MPEG Decoding Workload Characterization

Matthew J. HOLLIMAN, Eric Q. LI, and Yen-Kuang CHEN

Abstract-- **Media workloads have been a principal driving force behind processor designs for several years. While MPEG decoding has been extensively studied in the past, it continues to gain importance as a key workload underlying many present and emerging applications. Additionally, the emerging video coding standard MPEG-4 Part 10, also known as H.264, has some new features that impact the whole system performance. In this paper, we address the characterization of MPEG as well as H.264 decoding on current state-of-the-art superscalar and simultaneous multithreaded (SMT) micro-architectures, discussing both application-level behavior and the key kernels in the applications, e.g., variable-length decoding, IDCT, deblocking filter, and motion compensation. We also address the effectiveness of a number of current micro-architectural enhancements for speeding up this workload.**

***Index Terms*—multimedia, video codecs, MPEG, H.264, microprocessor, simultaneous multithreading**

I. Introduction

As the computing power available to users increases and rich content becomes prevalent, multimedia workloads assume growing importance during processor design and overall system performance assessment. One workload of particular importance is MPEG decoding, which is encountered not only as the basis of standalone applications such as DVD or HDTV playback, but also as a key underlying component in even more demanding applications such as interactive video, video editing, and so forth [20].

To date, computational power has typically increased over time through the evolution from simple pipelined designs to the complex speculation and out-of-order execution of many of today's deeply pipelined superscalar designs. However, while single-threaded processors are now much faster than they used to be, the rapidly growing complexity of such designs also makes achieving significant new gains ever more difficult. This work will first describe the workload characterization of MPEG decoding on current superscalar architectures, and then characterize the same workload on simultaneous multi-threading (SMT) architectures [17]. Specially, we use Intel® processors with Hyper-Threading Technology [14], which is one implementation of the SMT architecture.

The MPEG and H.264 decoders we use as benchmarks are heavily optimized using the latest ISA extensions [20][21]. For our performance analysis, we use a commercial software analysis tool and the performance counters available on today's processors [1][3][7].

The paper is organized as follows. In Section II, we provide a brief review of the basic principles behind most current video codecs, describing particularly the well-established MPEG-2 standard and the rapidly emerging MPEG-4 part 10 (also known as H.264) standard. Section III provides an overview of the overall application behavior of MPEG-2 and H.264 decoding, while Section IV discusses in further detail the implications of the key kernels for current and emerging architectures, and the impact micro-architectural design decisions can expect to have on MPEG decoding performance. Section V addresses application-level implications on threaded architectures. Finally, Section VI concludes the paper.

II. Overview of MPEG standards

The Moving Pictures Expert Group (MPEG) [4][15][16] is a standard group established in 1988. Since then, the group has defined a number of popular video and audio compression standards, including MPEG-1 [10], MPEG-2 [11], MPEG-4 [12], and H.264 [13]. This work focuses on MPEG-2 and MPEG-4 part 10 (H.264) decoding. This section provides a high-level overview of these standards.

A. MPEG-2

MPEG-2 is a popular video compression standard used in a variety of applications, including DVD and HDTV. The standard incorporates three major compression techniques: predictive coding, transform-based coding, and entropy coding. To implement these, the MPEG-2 encoding pipeline consists of motion estimation, discrete cosine transform (DCT), quantization (Q), and variable-length coding (VLC). The MPEG-2 decoding pipeline consists of the counterpart operations of variable-length decoding (VLD), inverse quantization (IQ), inverse DCT (IDCT), and motion compensation (MC).

Because there is temporal correlation between pixels in video sequences, MPEG-2 uses motion-compensated prediction, where only the differences between original images and motion-compensated prediction images are encoded. As all elements in a video scene are approximately spatially displaced, the motion between frames can be described by a number of motion parameters. Moreover, as the spatial correlation between motion vectors is often high, it is sometimes assumed that one motion vector is representative for the motion of a block of adjacent pixels. In general, in MPEG-2, one or two motion vectors are estimated, coded, and transmitted for each 16x16 pixels (macroblock). The basic operations of motion compensation are load, add, and store---load the motion-compensated blocks from the reference frame(s), add the decoded displaced frame difference, and store the decoded blocks back to the frame buffer. The prediction may be formed either from the closest preceding reference frame, or as the interpolation of the closest reference frames in both forward and backward directions. Because the motion vector is the same for a macroblock, the above equation is very suitable for SIMD operations.

Roughly speaking, operations in the MPEG-2 motion

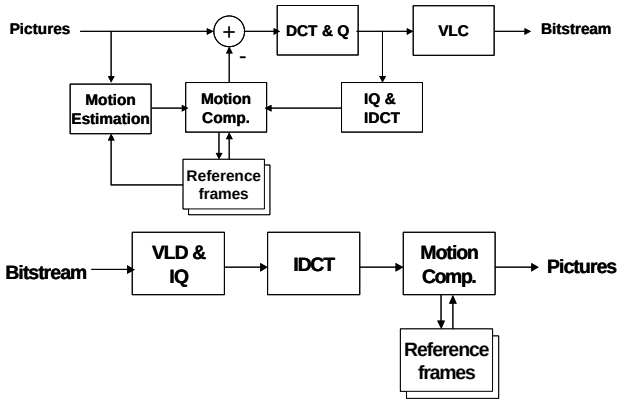

Figure 2.1. Block diagram of MPEG-2 encoder and decoder

compensation module are quite simple. Besides additions (including the averaging operations required in interpolation), most of the operations are memory accesses.

In transform-based coding, the image is transformed into a more compact representation. The most popular transformation in current standards is the discrete cosine transform (DCT). Because there is some spatial redundancy between adjacent pixels, most of the energy of the signals is compacted into a few coefficients after the DCT transformation. Following this, the number of non-zero coefficients is further reduced by quantization, which is the lossy part of the compression standard. The quantization process projects the continuous values of the resulting transformed coefficients into a finite set of symbols, each representing an approximation of the coefficient's value.

The MPEG standards use an 8x8 DCT/IDCT to transform spatial-domain pixels to frequency-domain coefficients or vice versa. Because the transform is separable, most implementations use two one-dimensional 8-point DCTs (or IDCTs), one vertically and one horizontally. The one-dimensional N-point DCT is given by the following equation:

$$y_n = c_n \sum_{k=0}^{N-1} \cos\left(\frac{n(2k+1)}{2N} p\right) x_k$$

where $c_0 = \frac{1}{\sqrt{N}}$ and $c_n = \sqrt{\frac{2}{N}}$ for n=1, …, N-1. Thus, most operations in the DCT/IDCT modules are mathematical operations.

In entropy coding, the non-zero coefficients are encoded using a variable-length code (VLC), whose length is based on its statistical likelihood of occurrence. Normally, a special scan order of non-zero coefficients is performed first to better exploit the statistical occurrence of zero-valued coefficients in an energy-compacting transform. Then, using a combination of run-length and entropy coding, the non-zero coefficients are encoded into the final bitstream. The key to reducing bit rate is to encode the most commonly occurring symbols with the fewest number of bits. Given their probabilities of occurrence, variable-length coding normally reduces the number of bits needed to encode a string of symbols. On the other hand, since the number of bits in the coming bitstream is highly variable, VLD is inherently serial with substantial data dependency; the length of the current codeword is not known until it has actually been decoded, so decoding of future codewords in the stream is dependent on decoding the current codeword.

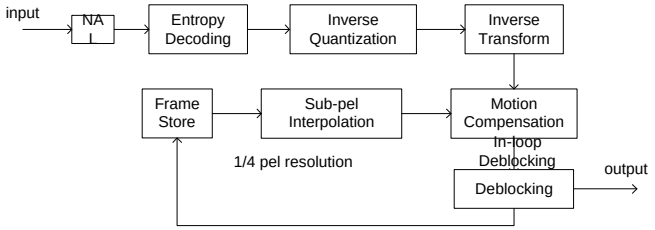

Figure 2.2 Decoder diagram of H.264

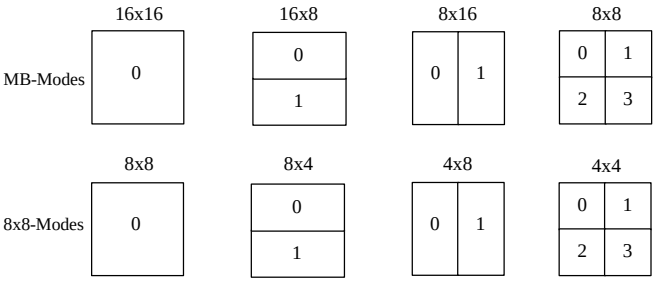

Figure 2.3 Block Models

An MPEG-2 decoder implements the reverse operations of the preceding, i.e. VLD, IQ, IDCT, and MC. The block diagrams of the encoder and corresponding decoder are shown in Figure 2.1.

B. H.264

H.264, also known as MPEG-4 part 10 "Advanced Video Coding," is the latest video coding standard aimed at very low bit rate real-time communication with both enhanced coding efficiency and low end-to-end delay. Figure 2.2 shows the decoder diagram of the standard. The underlying coding scheme defined by H.264 is similar to that employed in the prior MPEG video coding standards. It includes the use of translational block based motion compensation, DCT-like residual coding, scalar quantization, zigzag scanning and run-length VLC entropy coding. However, new concepts and some key additional features differentiate H.264 from earlier standards.

First of all, the motion compensation model used in H.264 is more flexible and efficient than those in previous standards. For example, multiple reference frames may be used for prediction, allowing motion-compensated predictions to come from more than the most recent reference frame(s). Moreover, a much larger number of different motion compensation block sizes may be used for motion compensation on each macroblock (16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4, as illustrated in Figure 2.3), for the entire macroblock up to sixteen individual motion vectors (one per block of 4x4 pixels) could be used in motion estimation, substantially improving the motion estimation accuracy, especially in areas with fine motion details.

Higher motion-vector resolution is also specified in the motion prediction model. Sub-pixel value interpolation is used to provide more precise spatial accuracy at fractional positions. Currently, quarter-pixel precision is the default.

TABLE 3.1. OVERALL DECODING PERFORMANCE USING VARIOUS CONFIGURATIONS ON 3.06GHZ PENTIUM® 4

Format	Sequence		Frames/sec
MPEG-2	720x480	3Mb/s	315.6
		6Mb/s	238.9
		9Mb/s	202.7
	1920x1024	17Mb/s	48.4
H.264	352x288	0.4Mb/s	149.2
		1.5Mb/s	94.2
	720x480	1.5Mb/s	40.1
		5.6Mb/s	27.3

For motion compensation, we can summarize that the new methods in H.264 have provided a more precise model, which can yield a much higher perceptual quality for the decoded video sequences than MPEG-2 at the same bitrate.

Second, another difference is that the DCT transform was replaced by a DCT-like integer transform in the H.264 standard, as shown below:

$$Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

The transform has a significant computational advantage compared with DCT transforms. Also, the small size helps to reduce the blocking and ringing artifacts. The precise integer transform eliminates any mismatch between the encoder and the decoder as well.

Third, while in previous standards, a deblocking filter is optional and thus out of the motion compensation loop, in H.264, a deblocking filter is required within the motion compensation loop. The use of deblocking filter within the motion compensation loop not only reduces visual artifacts but also improves the final video coding efficiency.

Fourth, the standard also implements a more complex and efficient entropy coding method: the context-based adaptive binary arithmetic coding (CABAC). Instead of using a universal coding table, the probability model of this coding method is adaptive to the changing statistics of incoming data; therefore it can offer better coding efficiency. The normally used VLC scheme is still reserved in the system and serves as an alternative method.

III. HIGH LEVEL WORKLOAD ANALYSIS

This section describes the overall application behavior of optimized MPEG-2 and H.264 decoders. The MPEG-2 decoder used in the study is part of the Intel Media Processing Library [20], which was developed and highly optimized for Pentium® 4 processors by Intel Labs; the H.264 decoder is heavily optimized as well [21]. The software was analyzed using the Intel Tune™ Performance Analyzer [1][3][7] on an Intel® Pentium® 4 processor with a 533MHz system bus, 8 KB L1 data cache, 512 KB L2 shared instruction/data cache, 845 chipset and 512 MB of 333MHz DDR main memory. To investigate frequency scaling effects, we varied clock frequency of the Pentium® 4 processor between 1.6GHz and 3.06GHz; however, for most of the study, we ran at 3.06GHz.

A. MPEG-2 application behavior

The video sequences used in MPEG-2 study are 3Mb/s, 6Mb/s and 9Mb/s versions of DVD resolution (720x480) MPEG-2 sequence and a 1920x1024 high-definition (HD) MPEG-2 sequence (17 Mb/s); for H.264 we use 1.5Mb/s and 5.6Mb/s of a 720x480 sequence, as well as 0.4Mb/s and 1.5Mb/s CIF resolution sequences. Each was chosen as being representative of a larger set of video sequences that we have used in our studies.

As Table 3.1 suggests, the behavior of the MPEG decoder is highly dependent on the characteristics of the video stream being decoded. H.264 is more complicated than MPEG-2. The 1.5Mb/s DVD-resolution H.264 sequence takes about the same CPU time to decode as a high-definition MPEG-2 sequence.

Figure 3.1 compares the relative breakdown of the decoder performance for a 3Mb/s and 9Mb/s version of the same content. For the 3Mb/s case, motion compensation is the most expensive module, followed by the inverse DCT. DCT coefficient decoding and inverse quantization (block-level VLD) takes around 16% of total time, with the rest of the time distributed amongst high-level bitstream parsing (picture header processing, macroblock-level VLD, etc.), motion vector prediction, IDCT address calculations, and other decoding overhead.

When looking at the 9Mb/s stream, a different picture emerges. The absolute time spent on motion compensation is effectively unchanged; however, IDCT becomes a larger contributor (28% overall), as fewer blocks are skipped at this

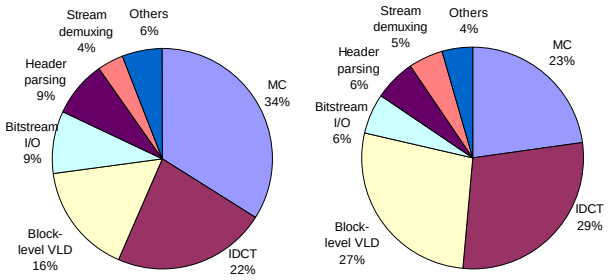

Figure 3.1. MPEG-2 decoding breakdown by time on 3.06GHz Pentium® 4 (720x480; left, 3Mb/s, and right, 9Mb/s).

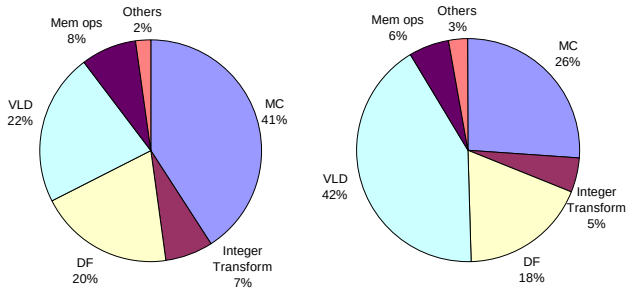

Figure 3.2. H.264 decoding breakdown by time on 3.06GHz Pentium® 4 (720x480; left, 1.5Mb/s, and right, 5.6Mb/s)

Table 4.1. Kernel characterization on 3.06GHz Pentium® 4.

	MPEG-2, 720x480, 9Mb/s			H.264, 720x480, 1.5Mb/s			
Kernels	MC	IDCT	VLD	MC	IDCT	DF	VLD
IPC	0.18	0.65	0.75	0.556	1.163	0.762	0.640
UPC	0.27	1.03	1.03	0.834	1.538	0.989	0.789
MMX/SSE/SSE-2 per 100 instructions	45	**91**	6	27	30	0	0
Branches per 1000 instructions	64	12	95	47	36	91	117
Mispredict. branches per 1000 cycles	1.2	0.3	**4.3**	1.0	0.2	**5.3**	**7.2**
L1 Hit-rate	70.2%	97.4%	95.5%	92.2%	91.5%	96.1%	95.5%
L2 Hit-rate	77.5%	90.0%	98.2%	90.9%	99.9%	73.5%	96.4%
Front-side bus utilization rate	**27.3%**	2.4%	3.4%	9.1%	2.7%	**14%**	7.4%

Table 4.2. MPEG-2 and H.264 frequency scaling for key modules, showing overall percentage of decode time and module speed-up.

		MPEG-2, 720x480, 9Mb/s					H.264, 720x480, 1.5Mb/s					
CPU		Overall		Key kernels			Overall		Key kernels			
Frequency	Scaling	FPS	Scaling	MC	IDCT	VLD	FPS	Scaling	MC	IDCT	DF	VLD
1.60GHz	1.0	120	1.0	1.0	1.0	1.0	23.1	1.0	1.0	1.0	1.0	1.0
1.87GHz	1.17	136.7	1.14	1.09	1.16	1.14	26.4	1.14	1.15	1.15	1.12	1.16
2.27GHz	1.42	160.6	1.34	1.18	1.41	1.37	31.1	1.34	1.34	1.39	1.30	1.41
2.67GHz	1.67	182.0	1.52	1.28	1.63	1.64	35.3	1.53	1.55	1.65	1.47	1.63
3.06GHz	1.92	202.7	1.69	1.34	1.84	1.88	40.1	1.72	1.73	1.88	1.63	1.87

bitrate, and VLD/IQ emerges to become a more prominent bottleneck, now taking 27% of the total decoding time.

B. H.264 application behavior

This section describes the overall application behavior of optimized H.264 decoder. Figure 3.2 compares the relative breakdown of the decoder performance for 1.5Mb/s and 5.6Mb/s versions of the same content.

For the 1.5Mb/s bitstream, motion compensation is the most time-consuming module, taking over 40% of the total CPU time. VLD takes about 22% of the total CPU time, whereas the DCT-like integer transform takes only 7%. The deblocking filter introduced in the H.264 decoder requires a large amount of computation as well, taking the remaining 20% of processing time.

For the 5.6Mb/s bitstream, even at 3.06GHz we can only decode 27 frames/s. The absolute time spent on motion compensation and inverse integer transform only slightly increase. Meanwhile, the proportion of time spent in the VLD kernel dramatically increases, taking now about 42% of the total decoding time.

Another interesting observation from the H.264 breakdown is importance of memory operations in the decoder. H.264 uses a 4x4 block size as its basic operation unit, which results in much larger buffers being required for motion vector and coefficient predictions. For each decoding frame, the H.264 decoder has to reset these buffers to zero, placing intensive demands on the memory subsystem. For example, for a high definition video sequence, the action of zeroing out these buffers takes nearly 20% of the total decoder time.

IV. KERNEL IMPLICATIONS FOR HIGH-PERFORMANCE SUPERSCALAR ARCHITECTURES

In this section, we address in greater detail some of the key kernels found in the MPEG-2 and H.264 decoding pipelines, paying particular attention to the implications for high-performance singled-threaded architectures, as are characterized by typical superscalar machines today [5]. We also discuss the trends we expect to observe as both decoding workloads and CPU micro-architectures continue to evolve over time.

Based on the application breakdowns of the previous section, the key kernels we discuss here are VLD, inverse transform (IDCT), MC, and deblocking filter (DF). The first three stages are common to MPEG-2 and H.264, albeit with significant differences in algorithms and complexity, but the last is a new feature in H.264.

The overall kernel characteristics for the H.264 and MPEG-2 on the 3.06GHz Pentium® 4 are shown in Table 4.1, and the per-module frequency scaling on Pentium® 4 processors is compared in Table 4.2. From the 1.60GHz to 3.06GHz scaling numbers given in Table 4.2, we can observe that both VLD and IDCT are entirely computation-bound for both MPEG-2 and H.264, showing performance scaling directly with CPU frequency. Motion compensation is memory intensive in MPEG-2, as reflected from the bus utilization rate from Table 4.1, showing little speed-up from the increase in clock frequency in Table 4.2. In comparison, H.264's motion compensation is not purely dependent on the memory system, as computations play a significant role as well. The deblocking filter is similarly memory intensive, its performance being co-impacted by computation and the memory system.

A. Variable-length decoding

As other parts of the video decoding pipeline, such as IDCT and motion compensation, have been accelerated with the SIMD-style extensions available in modern instruction sets, the serial operation of variable-length decoding (VLD) has assumed increasing importance as a key kernel in video processing.

VLD for both MPEG-2 and H.264 is characterized by substantial data dependency, limiting opportunities for instruction, data, and thread-level parallelism. Furthermore, both kernels require extensive bit-level manipulations to parse codewords from the input stream, an operation for which most

TABLE 4.3. VLD MICRO-ARCHITECTURE FOR ON 3.06GHZ PENTIUM-4

Statistics	MPEG-2 VLD	H.264 VLD	
		CABAC	NAL decoding
IPC	0.75	0.63	0.67
Branch mispredict. Rate	6%	8.7%	9.6%
Mispredict. branch/inst.	0.6%	1.25%	1.41%
% time lost to branch mispredictions	9.1%	15.8%	18.9%
L1 Hit-rate	95.5%	96.5%	94.2%
L2 Hit-rate	98.2%	99.3%	95.3%

general purpose architectures are ill suited.

Showing excellent scaling over different frequencies on Pentium® 4 processors in Table 4.2 indicates that the kernel is entirely computation bound.

1) MPEG-2

A breakdown of the MPEG-2 VLD kernel reveals that CPU time is divided fairly evenly between its three major components: bit stream input, Huffman table lookups, and inverse zigzag scan/inverse quantization.

A number of challenges arise in decoding variable-length symbols from an MPEG-2 bitstream efficiently. First, most implementations use a lookup table to decide which symbol is currently being examined. However, there is a trade-off between the size of the look-up table and the complexity of the lookup operation. The simplest lookup requires only a single table; however decoding a possibly 17-bit symbol with one table lookup requires the decoder to build a table of 2^{17} = *128K* entries, which is too large to fit into the first-level cache. Alternatively, since only a small subset of all of those entries correspond to valid Huffman codewords, the tables can be broken into several smaller pieces, so that all can fit into the L1 cache. In this case, however, it takes extra steps to decide which table to use. In our implementation, only three tables of total size 2Kbytes are used in VLD kernel.

Besides deciding which lookup table to use, VLD has a large number of conditional branches for every DCT coefficient. Basically, there are four questions to ask for each coefficient: (a) Are there enough bits remaining in the bit buffer? (b) Which lookup table should we use? (c) Is it end of block yet? (d) Is it an escape code (a special fixed-length code for rarely encountered symbols)?

Memory references throughout the kernel are basically limited to Huffman table lookups and reading of the bitstream, and typically hit L1 around 95% of the time; the L2 hit rate is around 98.2%. A comparison of two different approaches for reading variable-length bit strings in VLD suggests that the L1 misses in our implementation of the kernel result almost exclusively from reading of data from the bitstream rather than from the Huffman lookups. Meanwhile, reading from the bitstream is a purely sequential operation, and so is very well suited to the hardware prefetch mechanism in Pentium® 4; however, such data is prefetched only into L2.

Nonetheless, despite the good scaling on Pentium® 4 and the heavily scalar code, the IPC of VLD is poor; Table 4.3 shows the performance of VLD for the 9Mb/s MPEG-2 stream on the Pentium® 4 microarchitecture.

The first observation is that even though the Pentium® 4 processor uses a sophisticated branch predictor, there is still a relatively high number of mispredicted branches encountered during VLD. This is consistent with the intuition that many branches in variable-length decoding are inherently data-dependent and often essentially random. The Pentium® 4 processor uses a 20-stage pipeline, with a resulting large branch misprediction penalty. As future CPUs are expected to further increase the number of pipeline stages, branch mispredictions in VLD will become increasingly important limiters to performance in multimedia applications.

2) H.264

Unlike MPEG-2, context-based adaptive binary arithmetic coding (CABAC) has been employed for variable-length decoding in H.264, resulting in 5%~20% bitrate savings compared to the simpler Huffman entropy coding scheme used in MPEG-2. Although H.264 also supports Huffman coding, in our workload analysis, we prefer to use CABAC as our basic entropy-coding tool for its higher coding efficiency.

Similar to Huffman entropy decoding, CABAC is characterized by substantial data dependency. Whereas in MPEG-2 we can easily use a look-up table to facilitate the codeword parsing process, this approach is not feasible for CABAC in H.264.

In this new entropy decoding model, we find two kernels, i.e., arithmetic decoding and the codeword parser (NAL decoding in Table 4.3); a breakdown of time reveals that CPU time is divided approximately equally between them. Table 4.3 shows some micro-architectural metrics of CABAC. Mispredicted conditional branches are the major bottleneck in this kernel. NAL decoding also encounters the same problem. While MPEG-2 uses only one motion vector and limited prediction types for each 16x16 macroblock, H.264 allows up to 16 motion vectors to be used for more precise predictions. The result is that with more detailed classification information and the smaller block size in the H.264 coding model, the kernel needs more conditional checks to identify the corresponding coding information, such as, prediction type of each 4x4 block, and so on. Most of these vary on a block-by-block basis, and so cannot be predicted effectively by the branch predictor.

B. Inverse transform/inverse quantization

For block-based video coding standards, the inverse transform is mainly performed on an 8x8 or 4x4 block level immediately following VLD/IQ. The data is typically in L1, and thus we expect that the kernel is computationally intensive. The frequency scaling result confirms this intuition.

1) MPEG-2

IDCT has been frequently targeted for efficient implementation with SIMD-style instruction sets [8][9]. Indeed, as the kernel is completely computation-bound, it is a good candidate for speeding up with such approaches. However, despite the large amount of both data and

TABLE 4.4. MPEG-2 IDCT INSTRUCTION MIX

Event	Per instruction
64-bit MMX™ instructions retired	31.8%
128-bit MMX™ instructions retired	0.1%
Packed double-precision SSE inst. retired	44.4%
Packed single-precision SSE inst. retired	13.6%
Scalar instructions retired	10.1%

Table 4.5 MPEG-2 IDCT Pentium® 4 μop dispatches

Event	Per uop	
	MPEG-2	H.264
Port 0 ALU uops retired	19.8%	20%
Port 1 ALU uops retired	2.6%	11%
Port 1 slow ALU uops retired	0.5%	0.6%
Port 1 x87 and SIMD uops retired	58.4%	30%

instruction-level parallelism that can be exploited in this kernel, the table above reveals that with an IPC of 0.65 on the Pentium® 4 processor, IDCT actually makes less efficient use of machine resources than the data-dependent code of VLD.

An examination of the kernel shows that the code is dominated by general SSE operations (see Table 4.4), with interspersed register-to-register moves and stores; e.g., a sequence of `movaps`, `addps`, and `subps` is a typical recurring theme, corresponding to the well-known butterfly operation, surrounded by associated prescaling/multiply operations.

To explain the unexpectedly poor IPC of this kernel, we turn to the description in [5]. Pentium® 4 processors includes one full floating-point (FP)/SSE execution unit (port 1), as well as one additional FP/SSE execution unit that can handle moves and stores (port 0). We believe that the heavy MMX/SSE computational demands of the IDCT kernel lead to contention for these limited resources. Given the instruction mix shown in Table 4.4, an examination of micro-op dispatches reveals the expected bias, as shown in Table 4.5.

The throughput for most of the SSE single-precision arithmetic operations used in the IDCT is one instruction every two cycles on Pentium® 4, so given IDCT's IPC of 0.6, we believe there is little room for improvement in the IDCT implementation itself in current microarchitectures. From a hardware perspective, it appears that the inclusion of additional SSE execution units could substantially improve performance in this kernel. Alternatively, from a software perspective, since VLD uses predominantly scalar code and thus underutilizes the machine's FP/SSE execution units, we would expect to see an overall application speed-up if IDCT calculations were interleaved with VLD. Similarly, since IDCT makes little use of memory, we believe that motion compensation address calculations and prefetches could be effectively interleaved with IDCT computations.

2) H.264

Instead of the traditional DCT/IDCT used in previous standards, H.264 employs a 4x4 integer transform to transform spatial-domain signals into a frequency-domain representation and vice versa.

In [21], Zhou, Li, and Chen demonstrated a fast SIMD implementation of chained matrix multiplications. While it is suitable for SIMD-style instruction sets, the SIMD instructions comprise only 30% of all instructions in the kernel. The transform kernel in H.264 is based on 4x4 blocks, the size of which is smaller than MPEG-2. With inverse quantization, address calculations, and the data load/store operations used in the kernel, the number of scalar instructions is a larger proportion. Due to the more balanced and effective utilization of execution resources, this module has a better IPC than the MPEG-2 transform, Table 4.5 reveals the micro-op dispatches in the H.264 integer transform kernel, where the proportion of SSE-2 instructions is less than MPEG-2, alleviating the heavy computation burden on Port 1.

As previously illustrated, additional SSE execution units may significantly improve the whole performance of IDCT in MPEG-2. However, with the smaller block size and better balance in resource utilization in the integer transform, we expect that the H.264 kernel would benefit little from the additional SSE execution units. Similarly, from a software perspective, interleaving VLD and IDCT computations would see less benefit than the MPEG-2 case.

C. Motion compensation

Compared to the other modules in the MPEG decoding pipeline, motion compensation is memory intensive. For example, decoding of a 720x480 resolution video sequence has a working set size of (720×480 pixels/frame) × (1.5 bytes/pixel) × (3 frames) = 1.5 MBytes. This is far larger than the L2 caches on most of today's desktop machines. Given this, there are a number of possible factors that could impact the whole system performance:

Cache line size. Pentium® 4 processors use 128-byte L2 cache lines. When performing motion compensation on adjacent macroblocks with correlated motion vectors, we would expect fewer cache misses on Pentium® 4 than earlier architectures, since prior reads are more likely to have brought the required data into the L2 cache already.

Memory bandwidth. At a high level, motion compensation essentially reads two (or more) buffers, adds them, and writes them back to memory. With a 533MHz system bus, Pentium® 4 processors support much higher memory bandwidth than earlier systems (up to 4.26 GBytes/s).

Register size. Motion compensation defined in the MPEG standards uses integer arithmetic. As motion compensation in MPEG-2 is performed on a macroblock basis, i.e. 16x16 regions for progressive luma macroblocks, we would expect some benefit from 128-bit registers (SSE-2 integer instructions on Pentium® 4 processors) compared to 64-bit registers (SSE/MMX) for this standard. On the other hand, due to the use of smaller blocks for motion compensation in H.264, i.e., 4x4 or 8x8 block size, the benefits from wider registers would be expected to be less.

Hardware prefetch. Pentium® 4 processors have a hardware prefetcher, which was designed primarily in the

TABLE 4.6 MPEG-2 MC CHARACTERIZATION ON 3.06GHZ AND 1.6GHZ PENTIUM® 4(9MB/S, MPEG-2, 720X480)

3.06GHz Pentium® 4					
Average		Clocks/ MB (K)	L2 cache /Instr.	Bus util.	IPC
SSE (hardware prefetch)	Overall	77.5	0.009	27.4%	0.33
	Fwd	58.7	0.010	27.3%	0.29
	Bwd	44.0	0.008	25.0%	0.35
	Bi	318	0.008	28.8%	0.32
SSE-2 (hardware prefetch)	Overall	70.4	0.023	27.3%	0.19
	Fwd	53.9	0.024	27.3%	0.18
	Bwd	39.5	0.018	26.4%	0.23
	Bi	301	0.027	27.8%	0.18
SSE-2 (no hardware prefetch)	Overall	69.4	0.024	28.1%	0.19
	Fwd	52.0	0.024	28.5%	0.18
	Bwd	39.2	0.020	27.0%	0.21
	Bi	297	0.029	28.3%	0.18
1.6GHz Pentium® 4					
Average		Clocks/ MB (K)	L2 cache /Instr.	Bus util.	IPC
SSE (hardware prefetch)	Overall	54.9	0.008	20.8%	0.50
	Fwd	39.2	0.009	22.3%	0.49
	Bwd	32.6	0.008	19.7%	0.51
	Bi	231	0.008	21.3%	0.50
SSE-2 (hardware prefetch)	Overall	46.3	0.022	22.3%	0.30
	Fwd	33.0	0.022	23.5%	0.30
	Bwd	26.0	0.017	22.0%	0.34
	Bi	196	0.028	22.0%	0.26

context of applications that read data sequentially.

We first investigate these parameters in detail for MPEG-2, before comparing the newer H.264 MC kernel to its MPEG-2 predecessor.

1) MPEG-2

In order to understand the properties of the MPEG-2 motion compensation module, we have compared the behavior of two implementations: one using SSE/MMX and the other using SSE-2, where we turned on/off the hardware prefetcher to see its effectiveness in MC kernel. All of them run on Pentium® 4 processors. Fundamentally, we are interested in

- Is the kernel computation or memory bound?
- What is the impact of having wider registers, viz. 128-bit vs. 64-bit?
- How effective is the hardware prefetcher, particularly the linear predictor in the Pentium® 4 processor, in motion compensation?
- What is the impact of the large memory bandwidth on the Pentium® 4?
- What role does the large cache line size play in performance?
- What role does the macroblock coding decision play in performance? Examples of coding decisions to consider include the choice of prediction mode (forward, backward, bi-directionally-predicted).

Impact of prediction direction on temporal locality. According to Table 4.6, we observe that forward prediction consistently results in higher bus utilization than backward prediction. Backward prediction, which is allowed only in B-frames, always predicts from the most recently decoded reference picture, whereas forward prediction generally incurs a greater delay in terms of number of coded pictures from the reference frame.[1] Thus, in general, one would expect fewer L2 misses in backward prediction compared to forward prediction, and consequently fewer references to main memory. This is reflected in the relative performance of the MC module for the different prediction directions; backward prediction is on average 20-25% faster than forward prediction for our sequences, the difference becoming more pronounced at higher clock speeds.

Impact of register width. Comparing the SSE and SSE-2 implementations in Table 4.6, we see approximately 9% overall improvement when using SSE-2. At lower clock speeds, bi-directional prediction gains the most benefit, improving by 15% with wider registers. This is because this prediction mode involves more computation than forward or backward prediction. At higher clock speeds, the benefit is reduced as memory effects begin to dominate.

Impact of hardware prefetch. The Pentium® 4 introduced a hardware prefetcher, some of the operating characteristics of which are described in [6]. The hardware prefetcher attempts to stay 256 bytes beyond the current data access location, and follows only one stream per 4K page. Since for a raster scan memory layout, a macroblock typically crosses approximately three pages[2] at standard definition (720x480) resolution, we would assume that hardware prefetch would be ineffective at hiding memory latency in motion compensation, recognizing and prefetch at best only 3/16ths of all accesses. Indeed, since the hardware prefetcher in the Pentium® 4 is designed to handle application scenarios where adjacent data is read consecutively, it would be somewhat surprising if we saw a benefit in motion compensation, in which, while predictable in software, memory access patterns are more irregular. We also suspect that large cache lines would tend to reduce the effectiveness of hardware prefetch in this application, as a cache line size of 128 bytes corresponds to 8 macroblocks. For many sequences, motion vector correlation decreases substantially over this number of macroblocks, so read-ahead prefetching may not necessarily provide the expected gains compared to other video workloads. Table 4.6 confirms this intuition; disabling the hardware prefetch, the kernel runs in almost the same amount of time as with the hardware prefetch enabled, indicating that the hardware prefetch is of little benefit for this kernel.

The kernel is not well-suited to the current hardware prefetch mechanism, which assumes sequential accesses.

[1] Consider a standard GOP structure (IBBPBBP…). Ignoring edge effects at the beginning of the sequence, forward prediction predicts from the third-most recently decoded frame (in P-frames), or either the fourth- or fifth-most recently decoded frame (in B-frames). In contrast, backward prediction always predicts from either the first- or second-most recently decoded frame.

[2] 720 bytes/row × 16 rows / 4 KB/page.

TABLE 4.7 MPEG-2 BUS UTILIZATION AND IPC VS. FREQUENCY ON PENTIUM® 4.

CPU frequency	SSE		SSE-2	
	IPC	Bus utilization	IPC	Bus utilization
1.60GHz	0.51	0.21	0.29	0.22
1.87GHz	0.42	0.22	0.26	0.23
2.27GHz	0.41	0.24	0.23	0.24
2.67GHz	0.36	0.25	0.21	0.27
3.06GHz	0.33	0.27	0.19	0.27

TABLE 4.8 MPEG-2 MC UOPS DISPATCHES ON PENTIUM® 4

Event	Percentage
Port 0 ALU uops	39.6%
Port 1 ALU uops	15.4%
Port 1 slow ALU uops	3.3%
Port 1 x87 and SIMD instructions	20.4%
X87 and SIMD register and memory moves	21.3%

However, it may be a good candidate for future hardware prefetcher [18]. This is a question for further investigation.

Impact of CPU frequency. Table 4.2 shows the relative performance scaling of motion compensation vs. CPU frequency. We can see some speed-up with the increase of CPU frequency, indicating that the kernel is still partially computation-bound even when the CPU frequency is as high as 3.06GHz. However, the acceleration of speed-up decreases gradually and the amount of time spent on this kernel becomes relatively stable as we further increase the CPU frequency, which indicates that the kernel is almost entirely memory-bound when the CPU frequency is high enough. At that time, most of the CPU time will be spent on waiting for the memory accesses, while computations can be accomplished in the vacant interval time.

Table 4.7 confirms the intuition that the amount of time spent using the bus should increase with clock frequency. In these simulations, only CPU frequency was changed on a single machine, so the front-side bus clock frequency was fixed at 533MHz. Correspondingly, as table 4.7 shows, IPC decreases as memory latency begins to dominate.

Impact of execution resources. As the kernel is still partially compute-bound at today's clock frequencies, we examine the breakdown of μops retired in the kernel in Table 4.8. Kernel processing is divided between address calculations and loop overhead (scalar code, approximately 58% of μops retired), motion compensation arithmetic itself (i.e. addition of prediction error and interpolation, approximately 20% of μops retired), and loads and stores (the majority of the remaining 21% of μops).

Impact of memory bandwidth/cache line size. From Table 4.1, motion compensation needs 533MB/sec × 8 × 27.3% = 1164 MB/sec in bandwidth on a 3.06GHz Pentium® 4 system. Higher bandwidth in the Pentium® 4 processor clearly contributes to the performance improvement over earlier systems. We probably would not run out of bandwidth until the CPU is running at 7~8GHz.

Our MPEG decoder interleaves memory accesses with computation. In forward- or backward-predicted macroblocks, a row of pixels is loaded from memory, and added to the current prediction error. In bi-directionally-predicted macroblocks, a row of pixels is loaded from both reference frames, and added to the current prediction error. With 128-byte cache lines, approximately 90% of the time the load will reference a single cache line. This means that for each memory transaction in motion compensation, generally at most one or two cachelines worth of data will be returned from memory, for unidirectional or bi-directional prediction respectively. Based on this, one might conclude that, considered alone, the increased memory bandwidth available on Pentium® 4 does not provide a direct gain in motion compensation.

However, since motion compensation is an operation that is repeated for many macroblocks in a frame, typically referencing adjacent areas in reference frames due to correlation between motion vectors, we believe that the larger data transfer latency is outweighed by the improved cache hit rates associated with subsequent requests. The evidence thus suggests that the combination of the increased bandwidth and larger cache line size is the biggest contributor to the improvement of motion compensation on the Pentium® 4.

2) H.264

In MPEG-2 decoding, several characteristics have been demonstrated in motion compensation, where wide registers, increased memory bandwidth, and large cache line sizes make great contributions to the whole kernel performance.

While these are not unexpected, different characteristics have been identified in the H.264 decoder. First, H.264 motion compensation scales much better with CPU frequency, with bus utilization around 8%~10% (CIF~DVD resolution format), 2/3 less than MPEG-2. Second, there is less opportunity for data-level parallelism, due to the use of smaller block sizes and more complex coding decisions.

In the motion compensation kernel there are two underlying differences between H.264 and former video coding standards: the use of 1/4 pixel motion compensation, and spatial interpolation for luma/chroma component. Thus, MC in H.264 requires more computation than previous older standards, which used only simple integer or half-pixel interpolation.

Whereas in MPEG-2, the MC kernel can use the PAVGB/PAVGW instruction to perform interpolation, H.264 uses a 6-tap filter to precisely interpolate 1/4-pixel values. While our implementation of 1/4-pixel motion compensation uses the SSE-2 instruction PMADDWD for the filter operations, the H.264 MC kernel still requires more complex manipulations than MPEG-2. Moreover, some performance penalty is introduced by the 4x4 block size used in MC. Although a great deal of operations are based on 8x8 blocks, there still exist 4x4 interpolations that are less effectively parallelized. In this case, wide SIMD registers can only partially be utilized.

Due to the heavy computational burden involved in the MC

TABLE 4.9 H.264 MC CHARACTERIZATIONS ON PENTIUM® 4

Micro-Arch.	Hardware prefetch on	Hardware prefetch off
L2 misses/Instructions	0.31%	0.32%
Bus utilization	9.0%	9.1%
IPC	0.56	0.55
Clocks (billions)	3.06	2.94

TABLE 4.10 H.264 MC CHARACTERIZATIONS WITH DIFFERENT REFERENCE FRAME ON PENTIUM® 4

	H.264, 720x480	
Micro-Arch.	1.5Mb/s 1 Ref frame	1.4Mb/s 5 Ref frame
IPC	0.56	0.50
Branches/ 1000 Instr.	45	45
Branch Mis-predict. Rate	3.5%	3.9%
L1 misses/ 1000 Instr.	32	34
L2 misses/ 1000 Instr.	2.9	4.3
Bus utilization rate	9.0%	7.8%

TABLE 4.11 H.264 DEBLOCKING FILTER KERNEL ON PENTIUM® 4

Micro-Arch.	Strength Calculation	Block Filtering
Clockticks (%)	7.5%	14.4%
IPC	0.642	1.11
Branches/ 1000 Instr.	110	132
Branch Mis-predict. rate	5.7%	5.4%
L1 miss rate	4.7%	1.4%
L2 miss rate	25.0%	23.4%

kernel, the memory subsystem in H.264 decoding faces less pressure than in MPEG-2. The bus utilization rate is only 8%~10% in the kernel. This corresponds to 4264MB/sec x 9.1% = 388MB/sec in bandwidth, indicating that memory bandwidth for this kernel is not a major issue currently; we expect this kernel would not run out of bandwidth until the CPU is running at 8~10GHz. With the kernel's increased computational demands, and the similar memory access patterns as in MPEG, Table 4.9 indicates that hardware prefetching is of little benefit in H.264 MC.

As previously illustrated, multi-frame prediction is a major coding gain in H.264 codec. One might expect that this would place great demands on the memory subsystem. For example, with five reference image frames at DVD resolution, the working set size is 2.5 MB, far exceeding L2 capacity. In fact, although L2 misses increase (to about one every 230 instructions), bus utilization actually decreases, and IPC remains relatively unaffected.

D. Deblocking Filtering

Deblocking filtering is found only in H.264. This section of the pipeline contains two kernels: block strength calculation and block filtering, the relative times of which are highly dependent on the inherent characteristics of the input sequence.

The frequency scaling result of Table 4.2 shows that the deblocking filter kernel has lower scaling than expected. The reason for this is evident in Table 4.11, which shows a large number of L2 cache misses during strength calculation. This is caused by the working set for this kernel exceeding current cache sizes. For a 720x480 resolution sequence, taking motion information and at least 3 frames into consideration, the minimum working set size is 2.4Mbytes[3], far larger than the L2 cache size on the 3.06GHz Pentium® 4 processor. Therefore, in the deblocking filter kernel, after motion compensation, the motion vector information required by the kernel has already been thrashed out of the L2 cache.

Block filtering shows good IPC, due to heavy scalar instruction use. SIMD instructions are not easily used in this kernel due to a couple of reasons: a large number of conditional operations are required on adjacent pixels and the smaller block size encumbers use of SIMD operations. The first problem could be solved according to the probability model used in the conditioning operations; examining the statistics of the neighboring pixels distribution, we conclude that the condition operator has greater than 80% probability of being true, and can thus use SIMD instructions such as PCMPGTW to optimize this kernel. The other problem argues for new approaches to explicitly handling data-level parallelism.

V. SIMULTANEOUS MULTI-THREADING

In general, multimedia applications exhibit not only data-level (DLP) and instruction-level parallelism (ILP), but also the possibility for substantial thread-level parallelism (TLP). The decoder can divide the picture data into parts and use multiple threads, each decoding part of the picture in parallel. Such workloads are good candidates for speeding up on a number of different multi-threaded architectures. This section discusses the performance of a threaded MPEG decoder on several parallel architectures.

Intel Corporation recently introduced Hyper-Threading Technology, which supports two threads simultaneously on the same physical processor, in Intel® Xeon™ processors and 3.06GHz Pentium® 4 processors [14]. It is one implementation of an SMT architecture [17]. The motivation for SMT is that the performance of many programs is often limited by a single execution resource, while other resources tend to be under-utilized. If the CPU can interleave the execution of different tasks, for example, interleaving calculations with memory operations, then more execution resources can be utilized at the same time. To enable this, Hyper-Threading Technology supports two logical processors on a single physical processor, so that two different threads can run simultaneously, yielding more efficient use of machine resources. In today's Hyper-Threading Technology, only a small set of the microarchitecture state is duplicated, while the front-end logic, execution units, out-of-order retirement engine, and memory hierarchy are shared. Thus, compared to processors without Hyper-Threading Technology, the die-size is increased by less than 5% [14].

[3] Considering only frame buffer and motion vector information: (720 x 480pixels/frame)x(1.5bytes/pixel)x(3frames)+(720/4)x(480/4)x2x5x4 bytes = 2.42 Mbytes

TABLE 5.1 COMPARISON OF MPEG-2 DECODER ON SINGLE-THREADED, SMT PROCESSOR, AND DUAL-PROCESSOR SYSTEMS.

Event	Two copies of decoders (3.06GHz Pentium®4)				2-threaded MPEG-2 decoder, 720x480, 9Mb/s (2.0GHz Xeon™)		
	MPEG-2, 720x480, 9Mb/s		H.264, 720x480, 1.5Mb/sec				
	Single-thread	Hyper-threading	Single-thread	Hyper-threading	Single-thread	Hyper-threading	Dual-processors
Clockticks (millions)	22,587	17,996	14,949	13,265	11,794	10,779	7,452
Instructions retired (millions)	13,984	13,983	9,158	9,155	7,091	7,792	7,713
Uops retired (millions)	19,449	19,414	12,490	12,496	10,292	11,505	11,505
MMX/SIMD instr. (millions)	4,575	4,378	1,218	1,218	2,288	2,288	2,288
IPC	0.62	0.78	0.61	0.69	0.6	0.72	1.04
UPC	0.86	1.08	0.84	0.94	0.87	1.07	1.54
L1 cache misses (millions)	198	357	200	329	107	171	131
Bus utilization rate	10.6%	15.0%	11.6%	16%	9.60%	6.0%	10.6%

As mentioned earlier, our MPEG-2 and H.264 decoders are optimized for the Intel® Pentium® 4 processor. Nonetheless, due to the inherently sequential constitution of the algorithms, most of the modules in these well-optimized workloads cannot fully utilize all the execution units available in the microprocessor. For example, while motion compensation is memory-intensive, the VLD and IDCT modules are limited by computation. This makes MPEG decoding a good candidate for running on SMT machines.

Table 5.1 shows the comparison of our "multi-threaded" MPEG-2 and H.264 decoders on a single-threaded processor, on a single processor with Hyper-Threading Technology, and a dual-processor system. In general, on SMT processors, kernel characteristics become less important than the overall application characteristics, because multiple kernels are typically executing concurrently at a given time. Hence, it is hard to breakdown the workload characteristics in individual modules. Rather, we consider here the whole application as a whole. In the first four columns, the workload is not threaded. Instead, we run two copies of the same decoder simultaneously. The copies are not synchronized, so there is no synchronization overhead and interleaving of different modules happens naturally. Under these ideal conditions, the speed-up is a benchmark for the best-case gain achievable in a threaded decoder.

The first four columns show that we get a very good speed-up with two copies of the decoder running simultaneously on Hyper-Threading processors. UPC increases from 0.86 to 1.08 in MPEG-2 decoder, indicating 26% more efficient resource utilization—an impressive figure given the 5% die-size increase to support Hyper-Threading Technology [14].

As seen in the first four columns of Table 5.1, on Hyper-Threading Technology, UPC increases from Hyper-Threading Technology in two copies of the H.264 decoder are substantially lower than those in the MPEG-2 decoder (12% vs. 26%). This is a result of the fact that modules in the MPEG-2 decoder have more distinct bottlenecks than those in H.264 decoder, as shown in Table 4.1. For example, the MC module in MPEG-2 is verging on being memory-bound, while the MC module in H.264 is both computation- and memory-bound. Moreover, in the H.264 decoder, many modules use only scalar instructions, for which they contend for the same integer computation unit. Therefore, two copies of the H.264 decoder see less speed-up than those of the MPEG-2 decoder on Hyper-Threading Technology.

In addition to running two unsynchronized copies of the same decoder, the last three columns of Table 5.1 show a comparison of a more realistic multi-threaded workload. A single MPEG-2 decoder partitions its work for two threads. Pictures can be divided into slices of macroblocks, and each thread can be assigned to decode some slices of macroblocks, e.g., as shown in Figure 5.1. In this case, the workload has more fine-grained scheduling with synchronization overheads than the two copies of decoders. The workload also has a non-parallelizable portion, resulting in only 1.58x speed-up on a dual-processor system and 1.09x speed-ups on Hyper-Threading Technology.

While the shared cache may be a drawback for some applications running on processors with Hyper-Threading Technology, it can also provide better cache locality between the two logical processors for other applications. For example, one logical processor can be used to prefetch data into the shared caches to reduce a substantial amount of the memory latency of the application in the other logical processors [19]. Similarly, in multimedia applications, a shared cache can be exploited to reduce the impact of cache misses by scheduling threads to prefetch data for each other [2].

Our results also demonstrate that an MPEG decoder can benefit from cache sharing. Figure 5.1 illustrates two different multi-threading schemes used in our video decoder.

1. Static partitioning: In this method, one thread is statically assigned the first half of the picture, while another thread is assigned the other half of the picture (as shown in Figure 5.1(a)). Assuming that the complexity of the first half and second half are similar, these two threads will finish the task at roughly the same time. However, some areas of the picture may be easier to decode than others. This may lead to one thread being idle while the other thread is still busy.
2. Dynamic partitioning: In this method, slices are dispatched dynamically so as to achieve good load balance. A new slice is assigned to a thread when the

thread has finished its previously assigned slice. In this case, we don't know which slices will be assigned to which thread. Instead, the assignment depends on the complexity of the slices assigned. As a result, one thread may decode a larger portion of the picture than the other if its assignments are easier than the other thread's. The execution time difference between two threads in the worst case is the decoding time of the last slice.

The principle advantage to static task partitioning is its high cache locality in dual-processor systems. Figure 5.2 illustrates the cache locality in multiple frames of video. During motion compensation, the decoder uses part of the previous picture, the referenced part of which is roughly co-located in the previous reference frame, to reconstruct the current frame. It is faster to decode the picture when the co-located part of the picture is still in the cache. In the case of a dual-processor system, each thread is running on its own processor, each with its own cache. If the co-located part of the picture in the previous frame is decoded by the same thread, it is more likely that the local cache will have the pictures that we just decoded. This property makes static partitioning attractive.

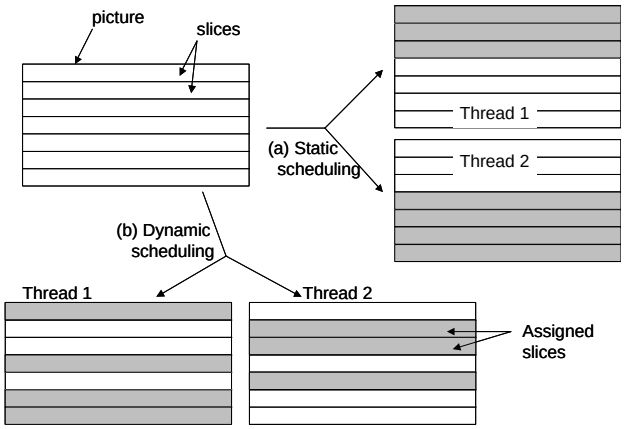

Figure 5.1. Two slice-based task partitioning schemes between two threads: (a) static scheduling and (b) dynamic scheduling.

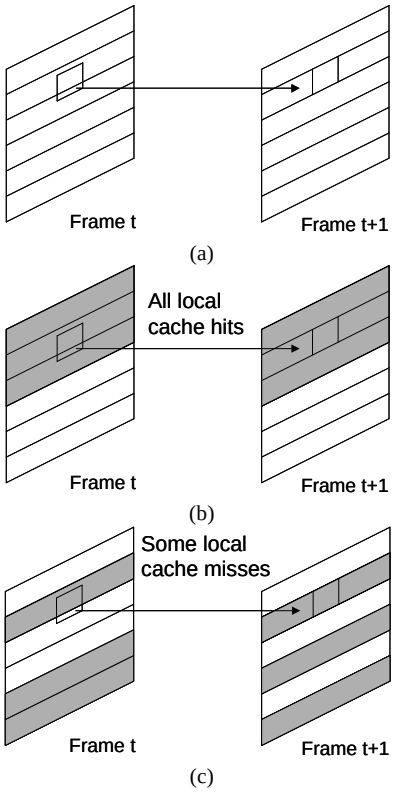

Figure 5.2. Cache localities, during (a) motion compensation, in (b) static partitioning, and in (c) dynamic partitioning.

While the foremost advantage of the dynamic scheduling scheme is its good load balance, there may be a drawback to dynamic partitioning in terms of cache locality on dual-processor systems. As we mentioned earlier, during motion compensation, we use part of the co-located previous pictures. In the case of a dual-processor system, each thread is running on its own processor (and cache). Since we dynamically assign slices to different threads, it is more likely that the co-located portion of the previous picture may not be in the local cache, as shown in Figure 5.2(c). Thus, dynamic partitioning incurs 45% more bus transactions on dual-processor systems.

On the other hand, on a processor with Hyper-Threading Technology, the FSB traffic is within 1% range from the static partitioning method to the dynamic partitioning method. This is because the shared cache on Hyper-Threading Technology can provide better cache locality between the two logical processors. Because of efficient cache sharing and better load balancing, the dynamic scheduling has 4% better performance than the static scheduling on Hyper-Threading Technology.

VI. CONCLUSIONS

In this paper, we have studied the performance of several highly optimized MPEG and H.264 decoders on state-of-the-art micro-architectures, describing the key characteristics of the workload on these systems, and have addressed the impact that various micro-architectural features can be expected to have on the workload.

To summarize the workload characteristics, we conclude the following:

- Application performance. H.264 decoding is much more demanding than MPEG-2; a 1.5Mb/s DVD-resolution main profile stream is almost as complex as a 17 Mb/s 1920x1024, HDTV MPEG-2 sequence.
- VLD/CABAC. VLD/CABAC is heavily data-dependent, uses predominantly scalar code, and is entirely computation-bound, scaling directly with frequency on Pentium® 4.
- IDCT/Inverse Integer Transform. IDCT and Inverse Integer Transform can be fully optimized with SIMD instructions, and are entirely compute-bound, also scaling directly with frequency on Pentium® 4.
- Motion Compensation. MPEG-2 motion compensation is mostly memory bound. In contrast, H.264 motion compensation is more computationally demanding. Neither version of the kernel is well-suited to current hardware prefetch mechanisms that assume sequential accesses.

- Deblocking Filter. Unlike previous video coding standards, a deblocking filter becomes a necessary component in H.264 decoders. The kernel is memory- and branch-intensive.

From a microarchitectural perspective, we conclude the following for the workload:

- Branch prediction. The majority of conditional branches in MPEG and H.264 decoding occur during VLD/CABAC, many of which are inherently data-dependent and difficult to predict. Branch mispredictions in other key kernels result in very little lost CPU time. Improving the sophistication of branch predictors can be expected to provide relatively little improvement in MPEG decoding performance.
- Memory subsystem. The performance of the memory subsystem has a dramatic impact on motion compensation in MPEG-2. Motion compensation seems to benefit most from the combination of the higher bandwidth and longer L2 cachelines used by the Netburst™ microarchitecture. On the other hand, the sequential hardware prefetcher of Pentium® 4 appears to provide little benefit in motion compensation.
- Wider registers. Increasing the size of SIMD registers from 64-bits to 128-bits results in 9% improvement in MPEG-2 motion compensation. However, these registers are underutilized in H.264, because most prediction in this standard is limited to regions at most 4 and 8 pixels wide.
- Execution resources. The performance of IDCT in MPEG-2, and to a lesser extent motion compensation, is bounded by the number of SIMD execution units available on the Pentium® 4.

Extending the current work, there are a number of possible issues to address, e.g., the characterization of MPEG encoders, and the impact of multi-threading. There should be more work to study future multi-threaded media workloads and their implications to future multi-threaded microarchitectures.

ACKNOWLEDGMENT

We wish to acknowledge the exceptional efforts of Intel Nizhny Novgorod Lab in developing the MPEG-2 decoders used in this study, especially Valery Kuriakin, Sergey Zheltov, Roman Belenov, and Alexander Knyazev. We also would like to thank Steven Ge and Justin Song for their assistance in some performance measurements. Finally, we thank Jih-Kwon Peir, Anwar Rohillah, and Ronak Singhal for the constructive comments that allowed us to improve our manuscript.

REFERENCES

[1] M. Atkins and R. Subramanism, "PC Software Performance Tuning," *IEEE Computer*, vol. 29, no. 9, pp. 47-54, Aug. 1996.

[2] Y.-K. Chen, E. Debes, R. Lienhart, M. Holliman, and M. Yeung, "Evaluating Performance of Multimedia Application on Simultaneous Multi-Threading," in Proc. of Int'l Conf. on Parallel and Distributed Systems, pp. 529-534, Dec. 2002.

[3] R. Coelho and M. Hawash, *DirectX®, RDX, RSX, and MMX™ Technology: a Jumpstart Guide to High Performance APIs*, MA: Addison-Wesley, April 1998.

[4] B. G. Haskell, A. Puri, and A. N.Netravali, *Digital Video: An Introduction to MPEG-2*, MA: Kluwer, 1997.

[5] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium® 4 Processor," *Intel Technology Journal*, Q1 2001.

[6] Intel Corp., *Intel® Pentium® 4 Processor Optimization Reference Manual*, Order number: 248966, (available on-line: http://developer.intel.com/design/Pentium4/manuals/248966.htm).

[7] Intel Corp., *Intel® Vtune™ Performance Analyzer*, (available on-line: http://developer.intel.com/software/products/vtune/).

[8] Intel Corp., "A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Streaming SIMD Extensions and MMX Instructions," Intel Application Notes AP-922 (available on-line: http://developer.intel.com/vtune/cbts/strmsimd/appnotes/ap922/ap922.pdf), Apr. 1999.

[9] Intel Corp., "Using MMX Instructions in a Fast iDCT Algorithm for MPEG Decoding," Intel Application Notes AP-528, (available on-line: http://developer.intel.com/software/idap/resources/technical_collateral/mmx/AP528.HTM).

[10] International Standard Organization, " Information Technology--Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1,5 Mbit/s---Part 2: Video," ISO/IEC 11172-2.

[11] International Standard Organization, "Information Technology--Generic Coding of Moving Pictures and Associated Audio Information---Part 2: Video," ISO/IEC 13818-2.

[12] International Standard Organization, "Information Technology–Coding of Audio-Visual Objects, Part 2---Visual," ISO/IEC 14496-2.

[13] ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC, Document JVT-D157, 4th Meeting: Klagenfurt, Austria, July 2002.

[14] D. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-Threading Technology Microarchitecture and Architecture," *Intel Technology Journal*, Vol. 6, Q1, 2002.

[15] J. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, *MPEG Video Compression Standard*, NY: Chapman & Hall, 1997.

[16] T. Sikora, "MPEG Digital Video-Coding Standards," *IEEE Signal Processing Magazine*, vol. 14, no. 5, pp. 82—100, Sept. 1997.

[17] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," in Proc. of Int'l Symp. on Computer Architecture, pp. 392--403, June 1995.

[18] S. P. Vanderwiel and D. J. Lilja, "Data Prefetch Mechanisms," *ACM Computing Surveys*, Vol. 32, No. 2, pp. 174-199, June 2000.

[19] H. Wang, P. Wang, R. D. Weldon, S. Ettinger, H. Saito, M. Girkar, S. Liao, and J. Shen,, "Speculative Precomputation: Exploring the Use of Multithreading Technology for Latency," *Intel Technology Journal*, vol. 6, no. 1, pp. 22-35, Feb. 2002.

[20] M. M. Yeung, "MPL: MPEG Processing Library---Tools and Advanced Technology for Video-Centric Applications," Intel Developer Forum, (Palm Spring, CA), Sept. 1999.

[21] X. Zhou, E. Q. Li and Y.-K. Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions", in *Proc. of SPIE Conf. on Image and Video Communications and Processing*, vol. 5022, Jan. 2003.