Analysis and Improvement of Cache performance for Multimedia Applications

Bo Zhang, Guohui Wang, Jesus Jason Sedano*
Dept. of Computer Science, *Dept. of Electrical &Computer Engineering,
Rice University, Houston, Tx
{bozhang, ghwang, jjsedano} @ rice.edu

Abstract

Multimedia applications are among the quickly growing areas in computer systems and becoming more and more important in people's daily life. They are essentially memory-intensive and time-critical applications. Previous results and our analysis both confirm that media applications have some special properties like predictable data access patterns. These properties make them relatively easy to be optimized on general-purpose processors using data prefetching techniques.

In this paper, we first systematically study the cache performance of multimedia applications to characterize their behaviors so that we can have deeper insights into media applications characteristics. Then based on our understanding, we use prefetching schemes, under different memory access models, to improve cache performance. Our simulation results show that prefetching is an effective way to improve cache performance in terms of miss rate.

1. Introduction

Multimedia applications are fast becoming one of the dominant workloads for modern computer systems [3]. The dependency of Internet use and the demand by the majority of users for graphically based programs will continue to trend upward. It can be seen that even the simplest tasks now use a complex graphical interface to make it more user friendly and will many times contain many bells and whistles usually being videos, sounds and fancy graphical pictures.

The real-time constraint of media applications demands a high level of performance density. To meet the high performance requirement, special purpose media processors, such as Imagine [4], were designed to run media processing efficiently. The media processor is expensive and less flexible, which can only be used effectively for special purpose systems. However, with the development of the Internet and the graphical interface, far more media applications are running on general purpose processors. General purpose processors offer increased flexibility but achieve performance levels two or three orders of magnitude worse than special purpose processors [4].

A common belief for this huge performance gap is that conventional general purpose architectures are poorly matched to the specific properties of media applications including little data reuse, high data parallelism, large data sets and computationally intensive processing. The memory systems of general-purpose architectures depend on caches optimized for reducing latency and data reuse, and don't efficiently exploit the available data parallelism in media applications [4]. So, it is interesting to study the cache performance when given media applications and in turn, try to improve the performance of media applications on general purpose processors.

In this project, we give a thorough analysis of cache performance for media applications on general purpose processors. We try to answer the following questions: how is media applications' cache performance? How does cache configurations impact the cache performance of media application? What is the memory access pattern for media application? Based on our analysis, we will be able to use a simple prefetch scheme to improve cache performance of media applications. The remainder of this paper is organized as follows: section 2 introduces our methodology to study cache performance of media applications. Section 3 discusses the analytical results of cache performance. Section 4 introduces our simulation of a desired prefetch scheme to improve cache performance of media applications. Section 5 give a discussion of simulation results for prefetching. Section 6 summarizes the paper.

2. Methodology

2.1 Simulation environment

Our study of cache performance is based on SimpleScalar simulator [1]. We use SimpleScalar Toolset version 3.0, PISA architecture. The simulation is run on SPARC system, with Solaris OS.

2.2 Benchmarks

2.2.1 MediaBench

MediaBench [2] consists of complete applications coded in high level languages. It includes core algorithms for most widely used multimedia applications. All the applications are publicly available and widely used on general purpose processors. We choose a subset from the MediaBench, including JPEG, MPEG, ADPCM, MESA, GHOSTSCRIPT, and EPIC. We choose these applications because they cover different types of media application, such as an image, video, audio, 3D graph, document and compression.

- JPEG: JPEG is a standardized image compression mechanism for both full-color and gray-scale images. It has the ability to compress an image with little or no noticeable degradation in image quality. We use JPEG decompression, the input file is a 5KB jpg file and the output .ppm file is 100KB.

- MPEG2: MPEG2 is a standard for digital video transmission. We use mpeg2 decoder in our simulation with a 34KB input file.
- EPIC: EPIC is an experimental image compression utility. The compression algorithms are based on a bi-orthogonal, critically sampled dyadic wavelet decomposition and a combined run-length/Huffman entropy encoder. We use a 65KB pgm file for compression with EPIC program.
- ADPCM: ADPCM stands for Adaptive Differential Pulse Code Modulation. It is a family of speech compression and decompression algorithms. A common implementation takes 16bit linear PCM samples and converts them to 4-bit samples. We use the program "rawdaudio" with a 72KB input adpcm file.
- GhostScript: An interpreter for the PostScript language. We use the program "gs" with a 78KB input file tiger.ps
- Mesa: Mesa is a clone of OpenGL, a commonly used 3-D graphics library.

We use a demo program “mipmap” in our simulation, which demonstrate using mipmaps for texture maps.

2.2.2 SPEC2000 Applications

We choose the SPEC2000 [5] benchmark to represent traditional programs. It is not easy to collect all the binary and input files of SPEC2000 programs. In our simulation, we use Compress, Go and GCC.

3. Cache Performance of media applications

It is important to quantitatively characterize the cache behavior of multimedia applications in order to provide insights for people’s future research on this area. In this section, the cache performance and memory access patterns of a set of representative multimedia benchmarks are analyzed first. We present a clear picture of the memory reference characteristics of multimedia applications with regards to their cache performances and memory access patterns. Cache performance of different multimedia applications under different parameters settings are studied in order to gain a better understanding of their memory access characteristics. For comparison purpose, three benchmarks from SPEC2000 are also studied to show the differences between multimedia and general-purpose applications.

3.1 Simulation Setting

Three factors are highly related to cache performance: cache size, associativity, and line size. Therefore, we conducted four sets of simulations to study how these three factors influence L-1 data cache performance. All four sets of simulations use LRU as their cache replacement algorithms. The configurations of the four sets of simulations are:

1) Varying cache size from 4k to 256k while keeping line size 32B constant and using directly mapping policy.
2) Varying cache size from 4k to 256k while keeping line size 32B constant and using 2-way mapping.
3) Varying associativity from 1 to 8 while keeping line size 32B and cache size 16KB.
4) Varying line size from 8 bytes to 64 bytes while keeping cache size 16 KB and 2-way mapping.

3.2 Simulation results and analysis

In this section, results for each above simulation are presented and detailed analysis is also provided. . In this section we only use L-1 data cache miss rate to represent cache performance.

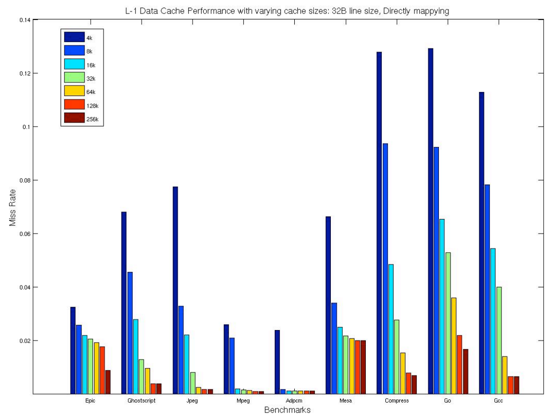


Figure 1. L1 D-cache performance (one-way, line size: 32Bytes)

Figure 1 compares the L-1 data cache miss rate of different media benchmarks and SPEC benchmarks by varying cache sizes from 4k to 256K, as we describe above, we keep other cache-related parameters constant, that is to say, we use 32B line size, directly mapping and LRU cache replacements algorithms. A couple of observations from Figure 1 are:

1) Miss rate decreases with the increase of cache size

2) Generally speaking, multimedia applications have better cache performance compared to SPEC2000 benchmarks, which is different from our expectations.
3) Different media applications show very different cache performance. The miss rates of ADPCM and MPEG are very low (miss rate ~ 0.1% at 16KB cache), while Mesa, EPIC and GhostScript perform much worse.
4) Within 6 media benchmarks, EPIC and MESA are different from others because their miss rate seems not very sensitive to cache size. That is to say, by only increasing cache size, it is hard to improve their miss rate.

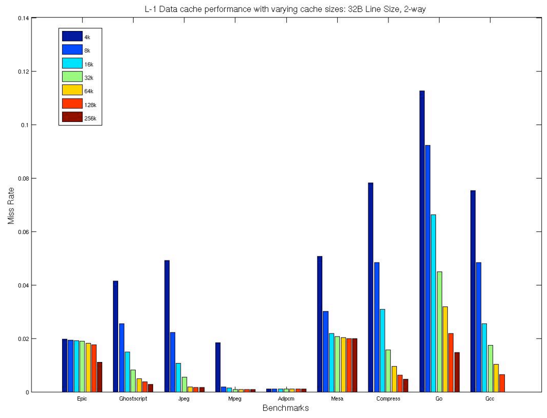


Figure 2. L1 D-cache performance (2-way, line_size: 32Bytes)

Figure 2 shows similar results using miss rate comparison results generated by simulation set 2. Same observations can be seen in Figure 2 too. The miss rates in Figure 2 are relatively lower than what was found in Figure 1, which is not surprising.

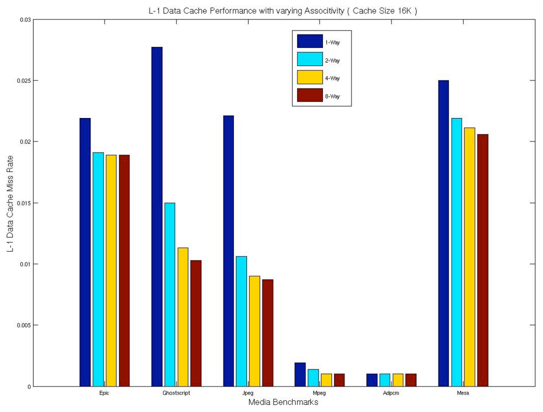


Figure 3. L1 D-cache performance (varying associativity)

Now we fix cache size and continue to study how other factors like associativity and line size can affect cache performances significantly so that we can get some deeper insights on how to improve the cache performance of multimedia applications. Figure 3 shows the results for simulation set 3. Big miss rate drop is observed when associativity increase from 1 to 2. But after that even we continue to increase associativity, we can't gain much benefit any more, which implies that associativity is not an effective factor to influence cache performance.

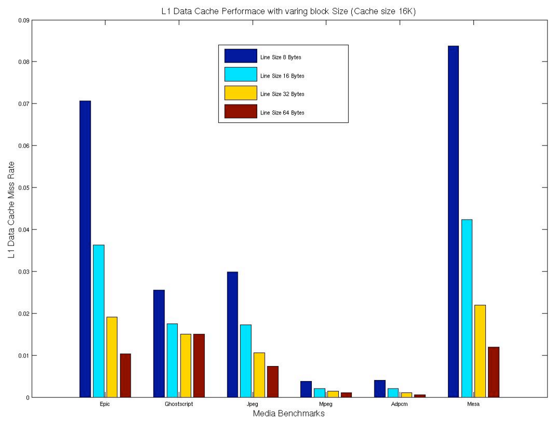


Figure 4. L1 D-cache performance varying line size

Then we switch to line size. Figure 4 presents the results of simulation set 4. As we can see, miss rates of all media benchmarks except GhostScript drop almost by half when we double line size. Of course, larger line size means more memory bandwidth usage. The fact that increasing line size can improve miss rate significantly implies that media applications have good locality and sequential memory access property. GhostScript 's miss rate doesn't decrease a lot with increase of line size, so GhostScript may not have strong sequential property.

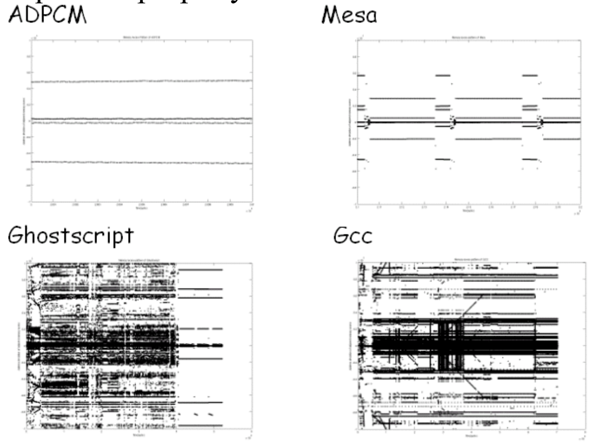


Figure5 Memory Access Patterns

In order to verify our conjecture, we collected simulation traces of different benchmarks and study their memory access patterns. Basically we plotted the address deviation of adjacent memory access. Figure 5 shows memory access patterns of different benchmarks under the cache configuration: 16KB, 32B line size and 2-way associativity. X axis is time (in cycle), Y axis is the address deviation. It can be observed that multimedia applications have more regular memory access pattern than spec2000 benchmarks. As we can see, Ghostscript's memory access pattern is pretty random. That is why we can't improve its cache performance by simply increasing cache sizes.

In summary, media applications have lower miss rate than traditional programs. But the cache performance for different media applications can be very different. ADPCM and MPEG have very low miss rate, while the miss rate of EPIC and Mesa is much higher. All the media application except GhostScript have regular memory access pattern. And we believe that this is due to the following reasons [3]:

1) Most multimedia applications apply block-partitioning algorithms to the input data and work on small blocks of data that easily fit in the cache.
2) Within these blocks, there is significant data reuse as well as spatial locality.
3) Third, a large number of references generated by multimedia applications are to their internal data structures, which are relatively small and can easily fit in reasonably sized caches.

These results inspire us to believe that prefetching should be a very promising technique to improve performance of media applications, such as Mesa and EPIC, due to their regular memory access patterns

So we hypothesize that by implementing appropriate prefetching schemes in general purpose processors, we can better leverage the properties of media applications to provide better cache performance. In next section we will introduce our experience of using prefetch to improve cache performance for media application.

4. Prefetch Simulation

4.1 One block look-ahead prefetching

We first try simple one-block-look-ahead prefetch[6] on the general purpose processor. One-block-look-ahead means that, once there is a cache miss, we prefetch the next block into cache. One-block-look-ahead prefetch seems to be very simple, but our intuition is that, this scheme could be a good choice for media applications. We know that, media applications have relatively regular memory access stride, but not constant stride. It is very hard to accurately predict the stride dynamically during running time. On the other hand, because of the streaming property of media applications, if one memory block is accessed, it is highly possible that the next block is going to be accessed in future.

4.2 Simulation assumptions

We implemented the one block look-ahead in SimpleScalar Version 3.0. In our simulation, different assumptions of memory access model are used:

(1) Pipeline memory model: the pipeline memory model is adopted by SimpleScalar simulation. In this model, processor can issue memory requests in a pipelined manner, and fetch multiple data blocks in parallel. The pipeline memory model implies unlimited memory bandwidth. In this model, the benefit of prefetch is actually overrated.

(2) Sequential memory model [7]: The pipeline memory model is obviously impractical in real system, so we implemented a sequential memory model in SimpleScalar. This model is the most restrictive one since it implies that no memory requests can be initiated until previous request is completed. This means that, if we place a memory request while another memory request is still outstanding, we have to wait until data of the outstanding request arrives from memory before the new request starts to be served. Another issue needs to be considered in this model is: what is priority of data fetching after cache misses? In our simulation, we use two policies: The first one is FIFO policy, which means we treat missed block and prefetching block at the same priority, first come first served. Second, missed block first policy, which means if there is a missed block request, we first serve this request. Only when there is no missed request and the memory bus is free, we can process the prefetching blocks in the request queue.

4.3 Simulation settings

We performed three sets of simulations. In the first set, we measured the cache performance with and without prefetching under a pipeline memory model. Second, we ran simulations under a sequential memory model and a FIFO policy. Thirdly, we use a sequential memory model and missed block first policy. In our experiments, the simulation settings we used are:

- Cahce line size: 32 Bytes
- Associativity: 2 way
- Cache size: 8KB, 16KB, 32KB
- L1 hit latency = 1 cycle
- L2 hit latency = 6 cycle
- L2 cache: 256KB 4-way set associative cache with 64 bytes line
- Memory access latency = 26 cycle
- Cache replacement policy: LRU

4.4 Simulation metrics

(1) L1 cache miss rate

This metric is used to reflect the success rate of prefetching schemes on the reducing cache misses. In our results, an access is considered a hit even if the data request for this access has been issued but has not yet completed.

(2) Cycles Per Instruction (CPI)

CPI represents the average number of cycles every instruction takes when running. This metrics is used to evaluate how the running performance of benchmarks is impacted by prefetching.

5. Results and Discussion

5.1 Results for pipelined memory model

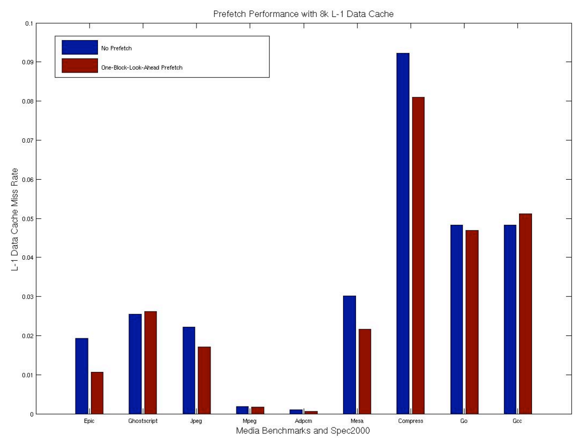
Figure 6 shows the results under the pipeline memory model, in which Figure 6(a), (c), (e) show the miss rates of different benchmark programs with different cache sizes. We can see that, for all the media applications, the prefetch scheme can reduce cache miss rate over no prefetch scheme. For an 8KB cache, the miss rate of Mesa is decreased from 3.0% to 2.2%, Epic's is decreased from 1.93% to 1.07%. The only exception is the GhostScript program for a cache size of 8KB. We believe the reason is that, the memory access pattern for Ghostscript is far messier compared to other media applications, when the cache size is small, the data pollution leads to more cache misses. For SPEC2000 benchmarks, because the memory access pattern is not as regular as media applications, the result is not so good. But we can still see that, the miss rates for Compress and Go are reduced, whereas the miss rate of GCC at 8KB cache increased.

Figure 6(b), (d), (f) show the CPI results for different benchmark programs. The CPIs of all media applications except GhostScript are reduced slightly (~1%). The impact of pre-fetch scheme on CPI is not very obvious. The reason is, first, CPI can be impacted by many factors, such as pipeline and memory latency, cache performance is only one of them. Second, the miss rate of media applications is already very low, so there is no much room to improve the overall performance by reducing cache misses.

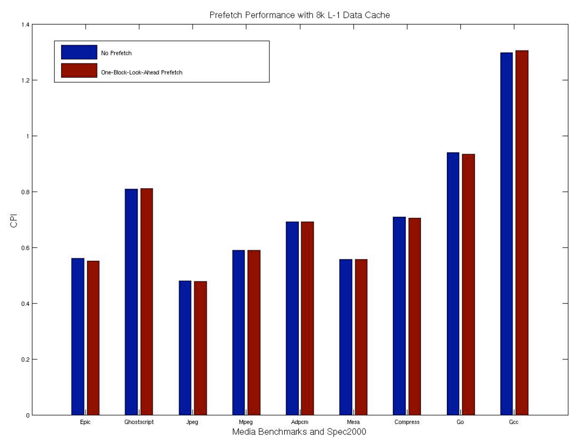
5.2 Results for sequential memory model

The results of the sequential memory model are showed in Figure 7. Let's first see the results for the FIFO priority policy. In Figure 7(a), (c), (e), we can see that, the miss rate of media applications can be reduced. However, in terms of CPI results in Figure 7(b), (d), (f), we can see that, the CPI for Mesa and GhostScript increased a little bit over no prefetch scheme. The reason is because we give the same priority to missed block request and pre-fetching. The data fetching after cache misses can be delayed by prefetching, and the pipeline is unnecessarily stalled in this case.

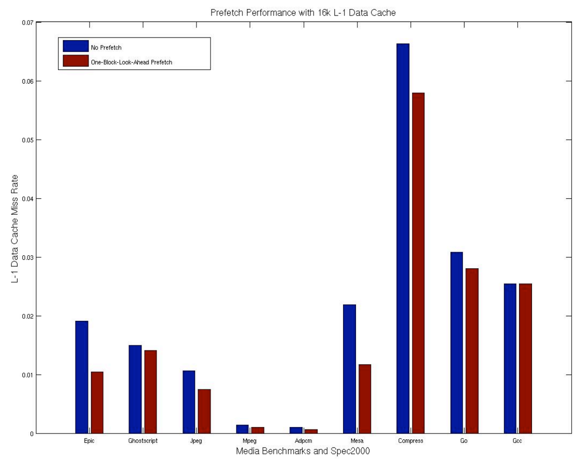
In missed block first policy, we give miss block a higher priority. The pre-fetch requests can be served only when there is no missed block request. From the results we can see that, the reduction of miss rates is not as good as FIFO policy. The reason is if we give pre-fetching a low priority, some pre-fetch requests can be invalidated because the data has already been missed and fetched before the pre-fetching is really served. However, with the missed block first policy, the CPI for every benchmark becomes better than FIFO policy. This is because unnecessary pipeline stall caused by pre-fetch is eliminated.



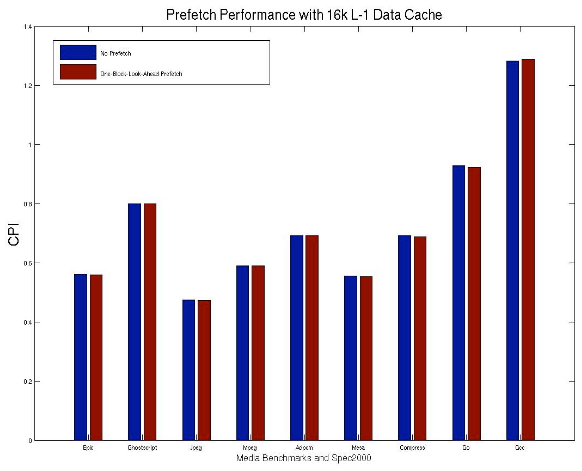
(a)



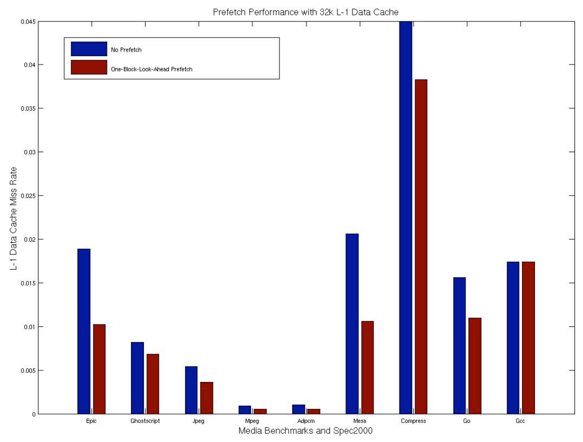
(b)



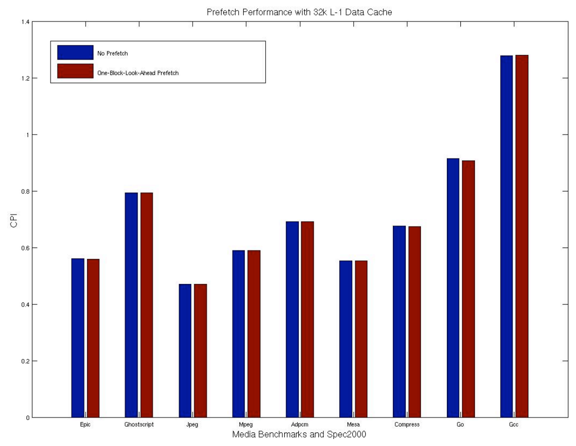
(c)



(d)

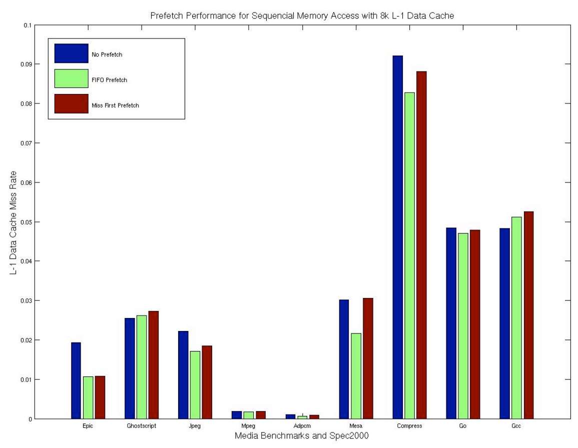


(e)

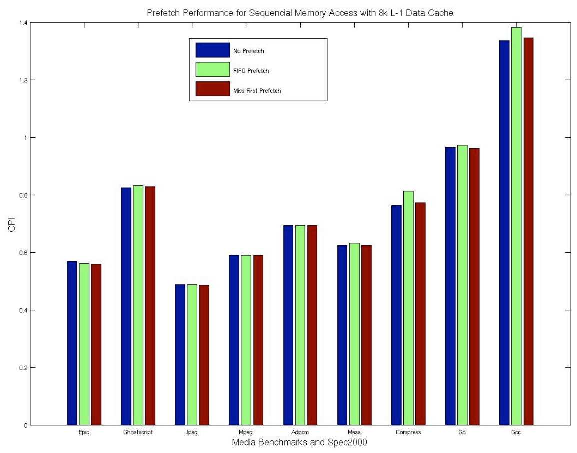


(f)

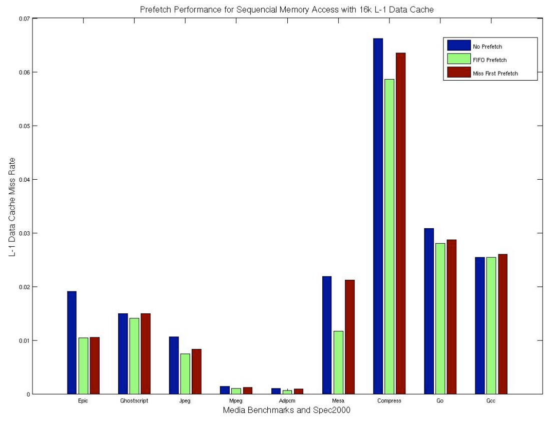
Figure 6. Prefetching results for pipelined memory: (a) Miss rate for 8k cache, (b) CPI result for 8k cache, (c) Miss rate for 16k cache, (d) CPI result for 16k cache, (e) Miss rate for 32k cache, (f) CPI result for 32k cache



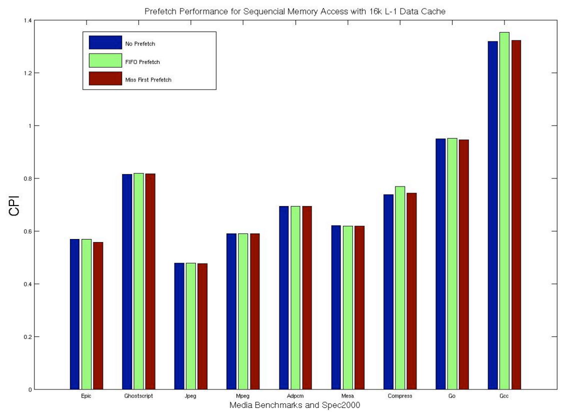
(a)



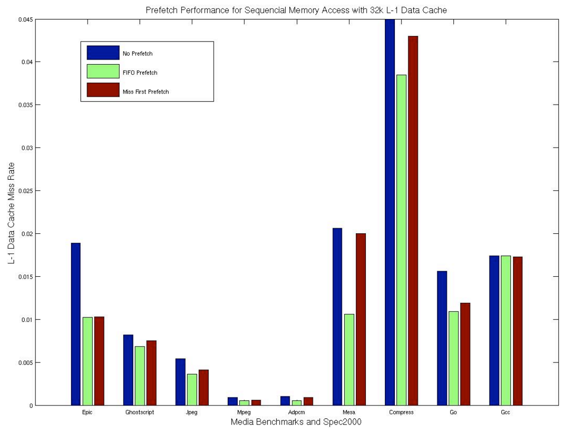
(b)



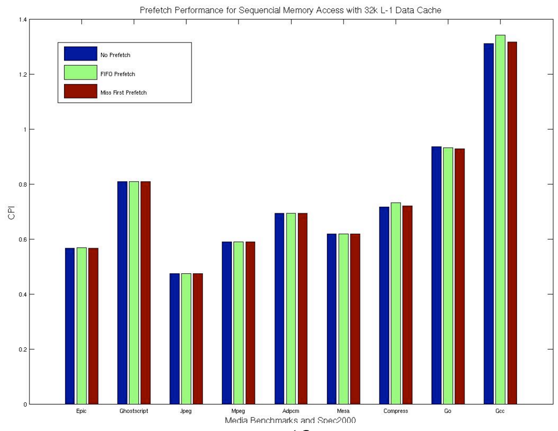
(c)



(d)



(e)



(f)

Figure 6. Prefetching results for sequential memory: (a) Miss rate for 8k cache, (b) CPI result for 8k cache, (c) Miss rate for 16k cache, (d) CPI result for 16k cache, (e) Miss rate for 32k cache, (f) CPI result for 32k cache

6. Conclusion and future work

Multimedia has become one of dominant applications on general-purpose processors. Since these applications normally have special properties, it is generally assumed that they have poor memory performances compared to traditional applications. In this paper, we performed a comprehensive study of memory performance on a subset representative media benchmarks and we found that:

1) Multimedia applications have lower L-1 data cache miss rate compared to general-purpose applications.

2) Some multimedia applications' performances are hard to get improved by only increasing cache sizes.

3) Media applications have regular memory access patterns that make it easier to be optimized by using data prefetching techniques.

We implemented one-block-look-ahead prefetching techniques under two different memory access models. The simulation results show the prefetch scheme can effectively improve L1 data cache miss rate of multimedia applications. However, because the miss rate of media applications is very low, the benefit of prefetch on CPI is not very significant.

There are some interesting future works. First, it is interesting to continue study on the memory access patterns of different applications. Secondly more intelligent or adaptive prefetching schemes should be further exploited. At last, more specific and advanced computational unit for multimedia applications should be studied to improve performance of media applications.

Acknowledgement

We would like to thank Prof. Rixner and Paul for their helpful feedbacks and comments through the entire process of the project.

7. References

[1] SimpleScalar Tool Set Version 3.0: http://www.cs.wisc.edu/mscalar/simplescalar.html

[2] C. Lee, M. Potkonjak, W.Mangione Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems", Proc. 30th Ann. Int'l Symp. Microarchitecture, pp.330-335, 1997

[3] S. Sohoni, Z.Xu, R. Min, and Y. Hu, "A Study of Memory System Performance of Multimedia Applications", in Proceedings of the ACM Joint International Conference on Measurement & Modeling of Computer Systems (SIGMETRICS 2001), Cambridge, Massachusetts, pp.206-215, June 2001

[4] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, and A. Chang, "Imagine: Media processing with streams," IEEE Micro, pp. 34--46, April 2001.

[5] SPEC2000: http://www.spec.org/cpu/

[6] A. J. Smith, "Cache Memories", ACM Computing Surveys, Vol.14, No 3, pp.473-530, 1982

[7] 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