Lazy Prefetching

Aleksandar Milenkovic, Veljko Milutinovic
University of Belgrade, Serbia, Yugoslavia
email: {vm | emilenka}@etf.bg.ac.yu

Abstract

High latency of memory accesses is critical to the performance of shared memory multiprocessors. The technology trends indicate that this gap between processor and memory speeds is likely to increase in the future. To cope with memory latency problem two software-controlled techniques have been investigated: prefetching and remote write. Prefetching is a consumer-initiated technique which moves data close to the processor before they are actually needed by explicit execution prefetch instruction. Remote write, a producer-initiated technique moves data close to the processor estimated to be the next consumer. However these techniques can degrade the performance in the case of misprediction of future needs and/or consumers. The new method called lazy prefetching which combines good properties of prefetching and remote write techniques is presented in this paper. The experimental methodology used for performance analysis is also described.

1. Introduction

The problem of the high memory latency is the most critical performance issue in the cache-coherent shared memory multiprocessors. There are two main techniques to attack memory latency: (a) the first set of techniques attempts to reduce memory latency and, (b) the second set attempts to hide it. The most promising software-controlled techniques for hiding the memory latency are data prefetching and remote write.

Data prefetching is a technique that moves data expected to be used by a given processor closer to the processor before it is actually needed. Software-controlled prefetching implies that prefetch instructions are inserted into application code, either explicitly by the programmer or automatically by the compiler. However, prefetching is inapplicable or insufficient for some communication patterns, especially for producer-consumer patterns.

Proceedings of the IEEE HICSS-98,
Mauna Lani, Hawaii, January 6-10, 1998.

In those cases producer initiated data transfers are a natural style of communication. Under the term of remote writes [2] we imply the producer initiated primitives which move data close to the consumer(s) as soon as the data becomes available, minimizing the latency at the consumer's read.

Lazy prefetching technique combines both prefetching and remote write to solve the problem of misprediction of future needs and/or consumers. This technique also eliminates the problem of early issued prefetching.

Performance analysis is performed using SPLASH-2 application suite. Experimental methodology is based on execution-driven simulator Limes [3] and instrumentation tool Scopa [4].

In the following section lazy prefetching strategy is presented. Also, a possible implementation in cache-coherent shared memory system is described. Section 3 describes experimental methodology used for performance analysis.

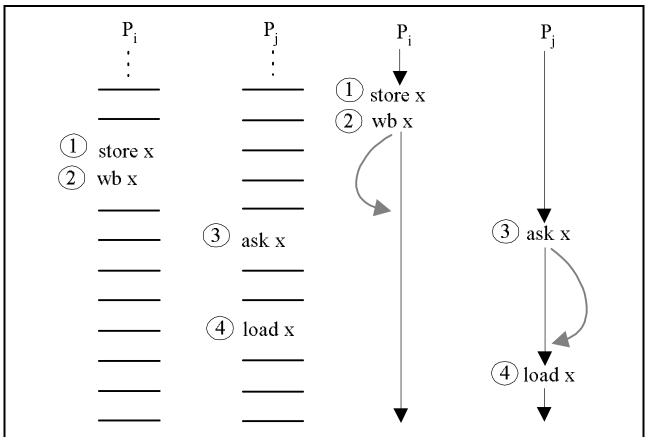
2. Lazy prefetching

The downside of prefetching and remote write techniques may sometimes outperform the benefit, especially in the cases of misprediction of future needs and/or consumers. Also, the problem arises in the case of early issued prefetch instruction when consumer asks for the data before the data are produced. Proposed solution called lazy prefetching (LP) combines good properties of both prefetching and remote write: consumer itself may know its future needs better than producer, and the moment when the data producing is finished, is the best-known to data producer.

In the case of LP consumer anticipates its future needs using **ask** instruction. The algorithm for inserting **ask** instructions can be the same as the algorithm for inserting **prefetch** instruction. **Ask** instruction checks the data cache and if the data is not present in the cache a request to memory is initiated. A memory agent accepts this message and checks the memory. If the memory copy of requested data is valid, the agent sends the requested block to the processor.

This scenario is described in Figure 1. Programing model and execution of the relevant threads are shown in Figure 1a and 1b, respectively.

However, if the memory agent finds that memory does not have a valid copy (some processor is exclusive owner of that block) the request (block address and number of processor) is saved in special table managed by memory agent. After the writer finishes the data producing, it initiates memory update operation using **wb** instruction. Write back cycle forces the memory agent to deliver the updated block to potential consumers according to the information from special table. This scenario is described in Figure 2. Programing model and execution of relevant threads are shown in Figure 2a and 2b, respectively.



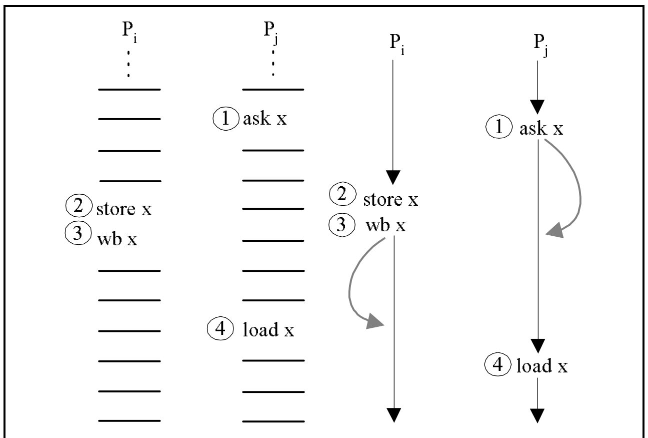
(a) (b)

Figure 1. Prefetching initiated with **ask** instruction. **Description:** Figure (a) shows producer thread (processor Pi) and consumer thread (Pj). Wb and ask instructions are inserted by programmer or compiler. Figure (b) shows relevant execution threads. Grey arcs demonstrate messages on the interconnection network. Since instruction ask finds that requested block is valid in memory it initiates the data transfer to processor Pj.

3. Performance analysis

The performance impact of the proposed solution will be evaluated by comparing five systems: a base cache-coherent shared memory system (Base) and the base system extended with software prefetching (PF), remote writes (RW), both prefetching and remote writes (PF + RW) and the system which combines all prefetching, remote writes and lazy prefetch/remote write (PF + RW + LP).

Performance evaluation will be performed using execution-driven simulator based on Limes and Scopa tools [4, 5].



(a) (b)

Figure 2. Prefetching initiated with **wb** instruction. **Description:** Figure (a) demonstrates scenario where consumer thread (Pj) express the needs using instruction ask before the block x is produced by data producer (thread Pi). Figure (b) shows execution threads. Since instruction ask doesn't find that requested block is valid in memory the memory agent saves the request (block address and processor ID) in special table. Data producer forces memory update using wb instruction after the data producing is finished. Simultaneously memory agent sends the block to interested consumers according to the information from special table.

In our experiments a set of synthetic benchmarks of interest (synthetic analysis) and applications from SPLASH-2 benchmark suite (realistic analysis) are used.

Several simple heuristics based on application behavior will be used for inserting prefetch (pf, ask) and remote write (ws, wb) instructions by hand.

Simulation and implementation analysis is performed for SMP systems based on dance-hall and bus-based shared memory architectures [5].

4. References

[1] Mowry, T., *Tolerating Latency Through Software-controlled Data Prefetching*, Phd thesis, Stanford University, 1994.

[2] Abdel-Shafi, H., Hall, J., Adve. S, Adve, V., "An Evaluation of Fine-Grain Producer-Initiated Communication in Cache-Coherent Multiprocessors," *Proceedings of the 3rd HPCA*, IEEE Computer Society Press, 1997, pp. 204-215.

[3] Magdic, D., "Limes: A Multiprocessor Simulation Environment," *TCCA Newsletter*, March 1997, pp. 68-71.

[4] Marinov, D., Magdic, D., Milenkovic, A., Protic, J., Tartalja I., Milutinovic, V., "Characterization of Parallel Applications for DSM Systems," Technical Report TI-ETF-97-040, School of Electrical Engineering, University of Belgrade, Serbia, Yugoslavia, September 1997.

[5] Milutinovic, V., Milenkovic, A., "Cache Injection/Cofetch Architecture: Initial Performance Analysis," *Proceedings of the 5th MASCOTS*, IEEE Computer Society Press, Los Alamitos, California, January 1997.