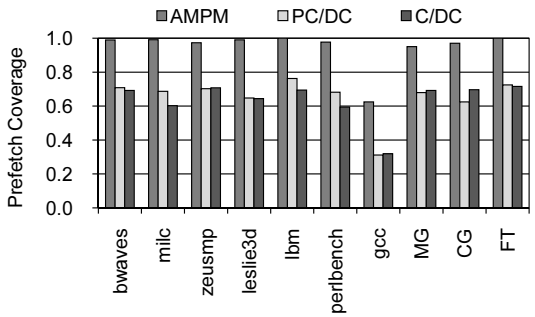

Figure 6: Prefetch Coverage

4. CONCLUSION

In this paper, the aggressive prefetching method — Access Map Pattern Matching — is proposed. The AMPM prefetch detects multiple prefetch candidates from pattern matching of memory access footprints. The AMPM prefetch realizes more aggressive prefetch than conventional methods. The simulation result shows that the AMPM prefetch increases prefetch requests and improves coverage ratio. The AMPM prefetching method [1] won the first data prefetching championship (DPC-1). In the future, we will evaluate detailed property of the AMPM prefetching.

5. REFERENCES

[1] Y. Ishii, M. Inaba, and K. Hiraki. Access map pattern matching prefetch: Optimization friendly method. *The first JILP Data Prefetching Championship*, 2009.

[2] K. J. Nesbit, A. S. Dhodapkar, and J. E. Smith. Ac/dc: An adaptive data cache prefetcher. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 135–145, 2004.

[3] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, pages 96–105, 2004.

[4] S. Palacharla and R. E. Kessler. Evaluating stream buffers as a secondary cache replacement. *SIGARCH Comput. Archit. News*, 22(2):24–33, 1994.