Prefetching MPEG with Set Hints

Brenda Liu SoYoung Park

Carnegie **M**ellon **U**niversity
Pittsburgh, PA 15213
{bhliu,seraphin}@cs.cmu.edu

Abstract

As the gap between memory latency and processor clock cycle time grows larger, prefetching has been employed in most modern processors in the attempt to hide the memory latency by fetching data that will likely to be used in the near future. This paper applies a technique that exploits the placement of prefetched data lines into the set-associative cache – the prefetching set hints – to the MPEG-2 video decoding application, and assesses the usefulness of the prefetches in this particular domain using the cello simulator. We analyzed the simulation results, and have concluded that the MPEG video decoding application is an area that does not benefit from having set hints.

1. Introduction

Some data have poor locality: they are accessed only once after being brought into the cache from memory. Prefetching these data could evict data with good locality from the cache and cause future cache misses. Based on this observation, we can make an improvement by forcing data with poor locality to occupy only a subset of the cache, while reserving a different part of the cache for data with good locality. Therefore, prefetching the data with poor locality would only evict other data of the same type, and data with good locality will stay in its own subsection of the cache.

While it is easy for a programmer to distinguish data with good or poor locality, it is difficult for the computer to automatically make that decision. With prefetching set hints, the programmer provides a "hint" whether a particular piece of data should reside in the streamed or the retained portion of the cache, and the processor will place the data appropriately according to the hint.

With the growing popularity of streaming multimedia files, the cache pollution by streaming data is creating more problem than ever before. Prefetching with set hints looks promising to solve this problem. Additionally, because it introduces very little change to the memory architecture, it is an appealing low-cost solution to handling data with poor locality. The simplicity of its implementation promises to be a very elegant solution for the applications that would benefit from it.

This paper explores the application of the set hints with the MPEG decoding application. The rest of this paper is organized as follows. Section 2 presents the background information for MPEG video and why the application may benefit from the set hints. Section 3 mentions previous work done with set hints. Section 4 describes the implementation of the simulation environment. Section 5 presents the results of our simulation. Section 6 discusses the various causes behind the outcome. Sections 7 and 8 describe possible future work and conclude the paper.

2. MPEG Background

2.1 Decoder

MPEG is the standard for coding video and audio streams onto compressed digital format to conveniently transport them over the network. In the case of video, MPEG takes advantage of the spatial redundancy within each frame (e.g. single-colored background), and the temporal redundancy between frames (e.g. only a small part of the picture changes from frame to frame) to achieve the encoding. The decoding process is just the reversal of the encoding process. Many decoders support both MPEG-1 and MPEG-2 decoding, and these two standards, as opposed to other MPEG video standards, appear to be the most prevalent on the web both in terms of available software and available information.

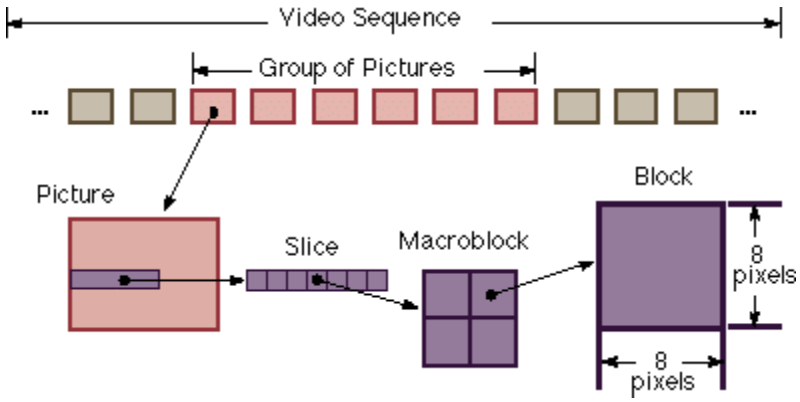

Figure 1. Video Stream Hierarchy [1].

Figure 1 shows the structure of an MPEG video bitstream. It is made of a sequence of GOPs (group of pictures), each of which is made of a series of frames (pictures), each of which is composed of a number of slices, each of which is composed of macroblocks. The encoding and decoding are done in the unit of a macroblock. A macroblock is 16-by-16 pixels in size and is made of 4 blocks. The beginning of each level in this hierarchy is marked by an appropriate header. More specifically, a sequence header announces the following GOPs; each GOP has a GOP header that announces the pictures; and so on.

In an encoded video stream, there are three kinds of frames: I (intra) frames, P (predicted) frames, and B (bi-directional) frames. The I frames are stand-alone, and contain a complete image, using only the spatial redundancy for compression. The P frames can use the closest previous I or P frame for reference, thus utilizing temporal redundancy in addition to spatial redundancy. The B frames achieve the highest level of compression by referring to the closest previous as well as the closest subsequent I or P frames. The B frames themselves cannot be used as references to other frames. Figures 2 and 3 illustrate the interframe relationships just described. Notice that, because of these inter-dependencies, the encoding and decoding orders of frames are different from the display order. As shown in Figure 4, the I and P frames must be decoded before the B frame that needs them. Having these B frames interleaving among I and P frames results in better compression. However, this interleaving places a heavier demand on the memory system because decoding each macroblock of the B frame requires reading two additional macroblocks from two different frames.

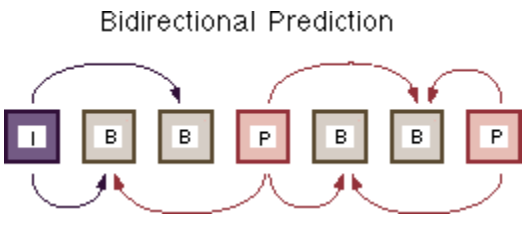

Figure 2. Bidirectional Prediction [1].

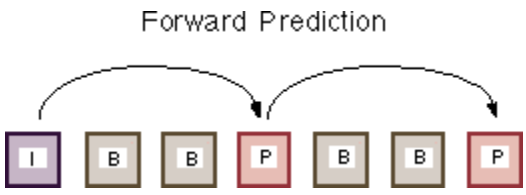

Figure 3. Forward Prediction [1].

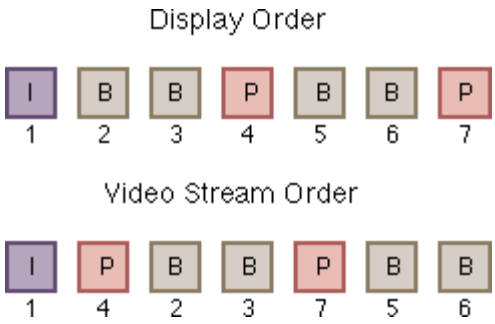

Figure 4. Display order vs. video stream (decode) order.

2.2 Cache Behavior

We have found only one previous work [14] that deals with optimizing data cache performance of an MPEG video decoder. This work classified the data set of a decoder into logical categories depending on access type (read/write) and fraction of references. Table 1 reproduces their findings. Our MPEG experiments were run based on these results. The Input and Output are clearly streaming data. The read-only Tabular constants are retained data. The Reference data is accessed often but is too large to fit into any primary cache of reasonable size, so we decided to treat them as streamed data. That way, they will not evict data marked as retained, and a large portion of them may stay in the secondary data, incurring only a minor penalty compared to going out to main memory. The remaining types, Block and State, are accessed often (more than half of total data accesses) and are small enough to be treated as retained without interfering with other retained data in the cache.

Our basis for applying Soderquist and Leeser's data set classification [14] is the general structure of the decoding algorithm strongly suggested by the video stream structure (Figure 1). This nested structure easily translates into a nested loop structure shown in Figure 5, which is reproduced from their work. The video decoding process is roughly as follows. Every level in the video stream hierarchy represents a loop nested within a higher level. The innermost loop deals with the blocks that make up one macroblock (MB in the figure). Any reference frames are accessed during the macroblock reconstruction, and each macroblock may require access to different parts of the reference frames. IDCT (inverse discrete cosine transform) reverses the DCT applied to the macroblock during encoding. As macroblocks are reconstructed, they are merged with the rest of the picture. This merge and the final output of the finished picture require accessing large amounts of data [14].

Table 1. Access pattern for different data types. Input refers to the movie being decoded. Output refers to the result of the decode. Tabular refers to constants used during decoding. Reference refers to current and any reference frames. Block refers to DCT coefficient and pixel values that belong to a macroblock. State refers to program settings that are not part of image data [14].

Data Type	*Access Type*	*Size*	*Fraction of References*
Input	read/write	2 KB	2.70%
Output	write only	500 KB	3.90%
Tabular	read only	5 KB	5.50%
Reference	read/write	1500 KB	23.70%
Block	read/write	1.5 KB	31.40%
State	read/write	0.5 KB	25.60%

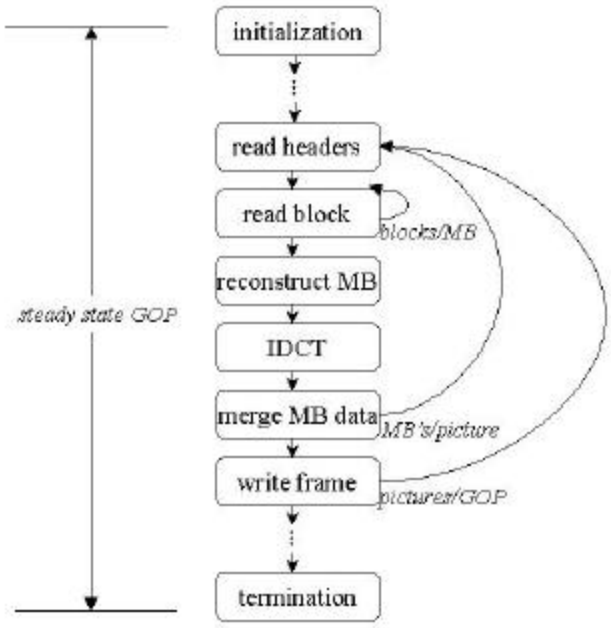

Figure 5. Video decoder model [14].

3. Relevant Previous Work

In Mowry's patent on "Consistently specifying way destinations through prefetching hints" [6], he mentioned a few scenarios that have potential to benefit from the use of prefetching set hints. For example, in blocked matrix multiplication, the "blocked" portion of the data is prefetched into the cache using the retained hint, while the non-blocked data are prefetched using the streamed hint. Therefore, the blocked data are less likely to be replaced from the cache and the latency is essentially hidden [6]. Another example is the multiplication of a matrix by a vector. The vector is prefetched into the cache using the retained hint, while the matrix is prefetched using the streamed hint. The rational for this is that each element of the vector is visited many times to carry out the multiplication while each element of the matrix are visited only once [6]. Some

other examples that are mentioned in the patent include DSP algorithm (retain the filter coefficients while streaming the signal data), bcopy, and bzero (stream all data).

4. Implementation

4.1 Simulator setup

To simulate and monitor the cache behavior of the MPEG decoder program, we use a superscalar multiprocessor performance simulator called cello. One of the reasons we chose cello over other simulators is that it already supports prefetch set hint, so we can focus on researching for suitable programs/algorithms to run instead of spending our time modifying the simulator. Another advantage of using cello is that it gathers extensive statistics and provides it in a straightforward manner so we may better understand the cache behavior.

We set up the system configuration to mirror that of a typical modern processor. We also have two sets of configurations, one with smaller cache (32KB primary and 1MB secondary) and one with larger cache (64KB primary and 2MB secondary). Simulations are run on several PCs running Linux. The details are as follows:

- Processor parameters:
  - Number of processors: 1
  - Clock rate: 250 MHz
  - Cycle time: 4 ns
  - Pipeline type: superscalar
- Instruction cache parameters:
  - Cache size: 64 KB
  - Instruction line size: 64 B
  - Associativity: 2
  - Replacement policy: random
- Data cache parameters:
  - Cache size: 32 KB (exp 1), 64 KB (exp 2)
  - Line size: 64 B
  - Associativity: 2
  - Replacement policy: random
- Secondary cache parameters:
  - Cache size: 1 MB (exp 1), 2MB (exp 2)
  - Line size: 64 B
  - Associativity: 2
  - Replacement policy: random
- Memory latency parameters:
  - Primary to secondary: 15 cycles
  - Primary to memory: 150 cycles
  - Memory access: 10 cycles

4.2 Prefetch levels

Since we want to have some control data to evaluate the cache behavior using prefetching set hint, we set up three different levels of prefetching: level0 includes no prefetch instructions, level1 includes prefetches without the set hint, and level2 includes prefetches with set hint. To do this, we downloaded an open-source MPEG decoder program written in C (mpeg2play), modified it to include the prefetch instructions, and compiled it on a SGI MIPS machine.

4.3 Input movie files

Since simulating a 48-second movie file takes the simulator between 12 to 32 hours depending on the CPU speed, we did not attempt to simulate a typical movie file, which is between one to two hours in length. We chose five representative files, varying in frame size, number of frames, and whether they incorporate interleaving. The information for the input movie files is summarized in table 2.

Table 2. Input movie files information.

File name	*File Size*	*Frame size*	*Number of frames*	*Run time (s)*	*Interleave*
2jaeger	236K	160x120	101	4	No
bicycle	702K	352x240	150	5	Yes
gates	724K	352x240	227	9	Yes
bige	723K	176x144	400	13	No
nightmare	3.5M	320x240	1210	48	Yes

5. Experiments & Results

We performed 30 simulations (2 cache sizes * 3 prefetching levels * 5 test cases), and summarized the results in figures 6 through 8. Here are a few immediate observations, the analysis of these observations are given in the next section:

- Cache hit rate without prefetching is high
- Prefetching (with and without set hints) improves cache hit rate slightly
- Using prefetching set hints made no difference in improving cache hit rate compared to prefetching with no set hints
- Interleaved movie files obtain a bigger performance gain with prefetching
- Execution time goes up slightly with prefetching, but is small enough to be negligible

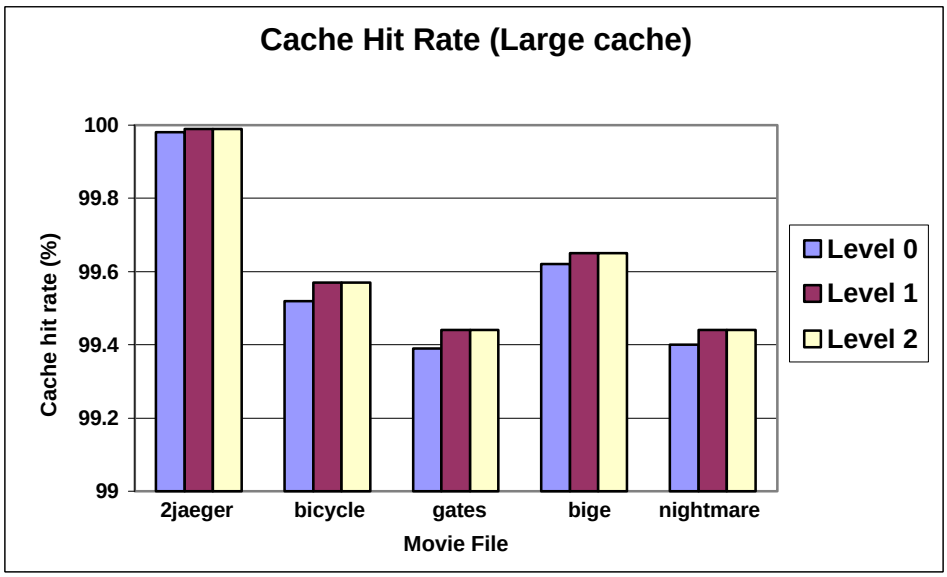

Figure 6. Cache hit rate with large cache.

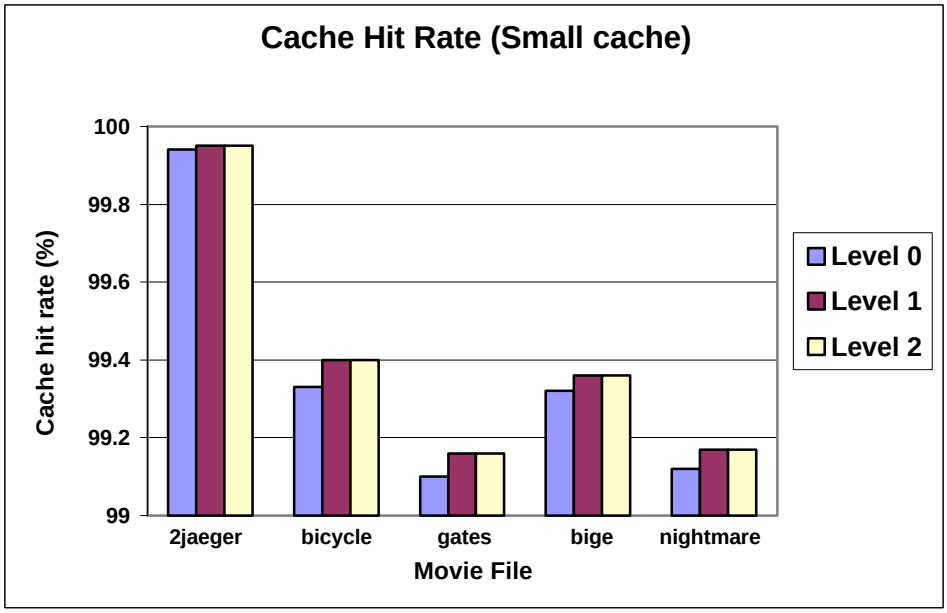

Figure 7. Cache hit rate with small cache.

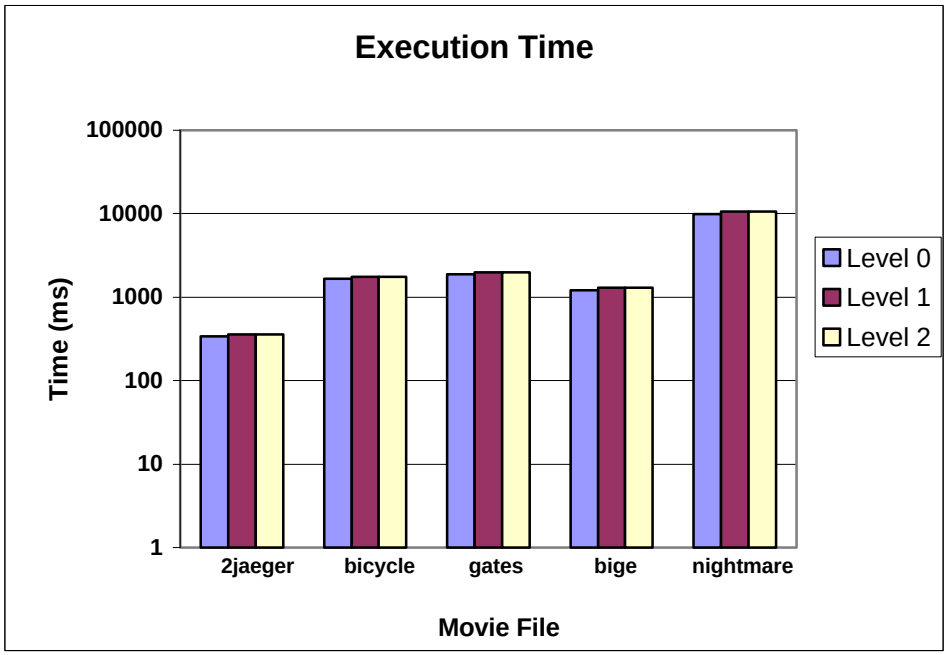

Figure 8. Execution time

6. Explanation of Results

The most obvious observation is that the data cache hit rate is very high even without prefetching. This leaves very little room for any kind of prefetching to improve the hit rate. This is true even for simulations run with data files whose size is much larger than the sum of the primary and secondary data cache size. So the inserted prefetch instructions contribute more to the execution overhead than to the reduction of the cache miss latency, resulting in the very small increases in execution time.

Bicycle has the largest average frame size of 4.68 KB. For a B frame with two reference frames, the working set is 14.04 KB, which fits well within the streamed way of the smallest primary data cache that we specified for cello. So this avoids capacity misses, which we are not interested in. However, it is quite peculiar that the data almost always seem to be in the cache when needed.

One possible explanation for this is that the out-of-order execution engine of cello sends for the data upon seeing a load or store, then reorders instructions so that it executes other instructions to hide the latency while the load or store waits in the reorder buffer. When the data arrives, the load or store is executed and hits in the primary data cache. If this were the case, then it would also explain the equal hit rates between prefetching with and without set hints. If the data is already in the cache, the prefetch instructions can do absolutely nothing.

7. Conclusions

We have analyzed the MPEG decoding algorithm to determine which categories of data, if any, may benefit from using prefetching set hints. We then simulated three levels of prefetching (no prefetch, without set hints, and with set hints) on the MPEG video decoder mpeg2play. Using the cello simulator, we ran test cases with two different cache settings, five input files, and three prefetching levels totaling 30 test cases, and have obtained cache measurements on cache hit rates and CPU time. Our results come as a surprise. They show very little improvement, if any, and even worse performance in some cases.

Even though the results were not what we expected, they were consistent. We offered possible explanations for the behavior, and conclude that the MPEG video decoding application is an area that does not benefit from having set hints. If our intuition is true in that the high cache hit rates are due to the simulated superscalar processor's out-of-order execution, then most of today's machines running MPEG video decoders would not benefit from the set hints.

Although we have not been able to demonstrate an application that can significantly benefit from the prefetching set hints, we believe firmly that the idea of having set hints still has potential. We have considered other applications that exhibit the need to prevent streamed data from evicting retained data in the cache; it is quite possible

that an application with lower cache hit rate may improve more significantly than the MPEG decoder program.

8. Future Work

MP3 (MPEG-1 layer 3) encoder is another possible application that may benefit from the set hints. The MP3 standard defines the format for an encoded MP3 file that takes 10 times less space than the original analog sound file without any noticeable loss of audio quality. The task of an MP3 decoder is very well defined, and involves reconstruction of audio frames from reference frames [4], much like the MPEG video decoding. Having already explored an application like the MP3 decoder, however, it seems more interesting to try a somewhat different application. An MP3 encoder, with its mixture of streamed input and retained values, may be a better choice. But the problem we would face with that choice is that encoders vary widely in implementation. The only common aspects among different encoders are that they make two passes on the data file, first with a psychoacoustics model and then with Huffman compression [4].

Search engines is another category of applications that we think may benefit from set hints, including web search, optical character recognition (OCR), and speech recognition software. These programs must process a large amount of data to find some specific item they are looking for. The items of interest (text strings/sound clips/graphics to match) and most program states information should be retained while the processed data should be streamed. This, however, will require a higher set associativity to ensure that we are not wasting space in the retained side of the cache, but equally distributing the retained and streamed loads into the cache.

9. Acknowledgements

We would like to thank Todd Mowry for the many productive discussions on the topic of prefetching set hints. Shimin Chen also provided us with helpful information about simulating programs with cello.

References

[1] Shanawaz Basith. MPEG: Standards, Technology and Applications. In *Surprise 96*, Vol. 2, Imperial College of Science Technology and Medicine. 1996.

[2] Chad Fogg. MPEG-2 FAQ. Available at http://bmrc.berkeley.edu/frame/research/mpeg/mpegfaq.html.

[3] Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. In *Communications of the ACM* Vol. 34, No. 4, April 1991

[4] Scot Hacker. MP3: The Definitive Guide. Chapter 2. O'Reilly & Associates, March 2000.

[5] Teresa L. Johnson and Wen-mei W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *Proceedings of the 24th international symposium on Computer architecture*. 315-326. 1997.

[6] Todd C. Mowry. US Patent US5732242: Consistently specifying way destinations through prefetching hints. Silicon Graphics Inc., 1998.

[7] Todd C. Mowry. Tolerating latency in multiprocessors through compiler-inserted prefetching. In *ACM Transactions on Computer Systems,* Vol. 16, No. 1, 55-92, February 1998.

[8] Todd C. Mowry. Tolerating Latency Through Software-Controlled Data Prefetching. In *Ph.D. thesis*, Stanford University, Computer Systems Laboratory, March 1994.

[9] Davis Yen Pan. Digital Audio Compression. In *Digital Technical Journal*, Vol. 5, No. 2, Spring 1993.

[10] Ketan Petel, Brian C. Smith, and Lawrence A. Rowe. Performance of a Software MPEG Video Decoder. In *Proceedings ACM Multimedia 93*, pp. 75-82.

[11] Parthasarathy Ranganathan, Sarita Adve and Norman P. Jouppi. Reconfigurable caches and their application to media processing. In *The 27th Annual International Symposium on Computer architecture*. 214-224. 2000.

[12] MIPS R10000 Microprocessor User's Manual (Version 2.0). SGI. October 1996.

[13] T. Sikora. MPEG Digital Video Coding Standards. In *Digital Electronics Consumer Handbook,* McGraw Hill Company, 1997.

[14] Peter Soderquist and Miriam Leeser. Optimizing the Data Cache Performance of a Software MPEG-2 Video Decoder. In *ACM Multimedia*, 1997.

[15] Dean M. Tullsen and Susan J. Eggers. Effective Cache Prefetching on Bus-Based Multiprocessors. In *ACM Transactions on Computer Systems,* Vol 13, No. 1, Pages 57-88, February 1995.

[16] Daniel F. Zucker, Michael J. Flynn, and Ruby B. Lee. A comparison of hardware prefetching techniques for multimedia benchmarks. In *Technical report CSL-TR-95-683,* Stanford University, 1995.