Proceedings of the 2002 International Conference on Multimedia and Expo

MULTI-LEVEL MEMORY PREFETCHING FOR MEDIA AND STREAM PROCESSING

Jason Fritts

Dept. of Computer Science, Washington University, St. Louis, MO

ABSTRACT

This paper presents a multi-level memory prefetch hierarchy for media and stream processing applications. Two major bottlenecks in the performance of multimedia and network applications are long memory latencies and limited off-chip processor bandwidth. Aggressive prefetching can be used to mitigate the memory latency problem, but overly aggressive prefetching may overload the limited external processor bandwidth. To accommodate both problems, we propose multi-level memory prefetching. The multi-level organization enables conservative prefetching on-chip and more aggressive prefetching off-chip. The combination provides aggressive prefetching while minimally impacting off-chip bandwidth, enabling more efficient memory performance for media and stream processing. This paper presents preliminary results for multi-level memory prefetching, which show that combining prefetching at the L1 and DRAM memory levels provides the most effective prefetching with minimal extra bandwidth.

1. INTRODUCTION

With the continued success of multimedia computing and networks, media and stream processing are coming to dominate the workloads of many computing systems. In fact, within the near future, media and communications processing are expected to make up over 90% of the workload on most computers. Additionally, media and network processing are integral to many consumer products, including DVD players, cellular phones, and PDAs, so effective computing support is an area of importance for both general-purpose and embedded processors.

A crucial problem in media and network processing is achieving efficient memory performance. Stream-based applications commonly spend 25-50% of their execution time in memory stalls on standard cache-based memory systems. These penalties arise because the memory access patterns for these applications does not correspond well with standard memory systems. Standard cache hierarchies were designed for general-purpose applications, which retain most of their data in on-chip data cache/memory and stream program instructions in from off-chip, as shown in Figure 1.1a and Figure 1.1b, respectively. These applications have good temporal and spatial locality, and are ideally supported by cache-based memory hierarchies. Conversely, media and stream-based applications stream much of their data memory on and off chip and retain their instruction code in on-chip instruction memory/cache [1,2]. Such streaming memory patterns offer poor temporal locality and cause many compulsory misses in standard cache-memory hierarchies. Improved memory performance for media and stream processing requires efficient support for streaming memory.

Streaming memory prefetching, such as provided by stream buffers [3] and stride-prediction tables [4,5], can help reduce these memory penalties. However, like all latency hiding techniques, the effectiveness of prefetching is dependent upon memory latency. As the gap between processor and memory speed continues to increase with advancing VLSI technology, it is increasingly difficult for prefetching to hide memory latency. More aggressive prefetching, i.e. prefetching further ahead in time, is needed to hide longer memory latencies. Unfortunately, the accuracy with which prefetching is able to predict and prefetch future memory accesses diminishes with the distance it attempts to predict into the future. At lower prefetch accuracies, more data must be prefetched in order to return the same amount of useful data. However, this aggressive prefetching is a problem with the increasingly limited memory bandwidths.

We propose multi-level memory prefetching, as shown in Figure 1.2, to enable aggressive prefetching while minimizing the extra bandwidth. The multi-level organization enables conservative prefetching on-chip and more aggressive prefetching off-chip. Multi-level memory prefetching follows the same premise as cache memory hierarchies. However, whereas caches reduce the *post-miss* memory access time by bringing the most critical data closest to the processor and storing increasingly less critical data progressively further from the processor, the prefetch hierarchy attempts to reduce the *pre-miss* memory access time by prefetching data that is *expected to be* more critical progressively closer to the processor and leaving data that is less *likely to be* critical further from the processor. Furthermore, as prefetch data is brought closer to the processor, and so comes closer to its expected use time, it becomes more apparent which data are needed. As a result, prefetching may proceed more conservatively on-chip, minimizing the extra prefetching memory bandwidth. More aggressive prefetching occurs at the outer levels of the hierarchy.

The remainder of this paper is organized as follows. Section 2 provides an overview of memory prefetching. In Section 3, we discuss the details of multi-level memory prefetching. Section 4 examines the results of a number of experiments comparing various multi-level and single-level memory prefetching configurations. Finally, Section 5 summarizes our conclusions.

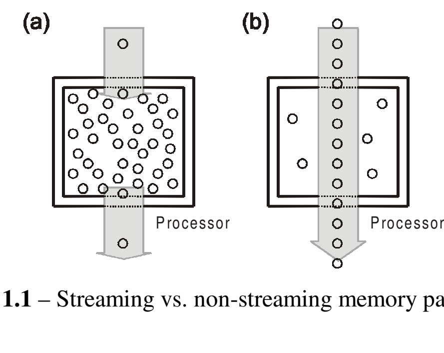

Figure 1.1 – Streaming vs. non-streaming memory patterns [1].

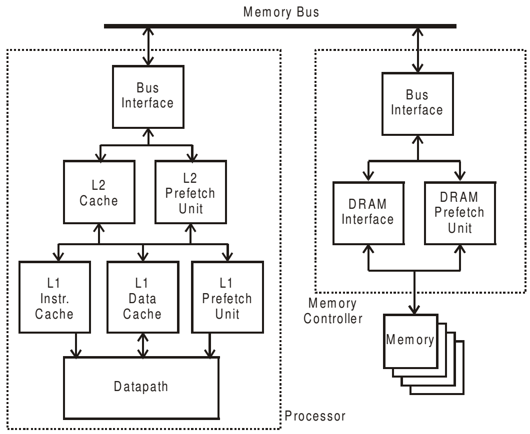

Figure 1.2 – Multi-level Prefetch Hierarchy

2. PREFETCHING TECHNOLOGY

In early cache memory systems, the benefit of prefetching was readily apparent as longer cache lines commonly decreased miss ratios in applications with high spatial locality [6]. This basic phenomenon – that fetching data into the processor prior to when it may needed can improve performance by reducing compulsory misses – lead researchers to the concept of *prefetching*.

Three major classes of prefetching have evolved: software prefetching, hardware prefetching, and hybrid hardware/software prefetching. Software prefetching uses static memory pattern detection to insert prefetch instructions for every datum to be prefetched [7]. Conversely, hardware prefetching methods are entirely dynamic, using hardware prediction in both detecting memory patterns and issuing prefetch requests [3,4,5,8]. Hybrid hardware/software prefetching is a relatively new alternative, which combines static memory detection with hardware prefetch issue units [9,10], requiring only a single prefetch instruction per stream. This instruction indicates when to begin prefetching, what elements to prefetch, how many elements to prefetch, and how far in advance to prefetch. When the instruction executes, this information is placed in a hardware table that controls issuing each individual prefetch request.

In this paper, we focus primarily on hardware prefetching methods, since software and hybrid prefetching methods rely on static program analysis by a compiler for determining memory access patterns and inserting prefetch instructions. Like data dependence analysis, static memory pattern analysis is a difficult problem, so compilers often miss opportunities where prefetching would be beneficial. In comparing hardware prefetching with hybrid hardware/software prefetching, Pimentel et al. [10] found that hybrid prefetching was more efficient in those code sections that were instrumented with stream prefetch instructions, but hardware prefetching performed best overall because it enabled prefetching throughout the entire application.

2.1. Stream Buffer

Among hardware prefetching methods, there are three primary alternatives: stream buffers, stride-prediction tables, and stream caches. The *stream buffer*, proposed by Jouppi [3], is a small, fully associative cache residing on the miss path of the L1 cache. Jouppi's method is an extension to Smith's *one-block lookahead* method for prefetching the next successive line on a cache miss [6], but instead of prefetching just one line from memory, the stream buffer prefetches the next *k* sequential lines from memory on a cache miss (where *k* is the number of entries in the stream buffer). Subsequent L1 cache misses check if the desired data resides in the stream buffer. If so, the miss data is moved from the stream buffer to the L1 cache. As opposed to Smith, Jouppi stores prefetch data in the stream buffer until needed, preventing prefetches from polluting the L1 cache. Jouppi extended this idea to multi-way stream buffers for supporting multiple streams.

2.2. Stride-Prediction Table

The second major hardware prefetch method is stride-based prefetching. Baer and Chen [4] and Fu and Patel [5] developed stride-based prefetching, nearly simultaneously, to overcome the limitation of stream buffers in only prefetching successive cache lines. Oftentimes, algorithms access memory in a strided fashion (i.e. consecutive accesses are separated by a constant distance, known as the *stride*) where the stride may be quite large. In such cases, the program may not need every cache line, so a *stride-prediction table (SPT)*, which resides in parallel with the L1 cache, monitors memory accesses and determines whether each static instruction accesses streaming or non-streaming memory.

Each entry in the SPT table is tagged with its instruction address, and contains information about its most recent memory accesses, including the last data access, the stride, and state bits. When a load instruction performs its first data access, a new entry is allocated and the 'last memory address' field is updated. When that instruction performs its next memory access, the table takes the differences of the addresses to generate the stride (i.e. *stride = current_addr – last_addr*) and store it in the table. Using the stride, it issues prefetch requests for the next predicted memory address(es) (i.e. *next_addr = current_addr + stride*). Prefetched data elements are placed directly into the cache.

2.3. Stream Cache

Zucker et al. extended the idea of the stride-based prefetching to create the *stream cache* [8]. They found that stride prefetching performance is often significantly lower in smaller caches because the prefetch data pollutes the L1 cache. To rectify this, they added a "stream cache", separate from the L1 cache, which uses a SPT table to determine what data to prefetch, but stores the data in the stream cache instead of the L1 cache. Consequently, the stream cache effectively enables separation of the L1 memory level into two smaller, faster memory structures. The stream cache was very beneficial in enabling effective prefetching when using a small L1 cache.

Aside from stream buffers, stride-based prefetching, and stream caches, most other hardware prefetching advances have essentially been enhancements to these fundamental methods. The majority of the enhancements fall into the category of more advanced address prediction, which enables both: a) advanced pattern recognition for prefetching of both stride-based and irregular memory access patterns, and b) more accurate pattern recognition for reducing the extra bandwidth from unused prefetch data. The various enhancements for improved address prediction include Palacharla and Kessler's allocation filter, non-unit stride detection, and minimum-delta techniques [11], Chen and Baer's correlation method [4], and Farkas et al.'s incremental prefetching and per-load stride predictor [12]. The other major category of enhancements allow for increased lookahead distances, such as (i.e. distance to prefetch ahead of the stream): Baer and Chen's lookahead PC [9].

3. MULTI-LEVEL PREFTCH HIERARCHY

Memory latency has long been the most critical aspect of memory performance in processor design. However, with the increasing processor-memory gap, off-chip bandwidth is also becoming a concern. Burger et al. [13] studied the impact of latency and bandwidth on performance and concluded that the growing processor-memory gap, and the use of aggressive latency-hiding, will significantly limit off-chip bandwidth in future systems, leading to potentially serious degradations in system performance. They argue that smarter memory systems are needed to offset the increasing processor-memory gap.

We propose multi-level memory prefetching (shown earlier in Figure 1.2) as an improved memory system design for future media and stream processing systems. Multi-level memory prefetching balances the competing factors of longer memory latencies and more limited bandwidth by enabling aggressive prefetching with minimal extra bandwidth consumption.

Prior studies in memory prefetching have examined methods that reduce the extra bandwidth requirements from prefetching [4,9,11]. However, they also noticeably degrade prefetching performance. Conversely, multi-level memory prefetching is designed to enable aggressive prefetching with minimal extra bandwidth by prefetching more conservatively closer to the processor and more aggressively further away from the processor. Previous memory prefetching studies primarily examined prefetching at only one level of the memory hierarchy (usually the L1 level). However, streaming data must traverse all levels of the memory hierarchy, so prefetching should occur at all levels of the memory hierarchy.

In essence, multi-level memory prefetching follows the same premise as multi-level cache memory hierarchies. The smallest, fastest memory structures reside closest to the processor, while memory structures further from the processor become progressively larger and slower. Streaming data lends itself very nicely to this type of organization due to the inherent nature of prefetch heuristics, which are most accurate at predicting data needed in the near future, and progressively less accurate at predicting data needed further into the future, as illustrated in Figure 3.1. The prefetch units closest to the processor are also closest in time to the point at which the prefetched data will likely be needed. Consequently, their predictions for which data should be prefetched are more accurate, so these on-chip memory structures conservatively prefetch less data. As their prefetch predictions are more accurate, the number of extraneous bus cycles consumed by mispredicted prefetches is minimized.

For prefetching units residing further from the processor, their predictions of which data to prefetch are more speculative (i.e. further ahead in time from when the data may be needed), so they must prefetch more aggressively in order to obtain the desired data. And while the accuracy of the prefetch data in these outer memory structures is much lower, the prefetch hierarchy acts to gradually filter out mispredicted prefetch data while bringing the desired data closer to the processor. Consequently, the prefetch data progressively becomes more accurate as it moves closer both to the processor and to the time at which it is needed.

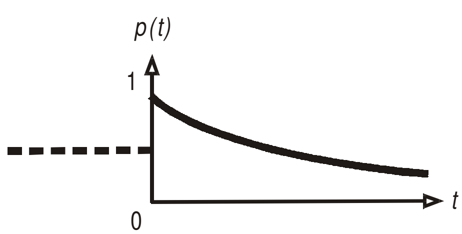

Figure 3.1 – Prefetch accuracy vs. prefetch lookahead distance.

To our knowledge, the only known research study that has looked at multiple levels of prefetching is the Impulse project [14]. They combined prefetching at the L1 cache level via Smith's one block lookahead method [6] with prefetching using an off-chip SRAM near the memory controller. They presented only limited results, but these indicate that prefetching at multiple levels enables performs better than either level alone.

4. RESULTS

This section presents results for the initial experiments that we have performed in evaluating the effectiveness of multi-level memory prefetching. These results give a preliminary indicator of the performance of a number of different configurations of multi-level memory prefetching, in terms of both speedup and the degree of extra bandwidth consumption.

We performed experiments using various multimedia benchmarks, some of which derive from the MediaBench benchmark.

- *jpg2Kdec* – a JPEG-2000 image decoder
- *mpeg4dec* – an MPEG-4 video decoder
- *pegwitdec* – a simple encryption decoder
- *unepic* – a wavelet-based image decoder
- *vorbisenc* – an MP3-like audio encoder

All of these are reasonably memory intensive benchmarks.

We performed our experiments under the Impact compilation/simulation environment. The simulator was extended to model prefetching using stream buffers at each of the memory levels, including the L1, L2, and DRAM memory controller levels. The base architecture used for the experiments models a 1 GHz out-of-order superscalar processor with a non-blocking 32 KB direct-mapped L1 data cache with 64-byte lines and a 15 cycle miss latency, a 256 KB 4-way set-associative L2 cache with 64-byte lines and a 100 cycle miss latency, and a 166MHz processor-memory bus that supports split transactions. The base architecture model does not employ any prefetching.

In evaluating the multi-level memory prefetching models, we tested three single-level prefetching models, each of which employs prefetching at only one of the memory levels (L1, L2, or MC), and three multi-level memory prefetching models, each of which combine L1 prefetching with prefetching at the L2 and/or DRAM memory controller levels (L1+L2, L1+MC, and L1+L2+MC). For the prefetching at the L1 memory level, we use an 8-way stream buffer with 5 entries per stream, whereas prefetching at the L2 and MC memory levels uses an 8-way stream buffer with only 3 entries per stream.

Finally, regarding the placement of the prefetch units, the L1 and L2 prefetch units are placed in the miss-path of the L1 and L2 caches, respectively. For prefetching at the memory controller though, the 8-way stream buffer is built using a 10ns SRAM which is positioned in front of the DRAM (i.e the DRAM is in the miss-path of the SRAM). So, if the desired data for the L2 miss resides in the SRAM, the miss penalty is only 50 cycles, as opposed to a 100 cycle miss penalty from the 60ns DRAM memory. Placing the SRAM in front of the DRAM provides the dual benefit of significantly reducing the average L2 miss penalty, while enabling prefetching between the SRAM and DRAM to occur over a private bus between the SRAM and

DRAM. Consequently, prefetch requests by the SRAM prefetch unit do not consume any external processor bus bandwidth.

The IPC performance results are shown in Figure 4.1. Because these are fairly intensive benchmarks, even the single-level memory prefetching methods enable significant speedups, with average speedups of 45%, 34%, and 24% for the L1, L2, and MC prefetching models. L1 achieves the best IPC of the three since it is has the most accurate prefetching. Among the multi-level memory prefetching methods, the L1+MC and L1+L2+MC models perform the best, with average speedups of 89% and 84%, respectively. The L1+L2+MC method typically performed better than the L1+MC, but the L1+L2+MC method performed worse on *jpg2Kdec* because the extra bandwidth of the L1+L2+MC method saturates the processor-memory bus, thereby increasing execution time.

In measuring the extra bandwidth consumption, we compared the bus utilization of each memory model, normalized according to speedup, with the bus utilization in the base architecture. The difference defined the excess bandwidth consumption. For single-level prefetching, the excess bandwidth was about 50% for the L1 model and nearly double that for the L2 model (it is obviously 0% for the MC model). For multi-level prefetching, the L1+L2 and L1+L2+MC models had excess bandwidths of roughly 100%, whereas the excess bandwidth for the L1+MC method was only about 50% (comparable to L1 prefetching). Consequently, these preliminary results indicate that the L1+MC multi-level prefetching model provides the best speedup vs. bandwidth-efficiency performance. And since its median speedup is nearly as good as that of the L1+L2+MC model, these initial experiments distinguish it as the best all-around option.

5. CONCLUSIONS

We present multi-level memory prefetching for media and stream processing as an effective prefetching method for enabling aggressive prefetching while minimizing the amount of extra prefetching bandwidth. A multi-level prefetch hierarchy effectively handles both aspects of the growing processor-memory gap, conservatively prefetching on-chip while aggressively prefetching off-chip, providing both latency hiding and bandwidth minimization. In essence, the proposed method creates a prefetch hierarchy similar to a cache memory hierarchy.

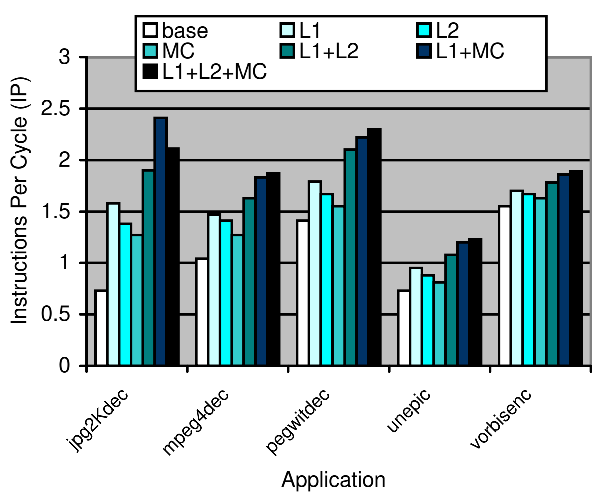

Figure 4.1 – IPC results for multi-level memory prefetching.

This paper provides a preliminary investigation into the effectiveness of multi-level memory prefetching. For more definitive conclusions, further evaluations are needed to comprehensively evaluate the various aspects of multi-level memory prefetching, including experimenting with different prefetching methods and bandwidth-efficiency enhancements, examining different degrees of prefetching lookahead, exploring variations with latency and bandwidth, and so on. However, these initial results indicate significant potential benefits. By prefetching at the memory controller as well as the L1 level, speed improved by 30% over L1 prefetching alone, with negligible change in the bandwidth requirements.

[1] John Stokes, "The Playstation2 vs. the PC: a System-level Comparison of Two 3D Platforms," *Ars Technica*, April 2000.

[2] Jason Fritts and Wayne Wolf, "Instruction fetch characteristics of media processing," *SPIE Photonics West, Media Processors 2002*, San Jose, CA, Jan. 2002.

[3] Norman P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associate cache and prefetch buffers," *Proc. of the 17th Intl. Symp. on Comp. Arch. (ISCA-17)*, pp. 62-73, May 1990.

[4] Tien-Fu Chen and Jean-Loup Baer, "Effective Hardware-Based Data Prefetching for High-Performance Processors," *IEEE Transactions on Computers*, vol. 44, no. 5, pp. 609-623, May 1995.

[5] J. Fu, J. Patel, and B. Janssens, "Stride directed prefetching in scalar processors," *Proc. of the 25th Intl. Symp. on Comp. Arch. (ISCA-25)*, pp. 102-110, Dec. 1992.

[6] Alan Jay Smith, "Cache memories," *ACM Computing Surveys*, vol. 14, no. 3, pp. 473-530, Sept. 1982.

[7] T.C. Mowry, M.S. Lam, and A. Gupta, "Design and evaluation of a compiler algorithm for prefetching," *Proc. of the 5th Intl. Conf. on Arch. Support for Prog. Lang. and Oper. Sys. (ASPLOS-5)*, pp. 62-73, Oct. 1992.

[8] D.F. Zucker, M.J. Flynn, and R.B. Lee, "A Comparison of Hardware Prefetching Techniques for Multimedia Benchmarks," *Proc. of the Intl. Conf. on Multimedia Comp. and Sys.*, June 1996.

[9] Tien-Fu Chen and Jean-Loup Baer, "A Performance Study of Software and Hardware Data Prefetching Schemes," *Proc. of the 21st Intl. Symp. on Comp. Arch. (ISCA-21)*, pp. 223-232, April 1994.

[10] A.D. Pimentel, L.O. Hertzberger, P. Struik, and P. van der Wolf, "Hardware versus Hybrid Data Prefetching in Multimedia Processors: A Case Study," *Proc. of the Intl. Perf., Comp., and Comm. Conf.*, pp. 525-531, Feb. 2000.

[11] S. Palacharla and R. E. Kessler, "Evaluating stream buffers as secondary cache replacement," *Proc. of the 21st Intl. Symp. on Comp. Arch. (ISCA-21)*, pp. 24-33, April 1994.

[12] K. I. Farkas, P. Chow, N. Jouppi, and Z. Vranesic, "Memory-system design considerations for dynamically-scheduled processors," *Proc. of the 24th Intl. Symp. on Comp. Arch. (ISCA-24)*, June 1997.

[13] D. Burger, J.R. Goodman, and A. Kagi, "Limited bandwidth to affect processor design," *IEEE Micro*, 17(6):55-62, 1997.

[14] John B. Carter, Wilson C. Hsieh, Leigh B. Stoller, Mark Swanson, Lixin Zhang, and Sally A. McKee, "Impluse: Memory System Support for Scientific Applications," *Journal of Scientific Programming*, 7(3):195—209, 1999.