A COMPARISON OF HARDWARE PREFETCHING TECHNIQUES FOR MULTIMEDIA BENCHMARKS

Daniel F. Zucker, Michael J. Flynn, and Ruby B. Lee

Technical Report: CSL-TR-95-683

December 1995

This work was supported by NASA under contract NAG2-842 and Hitachi America, Ltd. with equipment provided by Hewlett-Packard under gift No. 23487.

A Comparison of Hardware Prefetching Techniques For Multimedia Benchmarks

by

Daniel F. Zucker, Michael J. Flynn, and Ruby B. Lee

Technical Report: CSL-TR-95-683

December 1995

Computer Systems Laboratory

Departments of Electrical Engineering and Computer Science

Stanford University

Stanford, California 94305-4055

Abstract

Data prefetching is a well known technique for improving cache performance. While several studies have examined prefetch strategies for scientific and commercial applications, no published work has studied the special memory requirements of multimedia applications. This paper presents data for three types of hardware prefetching schemes: stream buffers, stride prediction tables, and a hybrid combination of the two, the stream cache. Use of the stride prediction table is shown to eliminate up to 90% of the misses that would otherwise be incurred in a moderate or large sized cache with no prefetching hardware. The stream cache, proposed for the first time in this paper, has the potential to cut execution times by more than half by the addition of a relatively small amount of additional hardware.

Key Words and Phrases: prefetching, cache, multimedia, mpeg

Copyright © 1995
by
Daniel F. Zucker, Michael J. Flynn, and Ruby B. Lee

Contents

1 Introduction **1**

2 Related Work **1**

3 Methodology **2**
- 3.1 Applications 2
- 3.2 Simulation Techniques 4
- 3.3 Performance Metrics 4
- 3.4 Memory Bandwidth 5

4 Stream Buffers **5**

5 Stride Prediction Tables **8**

6 Stream Caches **11**

7 Additional Stream Cache Improvements **12**
- 7.1 Transient Data Isolation 16
- 7.2 Truncated Table Size 16

8 Execution Time **16**

9 Conclusions **17**

List of Figures

1 Baseline miss rates 3
2 Hit rates for 4 and 8 way stream buffers 6
3 Hit rates for 16 way stream buffers 7
4 5-way Stream Buffer Architecture 7
5 Hit rates caches employing various size stride prediction tables 9
6 Hit rates caches employing various size stride prediction tables 10
7 Stride Prediction Table Architecture 11
8 Stream Cache Architecture 12
9 Hit rates for 128 and 256 entry stream caches using 128 entry stride prediction table 13
10 Hit rates for 512 entry stream caches using 128 entry stride prediction table 14
11 Modified stream cache architecture 14
12 Hit rates for 128 and 256 entry modified stream cache using 128 entry stride prediction tables 15
13 Relative execution times for 128 entry modified stream cache with 128 entry stride table 17
14 Absolute execution times for mpeg_play-hula with 128 entry modified stream cache and 128 entry stride table adjusted for extra area required. 18

List of Tables

1 Benchmark Image Characteristics 2

1 Introduction

As multimedia datatypes and applications become ubiquitous, new processors and architectures will have to support these applications as a matter of course. Our objective is to find low-cost enhancements for standard processor architectures to support multimedia applications. This paper focuses on the memory hierarchy as a means for providing this kind of performance enhancement.

Our earlier work [15] has focused on arithmetic enhancements aimed at improving the computational aspect of multimedia applications. Our current work is targeted at the memory hierarchy, which is of fundamental importance to multimedia system performance. While there has been much work studying memory performance for scientific and general purpose applications, there has been little work on the needs of multimedia applications. Our results show that relatively simple prefetching techniques can significantly improve the memory hit rates for multimedia applications. As processors become faster and utilize increasing instruction level parallelism, memory performance will have a dominating effect on overall processor performance. Improvements in memory performance can eventually result in performance increases of up to 2x with relatively little additional hardware.

An initial problem is that there is no accepted suite of multimedia applications. New applications are constantly emerging and existing applications are undergoing rapid changes. Our strategy, therefore, is to target fundamental kernels that are essential for all multimedia applications. Specifically, we target a workload aimed at data compression. Because of the vast amount of data required to manipulate audio, still images, and movies, data compression is fundamental for all types of multimedia applications. The MPEG and MPEG2 video compression standards were chosen as benchmarks.

2 Related Work

A number of techniques exist for cache prefetching. The idea of prefetching is to predict data access needs in advance so that a specific piece of data is loaded from the main memory before it is actually needed by the application. While a number of papers have been written studying both hardware and software prefetching techniques, no work has looked specifically at the memory behavior of multimedia applications.

The earliest hardware prefetching work was reported by Smith [13] who proposed a one-block-lookahead (OBL) scheme for prefetching cache lines. That is, when a demand miss brings block i into the cache, block i+1 is also prefetched. Jouppi [7] expanded this idea with his proposal for stream buffers. In this scheme, a miss that causes block i to be brought into the cache, also causes prefetching of blocks i+1, i+2, ..., i+n into a separate stream buffer. Jouppi also recognized the need for multi-way stream buffers so that multiple active streams can be maintained for a given cache. He reported significant miss rate reduction. Palacharla and Kessler [9] proposed several enhancements to the stream buffer. They have developed both a filtering scheme to limit the number of unnecessary prefetches, and a method for allowing variable length strides in prefetching stream data.

Another hardware approach to prefetching differs from the stream buffer in that data is prefetched directly to the main cache. In addition, some form of external table is used to

application	image	frame size	number of frames	frame pattern	data memory references
mpeg	hula_2.mpg	352x240	40	IPPIPPI	6e+07
mpeg	easter.mpg	240x176	49	IPBBIBBPBBI	6e+07
mpeg2	tennis.m2v	576x704	7	IBBPBBPP	8e+07
mpeg_encode	tennis.yuv	352x240	10	IPBBPBBPBB	3e+08

Table 1: Benchmark Image Characteristics

keep track of past memory operations and predict future requirements for prefetching. This has the advantage of efficiently handling variable length striding, that is data accesses that linearly traverse a data set by striding through in non-unit steps. Fu and Patel [5] proposed utilizing stride information available in vector processor instructions to prefetch relevant data. They later [4] expanded the application to scalar processors by use of a cache-like look-up table called the stride prediction table.

Chen and Baer [1] have proposed a similar structure called the reference prediction table. Their scheme additionally includes state bits, so that state information can be maintained concerning the character of each memory operation. This is then used to limit unnecessary prefetching. Further analysis of this scheme [2] investigate the timing issues of prefetching by use of a cycle-by-cycle processor simulation. Sklenar [12] presents a third variation on the same theme of the use of an external table to predict future memory references.

A number of techniques also exist to do software prefetching. Porterfield [11] proposed a technique for prefetching certain types of array data. Mowry, et al [8] is generally recognized as having the most practical software prefetch scheme. While software prefetching clearly has a cost advantage, it does introduce additional overhead to the application. Extra cycles must be spent to execute the prefetch instruction, and the code expansion that is often required may result in negative side effects such as increased register usage. Furthermore, software prefetching must be optimized for a given memory architecture and implementation. Despite these disadvantages, though, software prefetching is still a promising technology.

3 Methodology

3.1 Applications

The applications we have chosen for performance measurement are mpeg_play, mpeg2play, and mpeg_encode. Our objective is to choose fundamental algorithms necessary for a broad range of multimedia applications. Because of the extraordinarily large amount of storage volume necessary to store image and video data, data compression is a necessity for any sort of multimedia application. MPEG is rapidly growing in acceptance as the standard for video compression.

MPEG [6] extrapolates JPEG-style compression for motion images. JPEG [14] is a lossy still image compression standard based on discrete cosine transform (DCT). Image data is

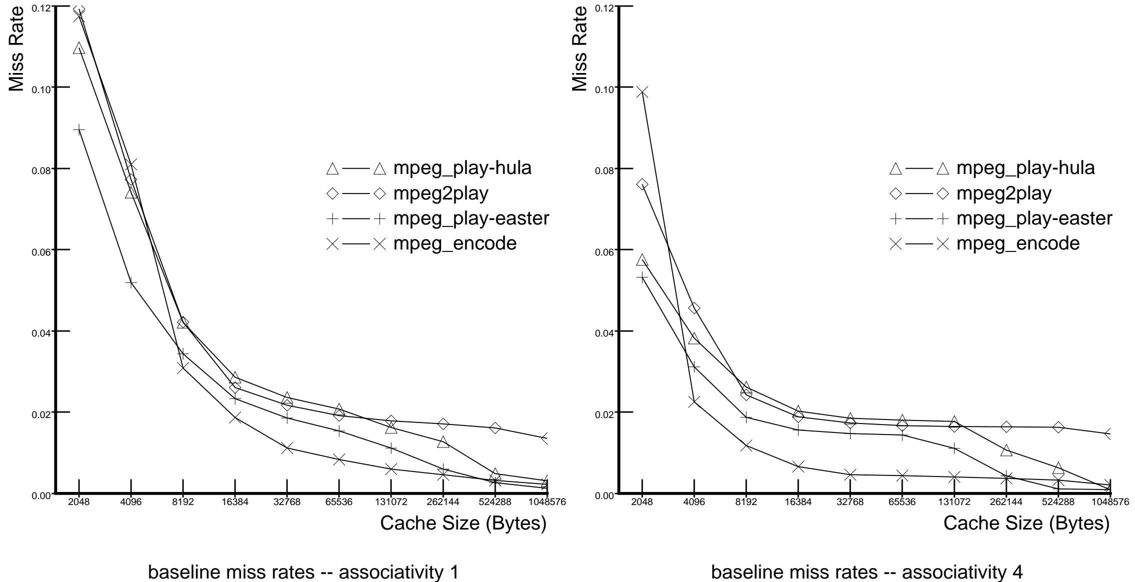


Figure 1: Baseline miss rates

first broken down into 8x8 arrays and is then DCT transformed to convert to the frequency domain. The results of the transform coefficients are then quantized. This is the lossy step in the algorithm. The coefficients are then zig-zag ordered and run-length encoded so that long strings of zeros can be efficiently represented. Finally, these results are encoded using variable length coding. The reverse process is used for decoding.

MPEG uses motion vector techniques in order to take advantage of repeated information between frames for compression. There are three types of frames: I, P, and B frames. I frames are essentially JPEG encoded frames that don't relay on information from any other frames. P frames are forward predicted frames that use motion estimation to extrapolate from previous I frames, while B frames result from a linear combination of predictions from both preceeding and ensuing frames. The MPEG implementations used in this paper are mpeg_play and mpeg_encode from the Berkeley MPEG Group [10]. Mpeg_play was executed using the -dither ordered option for dithering.

MPEG2 is an enhancement of MPEG to allow for an extended range of applications. The primary application intended for MPEG2 was digital television, but it is considered an improvement over MPEG for a number of reasons. The implementation used in this paper is mpeg2play from the MPEG Software Simulation Group's MPEG2 release [3]. Specific characteristics of the images used in our benchmarks are shown in table 1. Memory references refer to both data loads and data stores. Instruction accesses were not simulated. Although the number of frames for each application might seem small, since the miss rate rapidly converges to a stable average after only a few frames, the applications perform similarly to movies with many more frames. Miss rates for these applications run in a baseline cache with no enhancements are shown in figure 1.

3.2 Simulation Techniques

Trace driven simulations are used to model memory behavior in order to determine performance results. Application code was compiled to an assembly language format with the commercially available HP C Compiler version A.09.75. Maximum optimization was set by using the +O3 option. This assembly code was then instrumented using the RYO instrumentation tool for PA-RISC architecture [16]. The instrumentation added additional assembly code so that external library functions are called for every memory access instruction.

To save both execution time and disk space, discrete traces are not written to disk files. Instead, the cache simulator is executed concurrently with the instrumented executable so that address references are dynamically simulated. Because new traces are dynamically generated every execution, variables returned from system calls may cause slightly different traces at each run and may result in some run-to-run variation. The simulator provides data over a wide range of data cache sizes and associativities. A line size of 16 bytes was chosen for all simulations. This line size was chosen so as to better expose the potential benefits of prefetching. Because only a single process was simulated for each cache configuration, it is expected that the performance for the cache sizes reported corresponds to a larger cache size in a real system. Instruction memory accesses are not modelled.

3.3 Performance Metrics

Fraction of misses eliminated is the primary performance metric. This metric judges the performance of a given prefetch scheme independent of the particular cache implementation. A perfect prefetching scheme would eliminate all memory misses. This would have a fraction of misses eliminated value of 1.0 since all misses have been eliminated. Similarly, an architecture that eliminates half of all the misses of a cache with similar size and associativity would have a fraction of misses eliminated value of 0.5.

In the case of a second level cache, the fraction of misses eliminated metric is identical to the hit rate for the second level cache. Of all the misses that occur in the first level cache, the fraction of those that hit in the second level cache is, by definition, equal to the fraction of misses eliminated. The reason for using fraction of misses eliminated instead of second level hit rate is for those configurations such as the stride prediction table and modified stream cache, discussed in sections 5 and 7, where no discreet second level cache exists. In this way, comparisons with a common metric can be used across all cache configurations in the study.

This metric is desirable for a number of other reasons as well. In this way, performance improvement can be judged independently of other cache design considerations such as main cache size and associativity. The size of the main cache will have a dominating effect on miss rate, so that if results were simply compared in terms of absolute miss rates, the variation due to cache size would tend to mask out the variation due to prefetching scheme. Furthermore, performance can also be judged independently of memory implementation parameters such as time to access main memory. If this were not the case, varying memory parameters such as cycles to fill a main cache line could have a significant impact on results.

Results are also reported for execution time in numbers of cycles. For these results, an aggressive memory-limited processor model is assumed. An n-way superscalar processor is assumed such that there are sufficient resources to perform any non-memory operation in a single cycle. Therefore, the only limit to computation time is the number of memory operations. This model shows the maxium impact that the memory system architecture has on performance.

3.4 Memory Bandwidth

For the purposes of this study, memory bandwidth is assumed to be large enough such that this is not a limiting factor on performance. This assumption is made to study the effects of differing prefetch strategies independent of memory bus architectures. It is recognized that this assumption may not be valid in terms of today's architectures. However, the trend for wider bandwidth to memory indicates that this may not be a problem in the future.

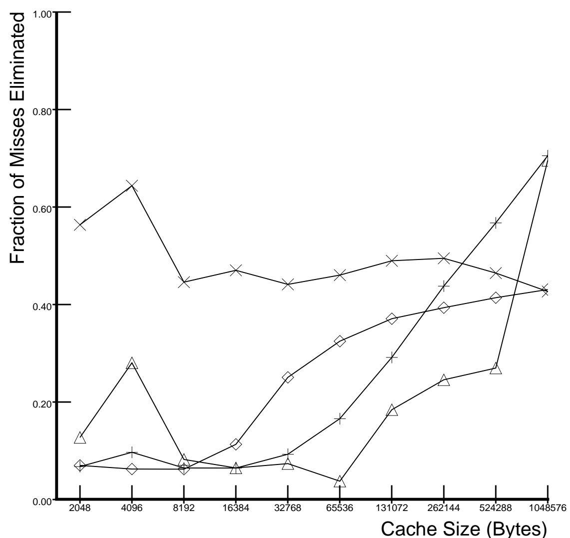
All architectures studied in this paper rely on significantly increasing accesses to main memory in the form of increased prefetching. Techniques for filtering ([9] and [2]) exist to address this issue. However, they may have a negative impact on total number of misses captured. Since the goal of this paper is to compare the upper limit of cache performance between the differing cache configurations, these filtering techniques have not been implemented in the cache models used.

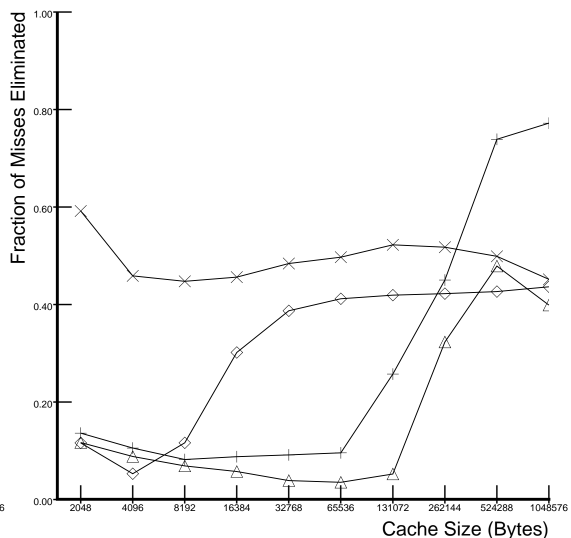
4 Stream Buffers

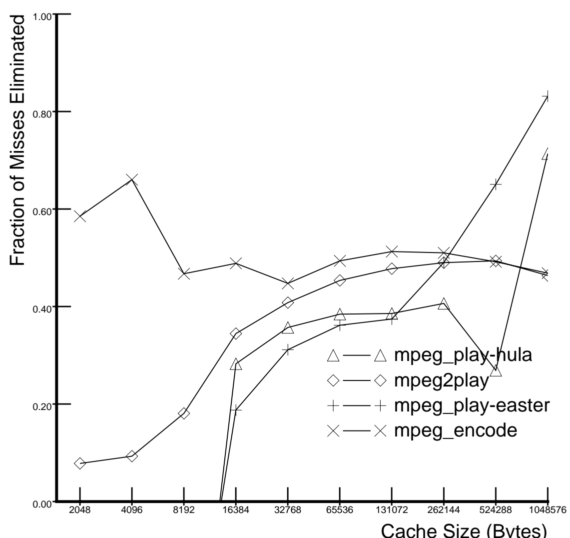
The stream buffer architecture simulated is shown in figure 4. As proposed by Jouppi, the stream buffer is a FIFO type queue that sits on the refill path to the main cache. A new stream is allocated at each successive data cache miss. When all stream buffers are allocated, the next data cache miss will replace the stream least recently accessed (LRU replacement). A memory access that misses in the main cache, but hits in the stride buffer is counted as a hit. Since the refill time from the stream buffer can be an order of magnitude faster than a refill from main memory, this assumption should not significantly affect the reported results.

Our simulations assume 16 parallel stream buffers. This number is selected to be large enough so that the the number of stream buffers is not a limiting factor in performance. We also simulate a stream buffer depth of 5 entries. Palarcharla [9] proposed an enhancement to the stream buffer to filter unnecessary excess prefetches. However, because memory bandwidth is not a limiting factor in our model, this could only potentially hurt performance and was therefore not included. He also proposed a mechanism to allow non-unit striding through the data. This requires a software bit mask that must be individually adjusted for a given application and architecture. Due to the impracticality of doing this for a real system, this was also not included in the model.

Performance data across a range of cache sizes and with direct mapped and 4-way associativities are shown in figure 2 and figure 3. For most applications, the stream buffer tends to peak out at eliminating about 50% of the misses. Mpeg_play playing easter.mpg is







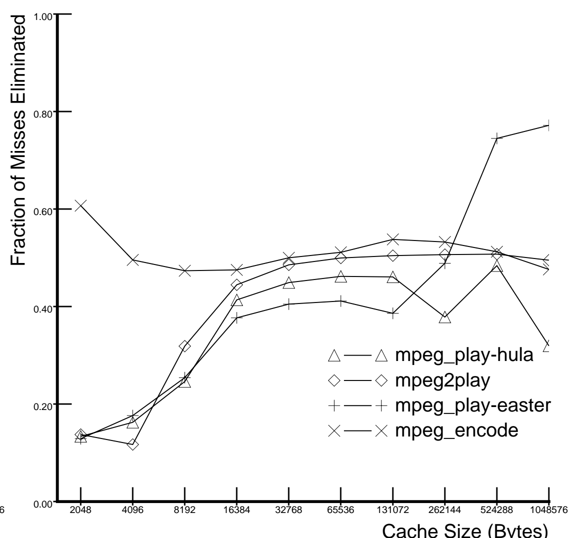


Figure 2: Hit rates for 4 and 8 way stream buffers

Figure 4: 5-way Stream Buffer Architecture

the single exception and can eliminate 80% of the misses for very large caches. This seems to be an outlying data point, however.

Thus, in the best case, only 50% of misses are eliminated with the stream buffer. This is because the relatively complicated algorithms involved tend to access the data in a non-unit strides, and the stream buffer is designed to aid only in cases of unit strides. This shows that even with a 16-way stream buffer at most 50 % of misses are eliminated, and therefore the stream buffer is not an effective prefetching technique.

5 Stride Prediction Tables

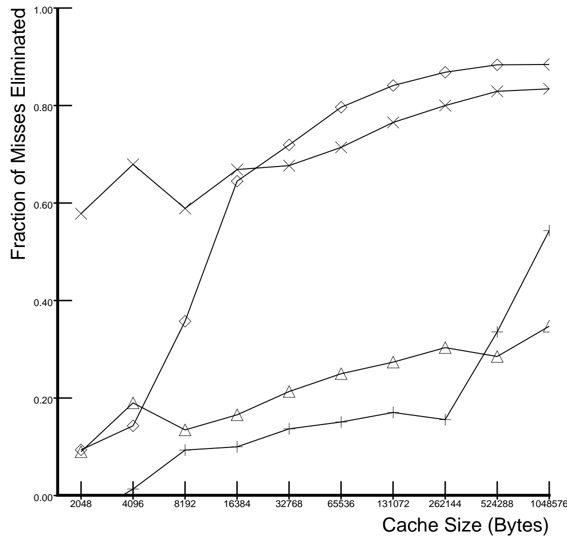
The structure of the stride prediction table (SPT) simulated is shown in figure 7. A table, indexed by instruction address, is maintained for all memory operations and holds the address of the last access. This requires access to the PC (program counter) and may therefore be slightly disadvantageous compared to the stream buffer. The stream buffer, since it relies only on external data requests, may be added more easily than the stride prediction table to an existing commercial processor. When a subsequent memory access is made by an instruction already contained in the stride prediction table, the current memory access address is subtracted from the previously stored address to calculate a data stride value. If this value is non-zero, the next predicted memory item, calculated by adding the stride value and the current memory address, is prefetched into the main cache. When the current instruction does not match an instruction stored in the stride prediction table, an SPT miss occurs. The new entry is added to the SPT replacing the least recently used entry (LRU).

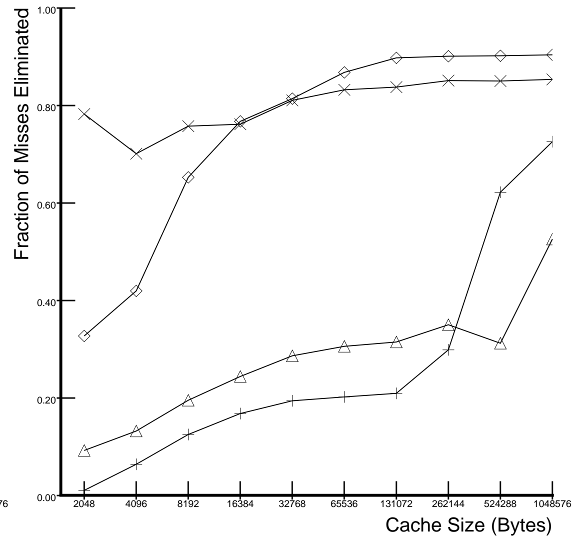
Data obtained from simulations using a variety of stride table sizes is shown in figure 5 and figure 6. All applications perform very well with a stride cache of approximately 128 entries. For large main cache sizes, between 70% and 90% of misses are eliminated relative to a cache with the same size and associativity, but no stride predicition mechanism. A knee in the curve appears, however, at a cache size of approximately 32KB below which the stride prediction table rapidly becomes less effective.

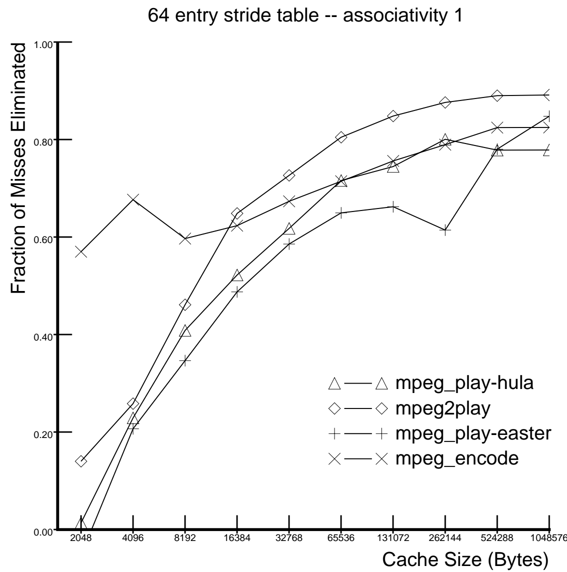
Surprisingly, this is not as major of a factor for mpeg_encode, for which the performance does not appreciably decay for small cache sizes. This is due to the memory intensive motion estimation that must be done for mpeg encoding. The fairly large search space required for motion vector encoding must be repeated for each 16x16 macro block in the image. Although this requires a very large total number of memory references, the memory locality is quite good, and the traditional cache structure performs well, even for very small caches. Therefore, the smaller number of remaining misses that are not captured by the traditional main cache are handled more easily by the stride prediction mechanism.

Finally, an interesting effect is observed for stride prediction tables of greater than 128 entries. In these cases, the stride prediction actually harms memory performance for relatively small cache sizes. The large number of non-useful prefetches begins to remove useful data from the cache. This problem could potentially be solved by the use of filtering techniques.

The stride prediction table works very well for middle and large cache sizes. Indeed, it







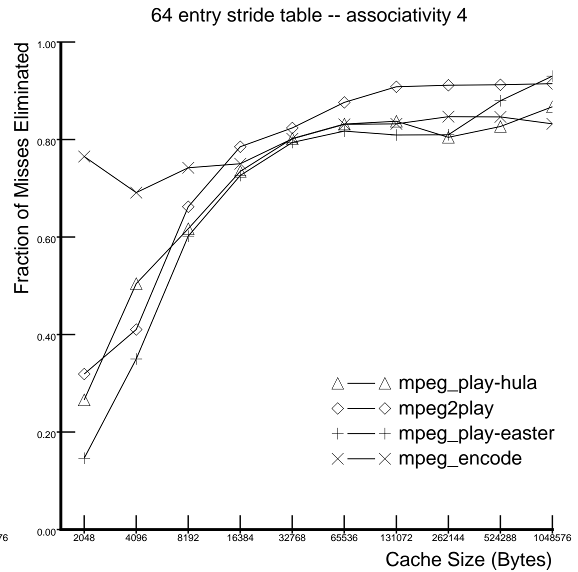
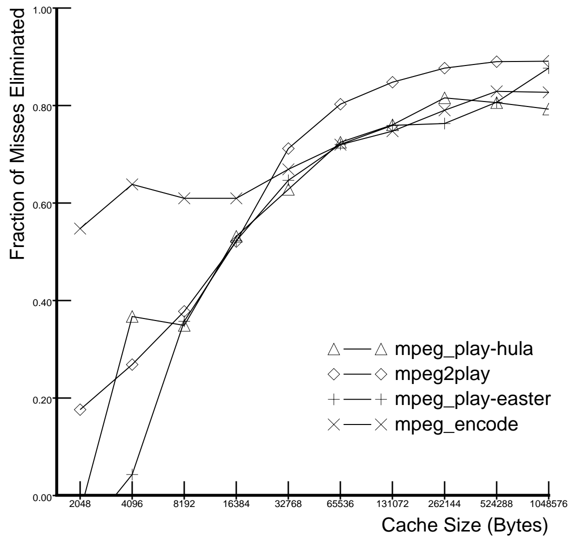
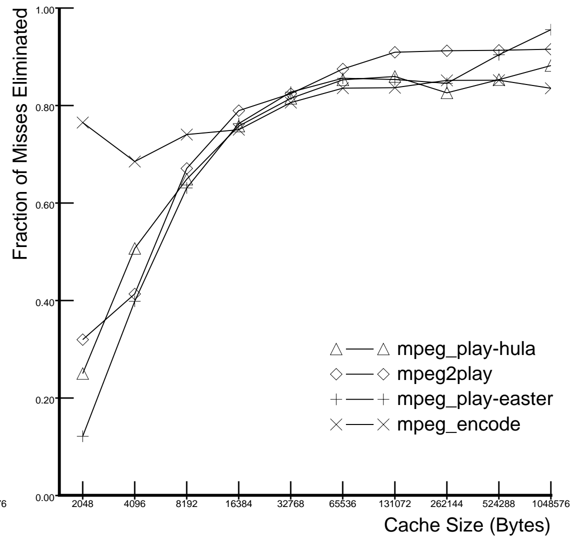


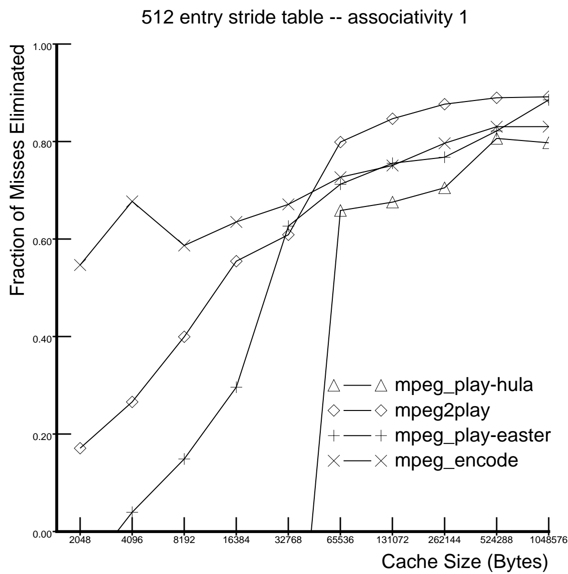
Figure 5: Hit rates caches employing various size stride prediction tables



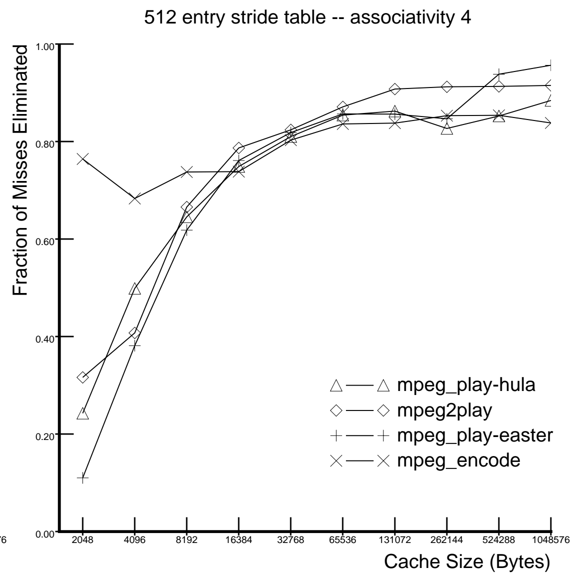
512 entry stride table -- associativity 1



512 entry stride table -- associativity 4



1024 entry stride table -- associativity 1



1024 entry stride table -- associativity 4

Figure 6: Hit rates caches employing various size stride prediction tables

Figure 7: Stride Prediction Table Architecture

would be difficult to do better than eliminating 90% of the misses. In this range, the SPT is an effective means of prefetching. However, the stride prediction table has two significant problems with the smaller cache sizes. A large stride prediction table and small main cache results in unpredictable performance and may be undesirable in a real system. Certain applications may result in improvement, while other applications would be unpredictably degraded. The second problem is that even when the performance of smaller cache sizes is not degraded, it is hardly improved. It is the smaller cache range, where the main cache is exhibiting a high miss rate, that has the most to benefit by successful prefetching. The larger caches, where the stride prediction table eliminates most of the misses, actually have fewer misses to eliminate, so that total execution time would be less impacted. Thus, it is this range where the is the most performance benefit potential that the stride prediction table is least effective.

6 Stream Caches

The stream cache potentially overcomes the problems of the stride prediction table by improving performance for the small cache sizes. A stream cache is a hybrid that combines the best features of the stream buffer and stride prediction table. The stride prediction table does a good job of predicting which data to prefetch, but fails for smaller cache sizes because it prefetches a large amount of unnecessary data. The stream cache uses the stride prediction table to prefetch data not to the main cache, but to a small stream cache that is on the refill path to the main cache. Because the data is not prefetched directly to the main cache, polluting the main cache is not a problem. Only useful data is copied from the stream buffer to the main cache, less useful data is eliminated from the main cache. The stream buffer only works well for unit strides and is inherently configured for a fixed number of streams. If a 16-way stream buffer is used, there should be 16 separate streams of application data for the cache memory to be efficiently utilized.

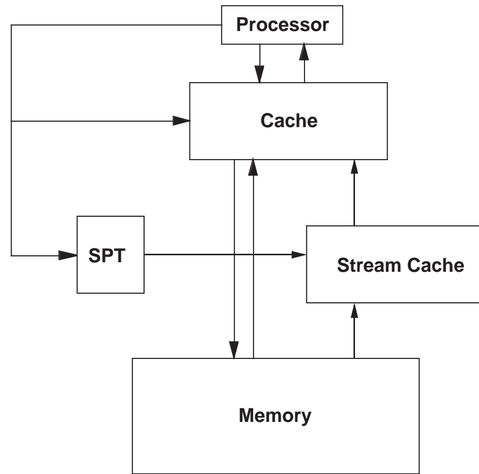


Figure 8: Stream Cache Architecture

The stream cache solves the problems of the stream buffer by uniting the separate FIFOs of multiple stream buffers into one relatively small fully associative stream cache. The stride prediction table is used as before to predict which data to prefetch, but the data is prefetched to the stream cache instead of the main cache. The stream cache is then queried after a main cache miss, and is used to fill the main cache with the desired data. Because the stream cache is unified, the specific number of streams in the application is irrelevant.

Once the data has been copied from the stream cache to the main cache, it is unlikely to be used again in the stream cache. Therefore, a most recently used (MRU) replacement policy is used when fetching new data into the stream cache. The stream cache architecture is shown in figure 8.

Performance data for a 128, 256, and 512 entry stream cache are shown in figure 9 and figure 10. The 512 entry stream cache appears large enough to give a fairly uniform performance improvement of between 60 and 80% across most cache sizes and both associativities. Performance for main cache sizes of less than approximately 32KB is significantly improved over the same cache configurations using only a stride prediction table.

This region on the left part of the graph is significant, since this is where the smaller main caches are not performing as efficiently and memory performance is a much higher percentage of execution time. The 512 entry stream cache, however, would require 8KB of additional memory and may be impractical. The next section introduces modifications to further improve performance for the small caches with the addition of less extra hardware.

7 Additional Stream Cache Improvements

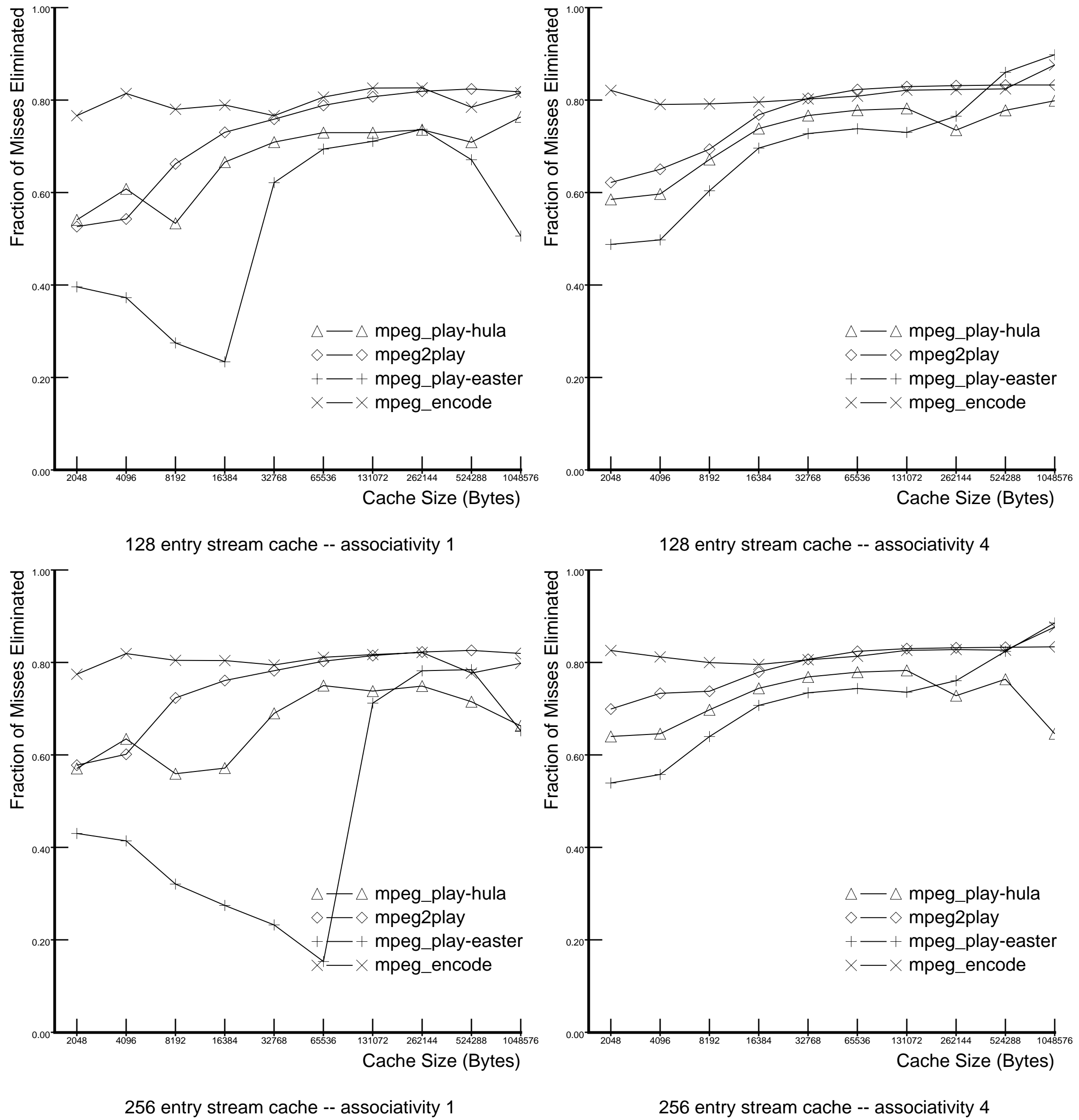


Figure 9: Hit rates for 128 and 256 entry stream caches using 128 entry stride prediction table

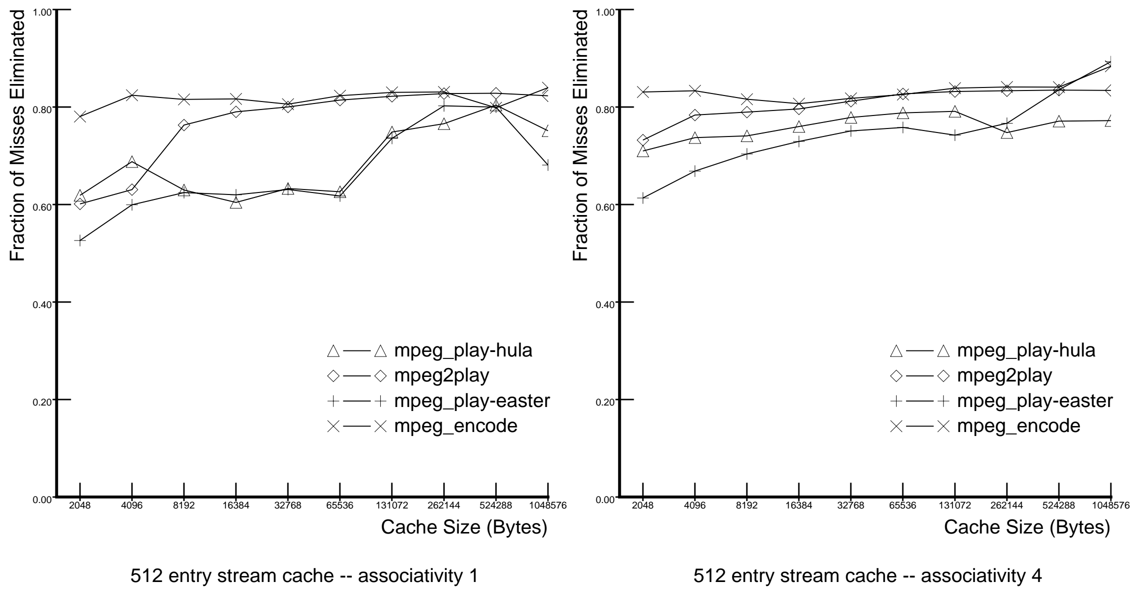


Figure 10: Hit rates for 512 entry stream caches using 128 entry stride prediction table

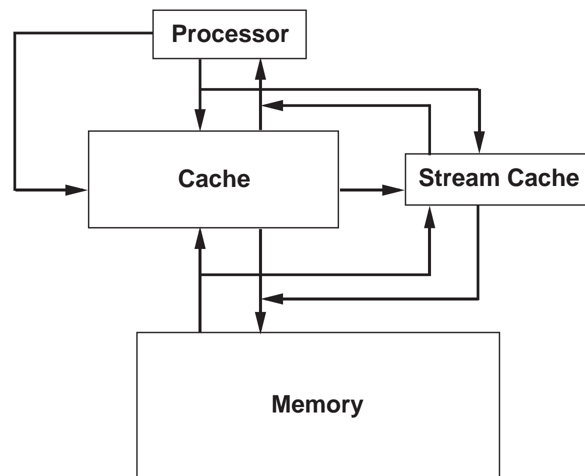


Figure 11: Modified stream cache architecture

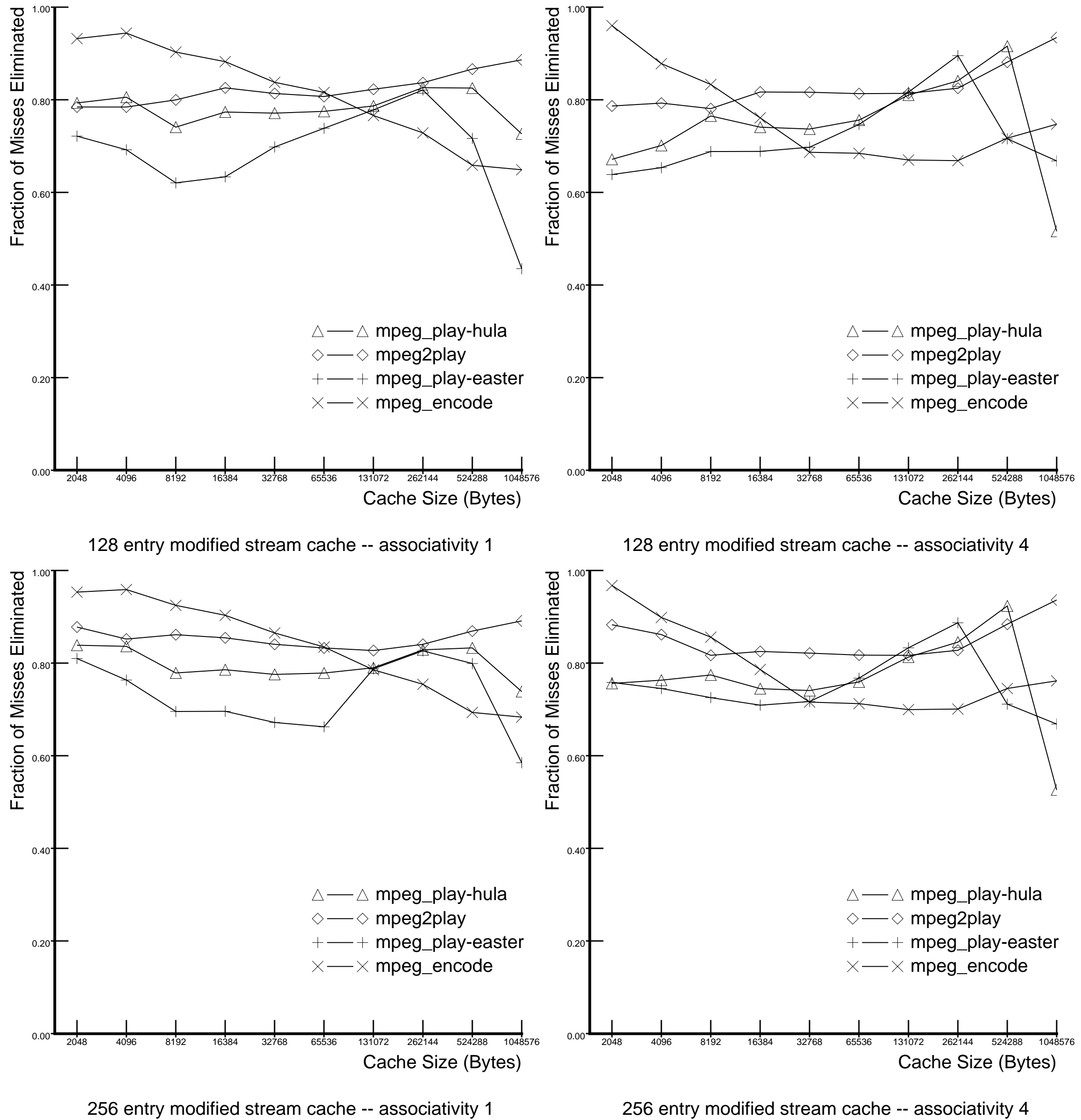


Figure 12: Hit rates for 128 and 256 entry modified stream cache using 128 entry stride prediction tables

7.1 Transient Data Isolation

One enhancement is to move the stream cache from the refill path of the main cache to a position parallel to the main cache. This is shown in figure 11. This is based on the hypothesis that multimedia applications tend to operate on a relatively small workspace of data that marches linearly through the image. The data in this workspace is operated on for a short time, but then is not frequently reused. The goal of the modified stream cache is to isolate this local workspace to the stream cache. Prefetched data is brought into the stream cache, but is not copied into the main cache. A cache access must search both the main cache and the stream cache in parallel. Because the data in the stream cache is reused before it becomes stale, an LRU replacement scheme is now employed.

Miss rate data for a 128 entry modified stream cache with a 128 entry SPT is shown in figure 12. The smaller caches show a greater enhancement than mid sized caches since there is a greater benefit from keeping less frequently used data out of the main cache. For small cache sizes, performance is better than the 128 entry stream cache described in section 6. Furthermore, this is the region where the main cache is suffering from high miss rates, so that this improvement is particularly beneficial. Section 8, in which execution times are compared illustrates this point.

7.2 Truncated Table Size

Each stride prediction table entry must hold a complete instruction address and data address for each entry and valid bit. Thus, not counting the valid bit, a 128 entry stride prediction table requires about 1KB extra area to hold two 32 bit words per entry.

However it is usually not necessary to store the full instruction address. Storing only the lower 16 bits as an address tag produces similar results. Unfortunately, because of the methodology used to generate this simulation data, the actual instruction address was not used as an index and therefore it is not possible to test this assumption.

The size of the data address field could also be reduced to 16 bits since it is unlikely to encounter data strides of more than 64KB. Employing this technique allows a 128 entry table to be built using only approximately 512B of area.

8 Execution Time

Finally, relative execution times to a baseline cache of the same size are shown for a modified stream cache in figure 13. Until here, fraction of misses eliminated was used as the primary metric for comparing various cache designs since it allowed for evaluation independent of specific design parameters. This section presents data for execution time so as to show the impact the cache would have on actual application performance.

The execution time is shown as a fraction of cycles required for a cache of the same size and associativity but without a stream cache. Execution time is calculated assuming a main memory latency for both the main and stream caches of 25 cycles. If data is needed while it is in the process of being loaded to the cache, then the balance of cycles remaining is counted in total execution time.

Absolute execution times for a single application are shown in figure 14. The horizontal axis is adjusted such that total area, including both the main cache and stream cache, is shown for the enhanced cache. For very small cache sizes, the stream cache can cut the execution time in half. For cache sizes of up to about 256 KB, less than 80% of the original time is required for execution. For very large cache sizes, the traditional cache design does a fairly good job of capturing the working set and the stream cache is proportionately less beneficial or even detrimental in some cases. In the case of large caches, then, the stride prediction table alone is an effective means of prefetching. As image sizes become larger, however, this break point will shift to the right and the stream cache will be useful over a larger range of caches. The reader is also reminded that only one application is executed in the cache, so that a real system executing multiple applications will have the performance of the smaller caches as per our graphs.

This data suggests that the stream cache is extremely effective in improving execution time for either a very small on chip cache or a low cost multimedia system using only a small cache. A 128 entry stride prediction table with a 128 entry stream cache adds only about 2.5KB extra area, but cause the 2KB main cache to perform as a baseline 16KB cache or a 4KB cache to perform as a baseline 128KB cache for the application shown.

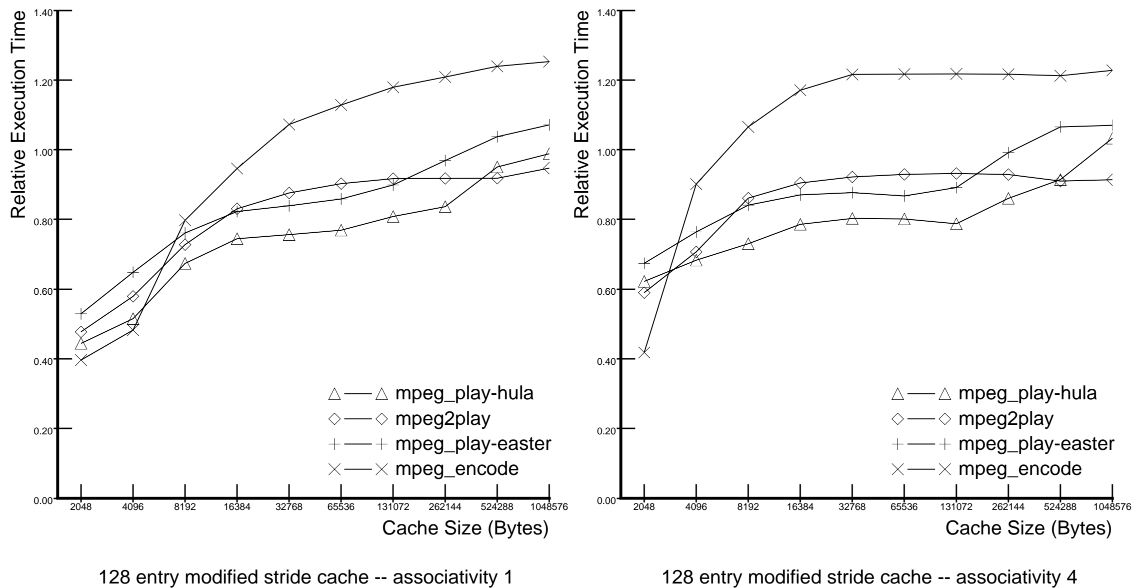


Figure 13: Relative execution times for 128 entry modified stream cache with 128 entry stride table

9 Conclusions

In this paper we investigated a number of hardware prefetching techniques for some common multimedia applications. It is clear that the regular memory access pattern of these applications makes some form of data prefetching an attractive strategy for improving memory

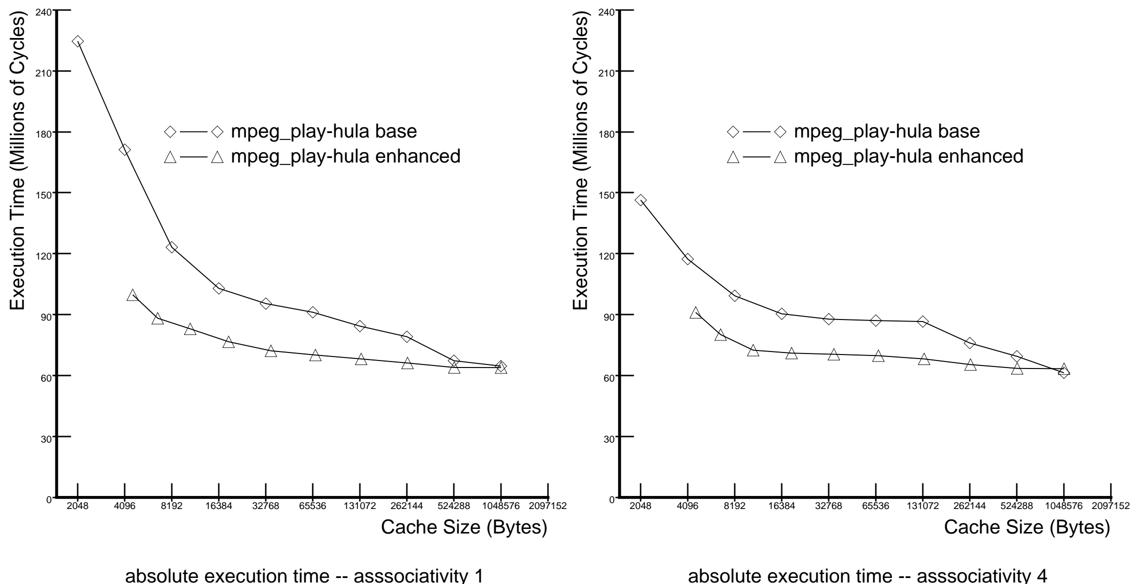


Figure 14: Absolute execution times for mpeg_play-hula with 128 entry modified stream cache and 128 entry stride table adjusted for extra area required.

performance.

We showed that stream buffers can eliminate up to about 50% of data misses for small and moderately sized caches. It is the small cache sizes, where the large number of misses contribute significantly to total execution time, where a large reduction in misses is desirable. The stream cache added improvement over the stride prediction table for smaller sized caches, and left the performance improvements intact for large caches. Finally, the modified stream cache resulted in extremely good performance enhancements for small cache sizes with a small amount of additional hardware, but in some cases did slightly worse than the stride prediction table for large cache sizes.

Data was presented for both direct mapped and 4 way associative caches to show that the same trends exist regardless of associativity, although the improvement is somewhat mitigated for the 4 way associative cache. 4 way associativity was chosen since unpublished simulations showed that 4 way associativity behaves approximately the same as fully associative.

Finally, execution times were simulated for a specific architecture and memory system which showed the modified stream cache could reduce execution time by more than a factor of two for small caches. Because these results are highly dependent on the parameters chosen for the memory models, however, execution time was not exhaustively simulated.

The specific cache sizes reported in this paper will perform as slightly larger caches in a real system because only single applications were run in each cache. The image frame size used also has a significant effect on the break point in a given cache performance curve. For this reason, results should be considered relative to one another, rather than fixed at a given cache size. Although a specific cache simulated here may perform as a smaller cache

in a given system, the relative trends will remain constant regardless.

Several common prefetching schemes were compared for multimedia benchmarks. The stride prediction table was shown to perform extremely well for large caches, and the stream cache and modified stream cache perform very well for small cache sizes. In these cases, significant performance improvements will result from a very small increase in hardware. Extremely cost or area sensitive applications, where a small cache is required, can benefit significantly from employing such a technique.

References

[1] Jean-Loup Baer and Tien-Fu Chen. An effective on-chip preloading scheme to reduce data access penaly. In *Proceedings of Supercomputing '91*, pages 176–186, November 1991.

[2] Tien-Fu Chen and Jean-Loup Baer. Effective hardware-based data prefetching for high-performance processors. *IEEE Transactions on Computers*, 44:318–328, May 1995.

[3] Chad Fogg. mpeg2 codec. ftp:ftp.netcom.com:/pub/cf/cfogg/mpeg2, MPEG Software Simulation Group, 1994.

[4] J. Fu, J. Patel, and B. Janssens. Stride directed prefetching in scalar processors. In *Proc. of the 25th International Symposium on Microarchitecture*, pages 102–110, December 1992.

[5] John W. C. Fu and Janak H. Patel. Data prefetching in multiprocessor vector cache memories. In *Proc. of the 18th Annual International Symposium on Computer Architecture*, pages 54–63, May 1991.

[6] D. Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications ACM*, 34(4):46–58, April 1991.

[7] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of the 17th Annual International Symposium on Computer Architecture*, pages 364–373, May 1990.

[8] T. Mowry, M. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. In *SIGPLAN Notices*, pages 62–73, September 1992.

[9] Subbarao Palacharla and R.E. Kessler. Evaluating stream buffers as a secondary cache replacement. In *Proc. of the 21st Annual International Symposium on Computer Architecture*, pages 24–33, April 1994.

[10] K. Patel, B.C. Smith, and L.A. Rowe. Performance of a Software MPEG Video Decoder. In *Proceedings ACM Multimedia 93*, pages 75–82, August 1993.

[11] A.K. Porterfield. Software methods for improvement of cache performance on supercomputer applications. Technical Report COMP TR 89-93, Rice University, May 1989.

[12] Ivan Sklenar. Prefetch unit for vector operations on scalar computers. *ACM Computer Architecture News*, 20:31–37, September 1992.

[13] Alan Jay Smith. Cache memories. *ACM Computing Surveys*, 14:473–530, September 1982.

[14] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *Communications ACM*, 34(4):30–44, April 1991.

[15] Daniel Zucker and Ruby Lee. Reuse of High Precision Arithmetic Hardware to Perform Multiple Concurrent Low Precision Calculations. Technical Report No. CSL-TR-94-616, Computer Systems Laboratory, Stanford University, April 1994.

[16] Daniel F. Zucker and Alan H. Karp. Ryo: A versatile instruction instrumentation tool for pa-risc. Technical Report No. CSL-TR-95-658, Computer Systems Laboratory, Stanford University, January 1995.