# MEMORY PERFORMANCE OPTIMIZATIONS FOR REAL-TIME SOFTWARE HDTV DECODING

*Han Chen*
Princeton University

*Kai Li*
Princeton University

*Bin Wei*
AT&T Labs Research

## ABSTRACT

This paper shows that the performance bottleneck in software MPEG-2 video decoders has shifted to memory operations, as microprocessor technologies have been improving at a fast rate during the past few years. We exploit concurrencies between the processor and the memory sub-system at macroblock level to alleviate the performance bottleneck. First, the paper introduces an interleaved-block order data layout to improve cache performance. Second, the paper describes an algorithm to explicitly prefetch macroblocks for motion compensation. Finally, the paper presents an algorithm to schedule interleaved decoding and output at macroblock level. Our implementation and experiments show that these methods successfully hide the latency of memory and frame buffer. These techniques improve the performance of an already optimized software MPEG-2 decoder by about a factor of two. On a 933 MHz Pentium III PC, the decoder can play 720p HDTV streams at over 62 frames per second.

## 1. INTRODUCTION

Pure software video decoding has the advantage of being cost-effective and tracking technology well. With various multimedia instruction extensions, software decoding of TV or DVD resolution content without hardware support has become commonplace on today's desktop and notebook computers. However, high resolution HDTV decoding still requires a powerful processor or assistance from a graphics accelerator. This paper presents methods to optimize a pure software MPEG-2 decoder to achieve real-time frame rate for HDTV on an entry-level PC.

In the past, the key to improving the performance of software decoders [13] has been to develop ways to satisfy its computational requirements. Much of the previous work focused on multimedia instruction extensions and effective ways of using such instructions to optimize core functions [11][16][17]. As memory performance has been improving at a much slower rate than the processor during the past decades, the performance bottleneck in a decoder has now shifted to memory operations.

By analyzing the distribution of *Cycles-Per-Instruction* (CPI) of an MMX/SSE optimized software MPEG-2 decoder, we find that the stalling of memory operations significantly increases the CPI of memory intensive functions. On a 933 MHz Pentium III PC, the average CPI of motion compensation is 1.74 and that of display is 14.50, several times higher than that of computation intensive functions.

To solve the memory performance bottleneck problem, we exploit the concurrency between the CPU and the memory subsystem in modern computers. We first introduce an interleaved-block order data layout to improve cache performance. We then describe an algorithm to explicitly prefetch macroblocks for motion compensation. Finally, we present an algorithm to schedule interleaved decoding and output at macroblock level. Our implementation and tests show that these techniques can improve the performance of an already highly optimized software decoder by about a factor of two. The CPI of motion compensation is reduced to 0.7, and that of display 1.07. As a result, the improved decoder plays 720p HDTV at over 62 frames per second.

The rest of the paper is organized as follows. Section 2 describes our methodology and testing environments. Section 3 analyzes a software decoder and identifies the bottlenecks. Section 4 presents and evaluates our optimization techniques. Section 5 discusses related work. Section 6 summarizes our study.

## 2. METHODOLOGY AND ENVIRONMENTS

We use CPI as a measure of how well the instructions in a core function perform. This is a well known technique in computer architecture research to understand the degree of *Instruction Level Parallelism* (ILP). We use Microsoft Visual C++ 6.0 compiler with maximum optimization for speed. We use Intel VTune 4.0 Performance Analyzer to estimate the CPI for functions with sequential or iterative structures. The number is calculated as $CPI = (Ft)/(nI)$, where $F$ is the CPU clock frequency, $t$ is the time spent in the measured function, $n$ is its execution count, and $I$ is its number of instructions. We obtain $n$ by using the VTune profiler, $I$ by hand counting (lacking an automated tool), and $t$ by using VTune's time measure.

Our test platform is a PC with a 933MHz Pentium III processor, 256 MB of PC133 SDRAM, and an NVIDIA GeForce256 graphics card. The PC runs Windows 2000 Professional SP1. We use DirectDraw overlay surfaces with YUYV pixel format for color space conversion. No built-in MPEG-2 support is used.

We use 480p and 720p videos to test the performance of a decoder, see Table 1. They represent both mainstream and high-end applications. SPR (Saving Private Ryan) and matrix (The Matrix) are two DVD clips. Fish is taken with an HDTV video camera, courtesy of Intel Microprocessor Research Lab. Fox5 is a clip recorded from the HDTV broadcast of Fox5 station.

Table 1. Test Videos and Baseline Decoders' Performance

| Stream | Resolution | Frames | Size (MB) | Performance (fps) | | |
|---|---|---|---|---|---|---|
| | | | | V0 | V1 | PDVD |
| spr | 720x480 | 2,514 | 58.7 | 33.5 | 74.9 | 80.3 |
| matrix | 720x480 | 2,320 | 57.5 | 33.0 | 77.6 | 83.1 |
| fish | 1,280x720 | 772 | 27.3 | 14.0 | 32.0 | n/a[1] |
| fox5 | 1,280x720 | 720 | 22.5 | 14.4 | 33.6 | n/a[1] |

1. PowerDVD 2.55 does not decode HDTV streams.

## 3. PERFORMANCE BOTTLENECK

We use the codec developed by the *MPEG Software Simulation Group* (MSSG) [5] as a baseline. The decoder iterates on decoding and displaying a picture. During decoding, it processes each macroblock through three steps: *Variable Length Decoding*

(VLD), *Inverse DCT* (IDCT), and *Motion Compensation* (MC). Each of these processing steps along with *Display* (DISP) has its own characteristics in terms of computation and memory bandwidth requirement.

• The VLD module reads a small amount of compressed data from disk, which can easily fit into the L1 cache of most processors. Because general-purpose processors are not optimized for bit operations, VLD is mostly computation intensive.

• The IDCT module needs only one block of pixel data (64 short integers) plus some tables, which can fit into the L1 cache of a processor easily. It takes a lot of cycles to compute the result; therefore it is computation intensive.

• The MC module is computation intensive for three reasons. First, it needs to average two or four pixels when half-pixel accuracy motion vectors are used. Second, it has to average two macroblocks for bi-directional predictions. Third, it needs to saturate the sum of prediction and residual. MC is also memory intensive because it is essentially a series of memory copies. For even a moderate resolution video, the working set size of three internal frame buffers can hardly fit into either the L1 or L2 cache.

• The DISP module used to be computation intensive, when the color space conversion was done by CPU. Nowadays virtually all graphics hardware have built-in support for YCrCb-to-RGB conversion. DISP has become a pure memory copy, whose speed is limited by the memory and bus bandwidth.

There are many methods to attack the computation intensive parts of the decoder. We apply these known optimizations to the MSSG decoder (Version 0, or V0), including MMX VLD, MMX IDCT routine, $1 \times 1$ and $4 \times 4$ IDCT fastpaths, and SSE MC routines. This results in a highly optimized decoder (V1), with performance comparable to a commercial software decoder (PowerDVD 2.55). As shown in Table 1, V1 is about 120% faster than V0, and only about 7% slower than PowerDVD.

We profile V1 using the fish video. Table 3 shows the time spent in each module and its CPI. We do not include the CPI of VLD, because it is difficult to count its number of instructions. Clearly, the MC and DISP modules dominate the runtime. The 0.57 CPI of IDCT indicates that the two MMX pipelines are working at nearly full throughput and confirms that IDCT is not memory limited. The 1.74 CPI of MC signifies memory stalls in MC. The 14.50 CPI of DISP shows that the CPU stalls severely. The main reason is that it takes many CPU cycles to transfer data in the write buffer to the AGP device. When the write buffer is full, further writes have to stall until an early entry is retired.

Figure 1 shows the time sequences of tasks in CPU, memory bus, and AGP port in V1. The CPU tasks include VLD, IDCT, reads in MC ($R_{MC}$), computation and writes in MC ($W_{MC}$), reads in DISP ($R_{DISP}$), and computation and writes in DISP ($W_{DISP}$). Memory read ($R_{MEM}$), memory write ($W_{MEM}$), and AGP write ($W_{AGP}$) are initiated by CPU tasks, and may cause dependent CPU tasks to stall. This explains the high CPI's of MC and DISP.

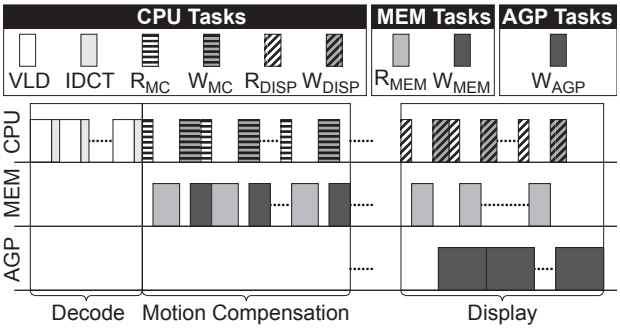


Figure 1. Unoptimized Utilization of System Resources

## 4. MACROBLOCK LEVEL CONCURRENCY

In this section, we propose three techniques to exploit concurrencies among the CPU, the memory sub-system, and the frame buffer. We then evaluate these methods with our implementation.

### 4.1. Interleaved-Block Order of Frame Buffer

Memory caches exploit 1D spatial localities in many applications. In a video decoder, the reference of data exhibits 2D localities. For example, $16 \times 16$ macroblocks are read and written at a time during MC. With a typical scan-line ordered internal frame buffer, such as that in V0 and V1, pixels within an $8 \times 8$ block are scattered across 8 cache lines, thus decreasing cache locality.

Many architectures, such as Pentium III, provide a mechanism called *Write Combine* (WC) [6][7] to improve the memory write performance. Successive partial writes are buffered and collapsed to form a single cache line write, saving an unnecessary read during allocation. Because most processors have only a handful of WC buffers, these partial writes should happen close to each other. With scan-line ordered internal buffers, writing a macroblock requires at least 32 partial writes (16 for luma, and 8 each for chromas), making write combine impossible.

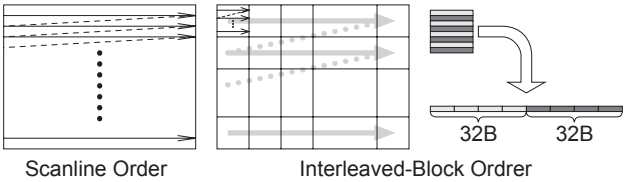


Figure 2. Interleaved-Block Order Layout

To improve cache locality and take advantage of WC, we propose an interleaved-block order of internal frame buffer. A frame buffer is row major ordered on an $8 \times 8$ -block basis. Within a block even scan-lines are stored together, followed by odd scan-lines, as shown in Figure 2. In this layout, a block fits into one or two cache lines, thus increasing locality and enabling WC. The interleaving structure also benefits field pictures and predictions.

### 4.2. Explicit Prefetching of Macroblocks

The working set size of a decoder is at least 3 frames. For 720p videos, it is about 4.1 MB. It can not fit into the L2 cache of any processor. The sequential decoding of macroblocks causes compulsory cache miss for almost every macroblock. Compiler-directed cache prefetching methods [18][19] were proposed. However, the address of reference is data dependent, automatically inserted prefetches are speculative at best. With too few prefetches, cache misses can still occur; but with too many, memory traffic is unduly increased, thus exacerbating the problem.

Fortunately, in an MPEG-2 video decoder, it is possible to prefetch accurately through algorithmic improvement. Because motion vectors are coded in the macroblock header, a decoder can deduce addresses of reference blocks prior to VLD and IDCT, which are not memory intensive. By prefetching reference macroblocks in these steps, the decoder hide the memory latency of MC. Prefetching works best with interleaved block-order, because a block can be prefetched with two instructions.

### 4.3. Interleaved Output and Decode

There are write buffers in the AGP port for fast burst transfer of data. However, writing an entire picture at a time overflows the write buffer and causes the overall speed to be the slowest in the

pipeline. Experiments show that the sustained transfer rate is only about 200MB/s in our system. Display becomes an obstacle to high performance MPEG-2 video decoding.

To achieve real-time HDTV decoding, we exploit the concurrency between the CPU and the AGP write buffer. Instead of copying an entire picture after it is decoded, we break the copying process into small units and interleave them with decoding. Macroblock is the natural granularity to use, because a picture is decoded one macroblock at a time. After the MC of a macroblock in the current frame, one macroblock from the output frame is copied to the graphics card. To further reduce the read memory latency time, we prefetch the output macroblock before MC.

Table 2. Original and New Algorithms

| Original Algorithm | New Algorithm |
|---|---|
| For each picture | For each picture |
| For each macroblock | **If current picture is B-type** |
| Decode MB header and MV | **Set $F_{out}$ to $F_{current}$** |
| For each block | **Else** |
| If a block is coded | **Set $F_{out}$ to $F_{fwd_ref}$** |
| Decode and IQ | For each macroblock |
| IDCT the block | Decode MB header and MV |
| Motion compensation | **Calculate address of ref MB** |
| If current picture is B-type | For each block |
| Display $F_{current}$ | **Prefetch a ref block** |
| Else | If a block is coded |
| Display $F_{fwd_ref}$ | Decode and IQ |
| | IDCT the block |
| | **Prefetch an MB from $F_{out}$** |
| | Motion compensation |
| | **Display the output MB** |

### 4.4. New Algorithm

Combining all three methods together, we arrive at a new decoding algorithm. Table 2 compares it with the original one. The new time sequence of tasks in the system is shown in Figure 3, (PF for Prefetch). Clearly, the algorithm exploits available concurrencies and fully utilizes system resources. When display is the only purpose of decoding a stream, we can further reduce the memory footprint by not storing B pictures in memory, (not shown).

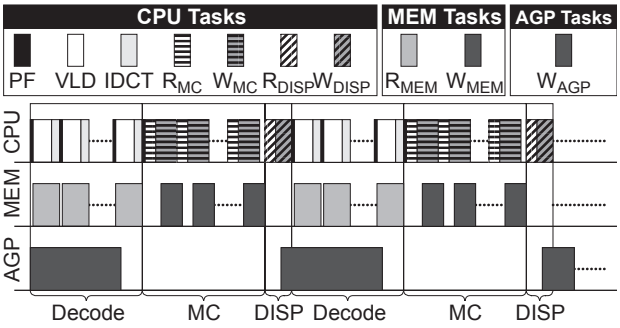


Figure 3. Optimized Utilization of System Resources

### 4.5. Evaluations

To evaluate the effect of each proposed technique, we implement three new versions of the decoder with new features incrementally added. Interleaved-block order is added in V2, explicit prefetching in V3, and interleaved output in V4. We profile all four versions with the fish video and calculate the CPI's for IDCT, MC, and DISP modules, as shown in Table 3. The follows offer a brief explanation for each version.

Table 3. Profiles of Decoders V1 through V4 (fish)

| | V1 | | V2 | | V3 | | V4 | |
|---|---|---|---|---|---|---|---|---|
| | T(ms) | CPI | T(ms) | CPI | T(ms) | CPI | T(ms) | CPI |
| VLD | 4,276 | --- | 4,117 | --- | 4,705 | --- | 5,059 | --- |
| IDCT | 1,437 | 0.57 | 1,390 | 0.55 | 1,509 | 0.59 | 1,454 | 0.57 |
| MC | 9,045 | 1.74 | 6,984 | 1.52 | 3,922 | 0.65 | 4,216 | 0.70 |
| DISP | 8,716 | 14.50 | 8,336 | 10.41 | 8,350 | 10.42 | 882 | 1.07 |
| Other | 628 | --- | 1,161 | --- | 835 | --- | 830 | --- |
| Total | 24,102 | --- | 21,988 | --- | 19,321 | --- | 12,441 | --- |
| FPS | 32.02 | | 35.13 | | 39.96 | | 62.09 | |

• **V2**. Clearly, interleave-block order improves cache locality. The time spent in MC is reduced from 9045 ms to 6984 ms. This is a 23% improvement for an already optimized module. Because of better cache locality and less memory traffic, CPU stalling is reduced. Consequently, the CPI of MC is down from 1.74 to 1.52 and the CPI of DISP from 14.50 to 10.41. We notice that there is a greater reduction in the CPI of DISP than the time of DISP. This is because interleaved-block order requires more complex address calculation. There is a 26% increase of instructions in DISP (706 million in V2 versus 558 million in V1). But this increase is completely absorbed by the latency in AGP.

• **V3**. The accurate macroblock prefetching removes the memory bottleneck in MC. As a result, the total time in MC is reduced from 6984 ms to 3922 ms. The CPI of MC is down from 1.52 to 0.65, indicating nearly full utilization of the two SSE pipelines. Prefetching does introduce overheads, as the times spent in VLD and IDCT are increased slightly. Overall, V3 still performs better than V2. The frame rate increases to near 40.

• **V4**. Interleaved output successfully exploits the concurrency between the CPU and the AGP write buffer. The time in DISP is reduced from 8350 ms to 882 ms, a nearly ten-fold speedup. The CPI is cut from 10.42 to 1.07, indicating full resource utilization. The effective pixel copy rate reaches 1.613 GB/s, exceeding even the theoretical limit of an AGP 4X port. The reason is that the AGP port transactions are hidden, and the decoder sees the write buffer speed. Again, there are small overheads in other modules, but they are overshadowed by the improvement in DISP. As a result, the frame rate is now 62.

We run all four test streams with V0, V1 and V4. Table 4 shows the frame rates and the speedup of V4 versus V0 and V1. The combined optimizations achieve a 1.7 to 2.0 speedup over V1, which is already extensively optimized with MMX/SSE instructions. The overall speedup over the original MSSG decoder is about 4.0 to 4.7. The improvement for HDTV streams is better than that for DVD, because the former is more memory bound than the latter in an unoptimized decoder.

Table 4. Performance Comparison of V0, V1 and V4

| Stream | Frame Rate | | | Speedup of V4 vs. | |
|---|---|---|---|---|---|
| | V0 | V1 | V4 | V0 | V1 |
| spr | 33.48 | 74.92 | 132.72 | 3.96 | 1.77 |
| matrix | 33.03 | 77.64 | 133.97 | 4.06 | 1.73 |
| fish | 14.06 | 32.02 | 62.09 | 4.42 | 1.94 |
| fox5 | 14.38 | 33.61 | 67.57 | 4.70 | 2.01 |

## 5. RELATED WORK

Patel *et al.* studied the performance of a software MPEG-1 video decoder [13] with focus on improving dithering performance. Subsequently, much work has focused on introducing and using multimedia instructions, such as Lee's MPEG-1 decoder on HP

PA-RISC with multimedia extensions [11], Zhou *et al.*'s MPEG-1 decoder on Sun UltraSPARC with VIS extensions [17], and Tung *et al.*'s MMX optimizations for Cyberlink's MPEG-2 decoder [16]. Rahaganathan *et al.* evaluated the performance benefits of multimedia extensions on different architectures [14].

There is a large body of literature on improving cache locality. These efforts include algorithm transformation [1] and tiling [10]. Prefetching techniques are well studied to address the issue of the widening gap between the performance of processor and memory. There are hardware prefetching techniques [8] for programs with sequential accesses, reference prediction table based preloading mechanism [3], software controlled prefetching [9][2], and compiler directed prefetching [12].

Soderquist and Leeser [15] studied the data cache performance of software MPEG-2 video decoders with a hardware simulator. However, they did not provide implementation or simulation results. Zucker *et al.* studied several hardware or compiler-directed prefetching techniques for MPEG-2 video decoding [18][19]. Cucchiara *et al.* [4] proposed architectural ideas to improve multimedia applications in general.

These studies did not address the issues of how to reorganize data structures and algorithms to exploit concurrency among CPU, memory sub-system, and frame buffer at fine granularity.

## 6. CONCLUSION

This paper reports our study on memory performance optimizations for a software MPEG-2 decoder. By analyzing the distributions of CPI in the core functions of an optimized decoder, we find that its performance bottleneck on today's computers is in memory operations. The CPI of the motion compensation module is 1.72 while the CPI of display is as high as 14.5.

Based on the principle of concurrency, we propose and evaluate three optimization techniques, including an interleaved-block order data layout to improve cache locality, an explicit macroblock prefetching algorithm, and an algorithm to schedule interleaved macroblock decoding and output. Our evaluation shows that each optimization improves certain aspect of the decoder and that the combined optimizations can remove the memory bottlenecks in a software MPEG-2 decoder almost completely. The resulting software decoder can decode and display 720p HDTV streams at over 62 frames per second on a 933 MHz Pentium III PC without special hardware support. We notice that using multimedia instructions extensively alone can speedup the original MSSG software decoder by about a factor of two, whereas the memory performance optimizations presented in this paper can further improve the decoder by another factor of two.

We did not test the decoder with 1080i HDTV streams, because the graphics card does not support overlay surfaces as large as $1920 \times 1080$. However, the decoder itself is not limited by the resolution. We plan to find means to evaluate our algorithm for higher resolution streams. Although we implement and evaluate the proposed methods on a Pentium III platform, our methods are not tied to any particular architecture except for the specific prefetching instructions used. We expect the techniques described in the paper to apply to other architectures as well.

## 7. ACKNOWLEDGEMENTS

This project is supported in part by Department of Energy under grant ANI-9906704 and DE-FC02-99ER25387, by Intel Research Council and Intel Technology 2000 equipment grant, and by National Science Foundation under grant CDA-9624099 and EIA-9975011. Han Chen is supported by a Wu Fellowship.

## 8. REFERENCES

[1] W. Abu-Sufah, D.J. Kuck, and D.H. Lawrie. Automatic program transformations for virtual memory computers. *Proceedings of the National Computer Conference*, pp969-974, June 1979.

[2] D. Callahan, K. Kennedy and A. Porterfield. Software Prefetching. *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp40-52, 1991.

[3] T.F. Chen and J.L. Baer. A Performance Study of Software and Hardware Data Prefetching Schemes. *Proc. of the 21st Annual Intl. Symposium on Computer Architecture*, pp223-232, 1994.

[4] R. Cucchiara, M. Piccardi, and A. Prati. Exploiting Cache in Multimedia. *IEEE International Conference on Multimedia Computing and System*, Vol. 1, pp345-350, 1999.

[5] S. Eckart and C.E. Fogg. ISO/IEC MPEG-2 Software Video Codec. *Proc. Digital Video Compression: Algorithms and Technologies*, pp100-109, SPIE, 1995.

[6] Intel Corp. Intel Architecture Optimization Reference Manual, http://developer.intel.com/design/pentiumii/manuals/245127.htm

[7] Intel Corp. Intel Architecture Software Developer's Manual Vol 3: System Programming, Chapter 9, Memory Cache Control. http://developer.intel.com/design/pentiumii/manuals/243192.htm

[8] N.P. Jouppi. Improving Direct-mapped Cache Performance by the Addition of a Small Fully-associative Cache Prefetch Buffers. *Proc. of the 17th Annual Intl. Symposium on Computer Architecture*, pp364-375, 1990.

[9] A.C. Klaiber and H.M. Levy. An Architecture for Software-Controlled Data Prefetching. *Proc. of the 18th Annual Intl. Symposium on Computer Architecture*, pp43-53, 1991.

[10] M.D. Lam, E.E. Rothberg and M.E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. *Proc. of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp63-74, 1991.

[11] R. B. Lee. Realtime MPEG Video via Software Decompression on a PA-RISC Processor. *Compcon '95. 'Technologies for the Information Superhighway'*, pp186-192, 1995.

[12] T. C. Mowry. Tolerating Latency in Multiprocessors Through Compiler-inserted Prefetching. *ACM Transactions on Computer System*, 16(1), pp55-92, Feb. 1998.

[13] K. Patel, B. C. Smith, and L. A. Rowe. Performance of a Software MPEG Video Decoder. *Proceedings of the 1st ACM International Conference On Multimedia*, pp75-82, 1993.

[14] P. Ranganathan, S. Adve, and N.P. Jouppi, Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions. *Proc. of the Annual Intl. Symposium on Computer Architecture*, pp124-135, 1999.

[15] P. Sederquist and M. Leeser. Optimizing the Data Cache Performance of a Software MPEG-2 Video Decoder. *Proc. International Conference on Multimedia*, pp291-301, 1997.

[16] Y. Tung, C. Ho, and J. Wu. MMX-based DCT and MC Algorithms for Real-Time Pure Software MPEG Decoding. *IEEE Intl. Conf. On Multimedia Computing and Systems*, pp357-362, 1999.

[17] C. Zhou *et al.* MPEG Video Decoding with the UltraSPARC Visual Instruction Set. *Compcon '95. 'Technologies for the Information Superhighway'*, pp470-477, 1995.

[18] D.F. Zucker, R.B. Lee, and M.J. Flynn. An Automated Method for Software Controlled Cache Prefetching. *Proc. of the 31st Hawaii Intl. Conference on System Sciences*. V7, pp106-114 1998.

[19] D.F. Zucker, R.B. Lee, and M.J. Flynn. Hardware and Software Cache Prefetching Techniques for MPEG Benchmarks. *IEEE Trans. on Circuits and Systems for Video Technology*, 10(5), pp782-796, Aug. 2000.