Real-Time Software MPEG Video Decoder on Multimedia-Enhanced PA 7100LC Processors

With a combination of software and hardware optimizations, including the availability of PA-RISC multimedia instructions, a software video player running on a low-end workstation is able to play MPEG compressed video at 30 frames/s.

by Ruby B. Lee, John P. Beck, Joel Lamb, and Kenneth E. Severson

Traditionally, computers have improved productivity by helping people compute faster and more accurately. Today, computers can further improve productivity by helping people communicate better and more naturally. Towards this end, at Hewlett-Packard we have looked for more natural ways to integrate communication power into our desktop machines, which would allow a user to access distributed information more easily and communicate with other users more readily.

We felt that adding audio, images, and video information would enrich the information media of text and graphics normally available on desktop computers such as workstations and personal computers. However, for such enriched multimedia communications to be useful, it must be fully integrated into the user's normal working environment. Hence, as the technology matured we decided to integrate increasing levels of multimedia support into both the user interface and the basic hardware platform.

In terms of user interface, we integrated a panel of multimedia icons into the HP VUE standard graphical user interface, which comes with all HP workstations. These multimedia icons are part of the HP MPower product.[1] HP MPower enables a workstation user to receive and send faxes, share printers, access and manipulate images, hear and send voice and CD-quality stereo audio, send and receive multimedia email, share an X window or an electronic whiteboard with other distributed users, and capture and play back video sequences. The HP MPower software is based on a client/server model, in which one server can service around 20 clients, which can be workstations or X terminals.

In terms of hardware platforms, we integrated successive levels of multimedia support into the baseline PA-RISC workstations.[2,3,4] First, we integrated support for all the popular image formats such as JPEG (Joint Photographic Experts Group)† compressed images.[5] Then, we added hardware and software support for audio, starting with 8-kHz voice-quality audio, followed by support for numerous audio formats including A-law, μ-law, and 16-bit linear mode, with up to 48-kHz mono and stereo. This allowed high-fidelity, 44.1-kHz stereo, 16-bit CD-quality audio to be recorded, manipulated, and played back on HP workstations. At the same time, we supported uncompressed video capture and playback.

In January 1994, HP introduced HP MPower 2.0 and the entry-level enterprise workstation, the HP 9000 Model 712, which is based on the multimedia-enhanced PA-RISC processor known as the PA 7100LC.[6,7,8] The video player integrated in the MPower 2.0 product is the first product that achieves real-time MPEG-1 (Moving Picture Experts Group)[9] video decompression via software running on a general-purpose processor. Typically real-time MPEG-1 decompression is achieved via special-purpose chips or boards. Previous attempts at software MPEG-1 decompression did not attain real-time rates.[10] The fact that this is achieved by the low-end Model 712 workstation is significant.

In this paper, we discuss the support of MPEG-compressed video as a new (video) data type. In particular, we discuss the technology that enables the video player integrated into the HP MPower 2.0 product to play back MPEG-compressed video at real-time rates of up to 30 frames per second.

Digital Video Standards

We decided to focus on the MPEG digital video format because it is an ISO (International Standards Organization) standard, and it gives the highest video fidelity at a given compression ratio of any of the formats that we evaluated. MPEG also has broad support from the consumer electronics, telecommunications, cable, and computer industries. The high compression capability of MPEG translates into lower storage costs and less bandwidth needed for transmitting video on the network. These characteristics make MPEG an ideal format for addressing the need for detail in the video used in technical workstation markets and computer-based training in commercial workstation markets.

MPEG is one of several algorithmically related standards shown in Fig. 1. All of these digital video compression standards use the discrete cosine transform (DCT) as a fundamental component of the algorithm. Alternatives to discrete cosine-based algorithms that we looked at include vector quantization, fractals, and wavelets. Vector quantization

† JPEG is an international digital image compression standard for continuous-tone (multilevel) still images (grayscale and color).

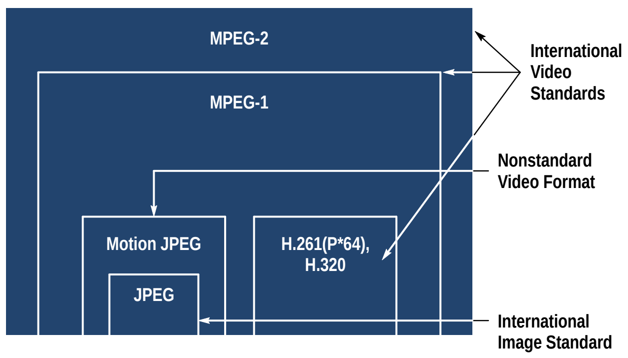

Fig. 1. Digital video standards based on the discrete cosine transform.

algorithms are popular on older computer architectures because they require less computing power to decompress, but this advantage is offset by poorer image quality at low bandwidth (high compression) compared to MPEG for practical vector quantization methods. Algorithms based on wavelet and fractal technology have the potential to deliver video fidelity comparable to MPEG, but there is presently a lack of industry consensus on standardization, a key requirement for our use.

Another advantage of a high-performance implementation of MPEG is the ability to leverage the improvements to the other DCT-based algorithms. Although the relationships shown in Fig. 1 do not represent a true hierarchy of algorithms is useful for illustrating increased complexity as one moves from JPEG to MPEG-2, or from H.261 to MPEG-2.

All of these formats have much in common, such as the use of the DCT for encoding. The visual fidelity of the algorithms was the key selection criterion and not ease of implementation or performance on existing hardware.

Although JPEG supports both lossy and lossless compression, the term JPEG is typically associated with the lossy specification.† The primary goal of JPEG is to achieve high compression of photographic images with little perceived loss of image fidelity. Although it is not an ISO standard, by convention, a sequence of JPEG lossy images to create a digital video sequence is called motion JPEG, or MJPEG.

H.261 is a digital video standard from the telecommunications standards body ITU-TSS (formerly known as CCITT). H.261 is one of a suite of conferencing standards that make up the umbrella H.320 specification. H.261 is often referred to as P*64 (where P is an integer) because it was designed to fit into multiples of 64 kbits/s bandwidth. The first frame (image) of an H.261 sequence is for all practical purposes a highly compressed lossy JPEG image. Subsequent frames are built from image fragments (blocks) that are either JPEG-like or are differences from the image fragments in previous frames. Most video sequences have high frame-to-frame coherence. This is especially true for video conferencing. Because the encoding of the movement of a piece of an image requires less data than an equivalent JPEG fragment, H.261 achieves higher visual fidelity for a given bandwidth than does motion JPEG. Since the encoding of the differences is always based on the previous frames, the technique is called *forward differencing.*

† In lossless compression, decompressed data is identical to the original image data. In lossy compression, decompressed data is a good approximation of the original image data.

The MPEG-1 specification goes even further than H.261 in allowing sophisticated techniques to achieve high fidelity with fewer bits. In addition to forward differencing, MPEG-1 allows backward differencing (which relies on information in a future frame) and averaging of image fragments. (Forward and backward differencing are described in more detail in the next section.) MPEG-1 achieves quality comparable to a professionally reproduced VHS videotape even at a single-speed CD-ROM data rate (1.5 Mbits/s).[9,11] MPEG-1 also specifies encodings for high–fidelity audio synchronized with the video.

MPEG-2 contains additional specifications and is a superset of MPEG-1. The new features in MPEG-2 are targeted at broadcast television requirements, such as support for frame interleaving similar to analog broadcast techniques. With widespread deployment of MPEG-2, the digital revolution for video may be comparable to the digital audio revolution of the last decade.

The approximate bandwidths required to achieve a level of subjective visual fidelity for motion JPEG, H.261, MPEG-1, and MPEG-2 are shown in Fig. 2. Motion JPEG will primarily be used for cases in which accurate frame editing is important such as video editing. H.261 will be used primarily for video conferencing, but it also has potential for use in video mail. MPEG-1 and MPEG-2 will be used for publishing, where fidelity expectations have been set by consumer analog video tapes, computer-based training, games, movies on CD, and video on demand.

MPEG Compression

MPEG has two classes of frames: intracoded and nonintracoded frames (see Fig. 3). Intracoded frames, also called *I-frames,* are compressed by reducing spatial redundancy within the frame itself. I-frames do not depend on comparisons with past or "reference" frames. They use JPEG-type compression for still images.[5]

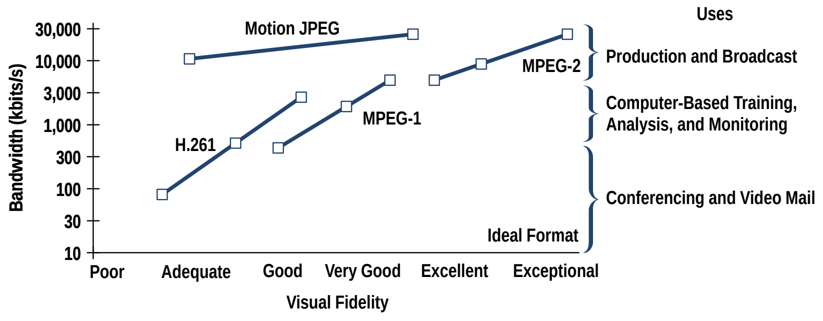

Fig. 2. Compressed video bandwidth versus subjective visual fidelity. The ideal format achieves exceptional visual fidelity at the lowest bandwidth.

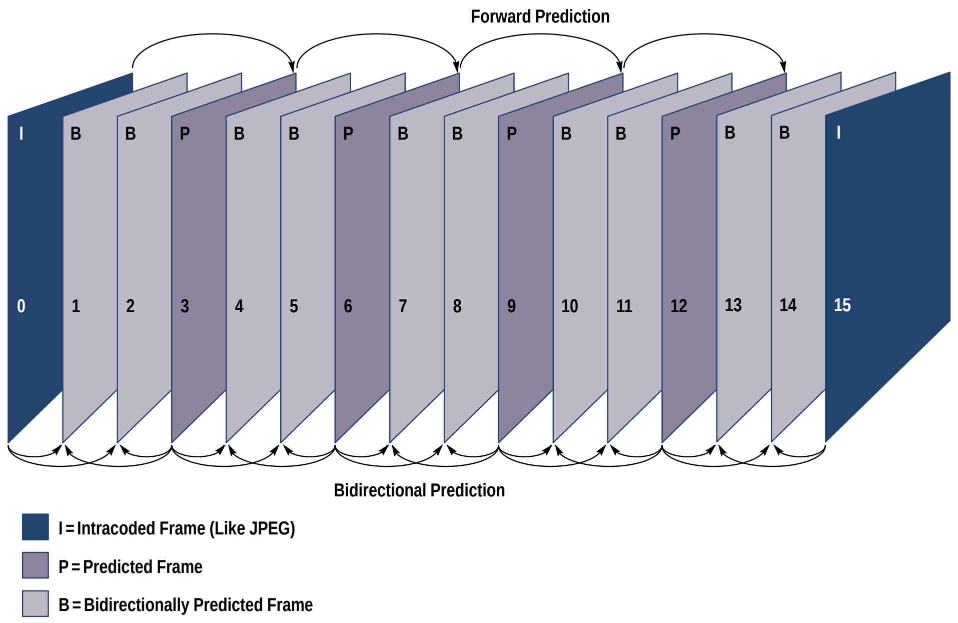

Fig. 3. MPEG frame sequencing.

Nonintracoded frames are further divided into *P-frames* and *B-frames.* P-frames are predicted frames based on comparisons with an earlier reference frame (an intracoded or predicted frame). By considering temporal redundancy in addition to spatial redundancy, P-frames can be encoded with fewer bits. B-frames are bidirectionally predicted frames that require one backward reference frame and one forward reference frame for prediction. A reference frame can be an I-frame or a P-frame, but not a B-frame. By detecting the motion of blocks from both a frame that occurred earlier and a frame that will be played back later in the video sequence, B-frames can be encoded in fewer bits than I- or P-frames.

Each frame is divided into macroblocks of 16 by 16 pixels for the purposes of motion estimation† in MPEG compression and motion compensation in MPEG decompression. A frame with only I-blocks is an I-frame, whereas a P-frame has P-blocks or I-blocks, and a B-frame has B-blocks, P-blocks, or I-blocks. For each P-block in the current frame, the block in the reference frame that matches it best is identified by a motion vector. Then the differences between the pixel values in the matching block in the reference frame and the current block in the current frame are encoded by a discrete cosine transform.

The color space used is the YCbCr color representation rather than the RGB color space, where Y represents the luminance (or brightness) component, and Cb and Cr represent the chrominance (or color) components. Because human perception is more sensitive to luminance than to chrominance, the Cb and Cr components can be subsampled in both the x and y dimensions. This means that there is one Cb value and one Cr value for every four Y values. Hence, a 16-by-16 macroblock contains four 8-by-8 blocks of Y, and only one 8-by-8 block of Cb and one 8-by-8 block of Cr values (see Fig. 4). This is a reduction from the twelve 8-by-8 blocks (four for each of the three color components) if Cb and Cr were not subsampled. The six 8-by-8 blocks in each 16-by-16 macroblock then undergo transform coding.

† Motion estimation uses temporal redundancy to estimate the movement of a block from one frame to the next.

Transform coding concentrates energy in the lower frequencies. The transformed data values are then quantized by dividing by the corresponding quantization coefficient. This results in discarding some of the high-frequency values, or lower-frequency but low-energy values, since these become zeros. Both transform coding and quantization enable further compression by run-length encoding of zero values.

Finally, the nonzero coefficients of an 8-by-8 block used in the discrete cosine transform can be encoded via variable-length entropy encoding such as Huffman coding. Entropy encoding basically removes coding redundancy by assigning the code words with the fewest number of bits to those coefficients that occur most frequently.

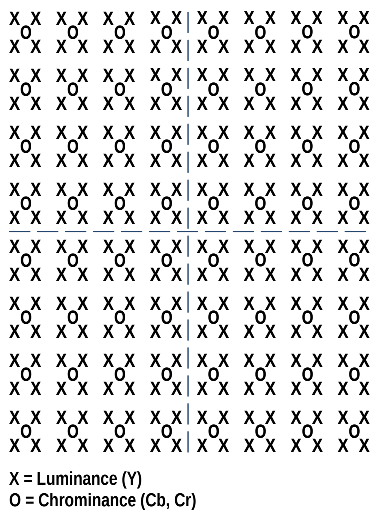

Fig. 4. Subsampling of the chrominance components (Cb, Cr) with respect to the luminance (Y) component.

MPEG Decompression

MPEG decompression reverses the functional steps taken for MPEG compression. There are six basic steps involved in MPEG decompression.

1. The MPEG header is decoded. This gives information such as picture rate, bit rate, and image size.

2. The video data stream is Huffman or entropy decoded from variable-length codes into fixed-length numbers. This step includes run-length decoding of zeros.

3. Inverse quantization is performed on the numbers to restore them to their original range.

4. An inverse discrete cosine transform is performed on the 8-by-8 blocks in each frame. This converts from the frequency domain back to the original spatial domain. This gives the actual pixel values for I-blocks, but only the differences for each pixel for P-blocks and B-blocks.

5. Motion compensation is performed for P-blocks and B-blocks. The differences calculated in step 4 are added to the pixels in the reference block as determined by the motion vector for P-blocks and to the average of the forward and backward reference blocks for B-blocks.

6. The picture is displayed by doing a color conversion from YCbCr coordinates to RGB color coordinates and writing to the frame buffer.

Methodology

Our philosophy was to improve the algorithms and tune the software first, resorting to hardware support only if necessary. We set a goal of 10 to 15 frames/s for software MPEG video decompression because this is the rate at which motion appears smooth rather than jerky.

We started by measuring the performance of the MPEG software we had purchased. This software initially took two seconds to decode one frame (0.5 frame/s) on an older 50-MHz Model 720 workstation. This decoding was for video only and did not include audio. Profiling indicated that the inverse discrete cosine transform (step 4) took the largest chunk of the execution time, followed by display (step 6), followed by motion compensation (step 5). The decoding of the MPEG headers was insignificant.

With this data we set out to optimize every step in the MPEG decompression software. After we applied all the algorithm enhancements and software tuning, we measured the MPEG decode software again. While we had achieved an order of magnitude improvement, the rate of 4 to 5 frames/s was not sufficient to meet our goal.

Hence, we looked at possible multimedia enhancements to the basic PA-RISC processor and other system-level enhancements that would not only speed up MPEG decoding, but also be generally useful for improving performance in other computations. In addition, any chip enhancements we added could not adversely impact the design schedule, complexity, cycle time, and chip size of the PA-RISC processor we were targeting, the PA 7100LC, which was already deep into its implementation phase at the time. The PA 7100LC is described in detail in the article on page12.

We approached this problem by studying the distribution of operations executed by the software MPEG decoder. Then, we found ways to reduce the execution time of the most frequent operation sequences. The application of algorithm enhancements, software tuning, and projected hardware enhancements was iterated until we attained our goal of being able to decompress at a rate greater than 15 frames/s via software.

Algorithm and Software Optimizations

In terms of MPEG video algorithms, we improved on the Huffman decoder, the motion compensation, and the inverse discrete cosine transform. A faster Huffman decoder based on a hybrid of table lookup and tree-based decoding is used. The lookup table sizes were chosen to reduce cache misses. For motion compensation, we sped up the pixel averaging operations.

For the inverse discrete cosine transform, we use a faster Fourier transform, which significantly reduces the number of multiplies for each two-dimensional 8-by-8 inverse discrete cosine transform. In addition, we use the fact that the 8-by-8 inverse transform matrices are frequently sparse to further reduce the multiplies and other operations required.

The MPEG audio decompression is also done in software. This algorithm was improved by using a 32-point discrete cosine transform to speed up the subband filtering.[12]

In terms of software tuning, we "flattened" the code to reduce the number of procedure calls and returns, and the frequent building up and tearing down of contexts present in the original MPEG code. We also did "strength reductions" like reducing multiplications to simpler operations such as shift and add or table lookup.

The last column of Table I shows the percentage of execution time spent in each of the six MPEG decompression steps after the algorithm and software tuning improvements were made. The first two columns of Table I show the millions of instructions executed in each of the six decompression steps and the percent of the total instructions executed (path length) each step represents. The input video sequence was an MPEG-compressed clip of a football game. The total time taken was 7.45 seconds on an HP 9000 Model 735 99-MHz PA-RISC workstation, with 256K bytes of instruction cache and 256K bytes of data cache.

Table I
Instructions and Time Spent in each MPEG Decompression Step on an HP 9000 Model 735

	Millions of Instructions	Path Length (%)	Time (%)
Header decode	0.6	0.1	0.1
Huffman decode	55.3	10.2	7.5
Inverse quantization	8.7	1.6	2.4
Inverse discrete cosine transform	206.5	38.3	38.7
Motion compensation	79.9	14.8	18.3
Display	188.7	35.0	33.0
Total	539.7	100.0	100.0

The largest slice of execution time (38.7%) and the largest chunk of instructions executed (38.3%) were still the inverse discrete cosine transform. We studied the frequencies of generic operations in this group and attempted to execute them faster. This resulted in new PA-RISC processor instructions for accelerating multimedia software.

PA-RISC Processor Enhancements

The new processor multimedia instructions implemented in the PA 7100LC processor allow simple arithmetic operations to be executed in parallel on subword data in the standard integer data path. In particular, the integer ALU is partitioned so that it can execute a pair of arithmetic operations in a single cycle with a single instruction. The arithmetic operations accelerated in this way are add, subtract, average, shift left and add, and shift right and add. The latter two operations are effective in implementing multiplication by constants.

PA-RISC Multimedia Extensions 1.0. The PA 7100LC PA-RISC processor chip contains some instructions that operate independently and in parallel on two 16-bit data fields within a 32-bit register. These operations are independent in that bits carried or shifted out of one of the fields never affects the result in the other field. These operations occur in parallel in that a single instruction computes both 16-bit fields of the result. Table II summarizes these instructions.

HADD does two parallel 16-bit additions on the left and the right halves of registers ra and rb, placing the two 16-bit results into the left and right halves of register rt.

HSUB does two parallel 16-bit subtractions on the left and right halves of registers ra and rb, placing the two 16-bit results into the left and right half of register rt.

Both HADD and HSUB perform modulo arithmetic (modulus 2^{16}), that is, the result wraps around from the largest number back to the smallest number and vice versa. This is the usual mode of operation of twos complement adders when overflow is ignored.

HADD and HSUB also have two saturation arithmetic options. With the signed saturation option, HADD.ss, both operands and the result are considered signed 16-bit integers. If the result cannot be represented as a signed 16-bit integer, it is clipped to the largest positive value (2^{15}–1) if positive overflow occurs, or it is clipped to the smallest negative value ($–2^{15}$) if negative overflow occurs.

With the unsigned saturation option, HADD.us, the first operand (ra) is considered an unsigned 16-bit integer, the second operand (rb) is considered a signed 16-bit integer, and the result (in rt) is considered an unsigned 16-bit integer. If the result cannot be represented as an unsigned 16-bit integer, it is clipped to the largest unsigned value (2^{16}–1) if positive overflow occurs, or it is clipped to the smallest unsigned value (0) if negative overflow occurs.

The signed saturation and unsigned saturation options for parallel halfword subtraction are defined similarly.

HAVE, or halfword average, gives the average of each pair of halfwords in ra and rb. It takes the sum of parallel halfwords and does a right shift of one bit before storing each 16-bit result into rt. During the one-bit right shift, the carry is shifted in on the left and unbiased rounding* is performed on the least-significant bit on the right. Because the carry is shifted in, no overflow can occur in the HAVE instruction.

Table II
PA-RISC Multimedia Instructions in PA 7100LC
ra contains a1; a2
rb contains b1; b2
rt contains t1; t2

Instruction	Parallel Operation

HADD ra,rb,rt	t1 = (a1+b1) mod2^{16};
	t2 = (a2+b2) mod2^{16};

HADD.ss ra,rb,rt	t1 = IF (a1+b1) $>$ (2^{15}–1) THEN (2^{15}–1)
	ELSEIF (a1+b1) $<$ -2^{15} THEN (-2^{15})
	ELSE (a1+b1);
	t2 = IF (a2+b2) $>$ (2^{15}–1) THEN (2^{15}–1)
	ELSEIF (a2+b2) $<$ -2^{15} THEN (-2^{15})
	ELSE (a2+b2);

HADD.us ra,rb,rt	t1 = IF (a1+b1) $>$ (2^{16}–1) THEN (2^{16}–1)
	ELSEIF (a1+b1) $<$ 0 THEN 0
	ELSE (a1+b1);
	t2 = IF (a2+b2) $>$ (2^{16}–1) THEN (2^{16}–1)
	ELSEIF (a2+b2) $<$ 0 THEN 0
	ELSE (a2+b2);

HSUB ra,rb,rt	t1 = (a1–b1) mod2^{16};
	t2 = (a2–b2) mod2^{16};

HSUB.ss ra,rb,rt	t1 = IF (a1–b1) $>$ (2^{15}–1) THEN (2^{15}–1)
	ELSEIF (a1–b1) $<$ -2^{15} THEN (-2^{15})
	ELSE (a1–b1);
	t2 = IF (a2–b2) $>$ (2^{15}–1) THEN (2^{15}–1)
	ELSEIF (a2–b2) $<$ -2^{15} THEN (-2^{15})
	ELSE (a2–b2);

HSUB.us ra,rb,rt	t1 = IF (a1–b1) $>$ (2^{16}–1) THEN (2^{16}–1)
	ELSEIF (a1–b1) $<$ 0 THEN 0
	ELSE (a1–b1);
	t2 = IF (a2–b2) $>$ (2^{16}–1) THEN (2^{16}–1)
	ELSEIF (a2–b2) $<$ 0 THEN 0
	ELSE (a2–b2);

HAVE ra,rb,rt	t1 = (a1+b1)/2;
	t2 = (a2+b2)/2;

HSLkADD ra,k,rb,rt	t1 = (a1$\ll$k) + b1;
	t2 = (a2$\ll$k) + b2;
	(for k = 1, 2, or 3)

HSRkADD ra,k,rb,rt	t1 = (a1$\gg$k) + b1;
	t2 = (a2$\gg$k) + b2;
	(for k = 1, 2, or 3)

ss = signed saturation option
us = unsigned saturation

HSLkADD, or halfword shift left and add, allows one operand to be shifted left by k bits (where k is 1, 2, or 3) before being added to the other operand.

HSRkADD, or halfword shift right and add, allows one operand to be shifted right by k bits (where k is 1, 2, or 3), before being added to the other operand.

Both HSLKADD and HSRKADD use signed saturation.

* Unbiased rounding means that the net difference between the true averages and the averages obtained after unbiased rounding is zero if the results are equally distributed in the result range.

Saturation Arithmetic. In saturation arithmetic a result is said to have a positive overflow if it is larger than the largest value in the defined range of the result. It is said to have a negative overflow if it is smaller than the smallest value in the defined range of the result. If the saturation option is used for the HADD and HSUB instructions, the result is clipped to the maximum value in its defined range if positive overflow occurs and to the minimum value in its defined range if negative overflow occurs. This further speeds up the processing because it replaces using about ten instructions to check for positive and negative overflows and performs the desired clipping of the result for a pair of operations in one instruction.

Saturation arithmetic is highly desirable in dealing with pixel values, which often represent hues or color intensities. It is undesirable to perform the normal modulo arithmetic in which overflows wrap around from the largest value to the smallest value and vice versa. For example, in 8-bit pixels, if 0 represents black and 255 represents white, a result of 256 should not change a white pixel into a black one, as would occur with modulo arithmetic. In saturation arithmetic, a result of 256 would be clipped to 255.

Effect on MPEG Decoding. These parallel subword arithmetic operations significantly speed up several critical parts of the MPEG decoder program, especially in the inverse discrete cosine transform and motion compensation steps. More than half of the instructions executed for the inverse transform step are these parallel subword arithmetic instructions. Their implementation does not impact the processor's cycle time, and adds less than 0.2% of silicon area to the PA 7100LC processor chip. Actually, the area used was mostly empty space around the ALU, so that these multimedia enhancements can be said to have contributed to more efficient area utilization, rather than adding incremental chip area. See "Overview of the Implementation of the Multimedia Enhancements" on page 66.

Since the PA 7100LC processor has two integer ALUs, we essentially have a parallelism of four halfword operations per cycle. This gives a speedup of four times, in places where the superscalar ALUs can be used in parallel. Because of the built-in saturation arithmetic option, speedup of certain pieces of code is even greater.

System Optimization

The second longest functional step (see Table I) in MPEG decompression was the display step. Here, we leveraged the graphics subsystem to implement the color conversion step together with the color recovery already being done in the graphics chip.[7] Color conversion converts between color representations in the YCbCr color space and the RGB color space. Color recovery reproduces 24-bit RGB color that has been color compressed into 8 bits before being displayed. Color compression allows the use of 8-bit frame buffers in low-cost workstations to achieve almost the color dynamics of 24-bit frame buffers. This leveraging of low-level pixel manipulations close to the frame buffer between the graphics and video streams also contributed significantly to the attainment of real-time MPEG decompression. Color recovery and the graphics chip are described in the articles on pages 51 and 43, respectively.

Other PA 7100LC processor enhancements streamline the memory-to-I/O path. By having the memory controller and the I/O interface controller integrated in the PA 7100LC chip, overhead in the memory-to-frame-buffer bandwidth is reduced. Overhead in the processor-to-graphics-controller-chip path is also reduced for both control and data.

Path Length Reduction

Table III shows the same information as Table I but for the low-end Model 712 workstation which uses the multimedia-enhanced PA 7100LC processor and the graphics chip mentioned above.

Table III
Instructions and Time Spent in each MPEG Decompression Step on a Model 712 Workstation

	Millions of Instructions	Path Length (%)	Time (%)
Header decode	0.60	0.2	0.3
Huffman decode	55.0	16.1	14.5
Inverse quantization	8.9	2.6	4.5
Inverse discrete cosine transform	138.5	40.6	34.4
Motion compensation	74.8	21.9	25.6
Display	63.0	18.5	20.7
Total	340.8	100.0	100.0

The Model 712 executes consistently fewer instructions than the Model 735 for the same MPEG decompression of the same video clip. It is also faster in MPEG decompression even though it operates at only 60% of the 99-MHz rate of the high-end Model 735 and has only one eighth of the cache size. This shows the performance benefits from the path length reduction enabled by the PA-RISC processor and system enhancements for multimedia acceleration.

Performance

The performance of the PA-RISC architectural enhancements and the leveraging of the graphics subsystem for video decompression can be seen in Fig. 5. This data is for a

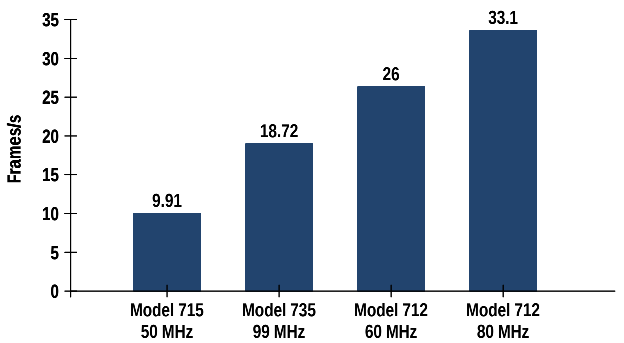

Fig. 5. Maximum MPEG decode frame rates for different models of HP 9000 Series 700 workstations. These rates are for a 352-by-240-pixel clip that was encoded at 30 frames/s.

Overview of the Implementation of the PA 7100LC Multimedia Enhancements

One goal in adding the multimedia instructions was to minimize the amount of new circuits to be added to the existing ALUs and to minimize the impact on the rest of the CPU. This goal was accomplished. The only circuit changes to the CPU were in the ALU data path and decoder circuits. These instructions reuse most of the existing functionality and very small modifications and additions were required to implement them.

All of the new instructions implemented require two 16-bit adds or subtracts to be done in parallel. The existing ALU adder was modified to provide this functionality. These instructions required that the existing 32-bit adder be conditionally split into two 16-bit halves without sacrificing the performance of the 32-bit add. Conceptually this is equivalent to blocking the carry from bit 16 to bit 15 in a ripple-carry adder. To accomplish this, we made the following modifications.

The ALU adder is similar to a carry lookahead adder. The first stage of the adder calculates a carry generate and a carry propagate signal for each single bit in the adder. In this case, 32 single-bit generate and 32 single-bit propagate signals are calculated. These single-bit carry generate and carry propagate signals are used in subsequent stages of the carry chain to calculate carry generate and carry propagate signals for groups of bits.

The 32-bit adder was divided into two 16-bit halves between bits 15 and 16 by providing alternate signals for the carry generate and carry propagate signals from bit 16 (Fig. 1). The new generate and propagate signals from bit 16 are created with a two-input multiplexer. When a 32-bit addition or subtraction is being performed, the multiplexer selects the original generate and propagate signals to be passed onto the next stage of the carry chain. When 16-bit addition or subtraction is being performed the multiplexer selects the value for generate and propagate from the second input which is false (logical 0) for additions and true (logical 1) for subtractions.

The new generate and propagate signals can be forced to be false for instructions requiring halfword addition. This stops the carry from being generated by bit 16 or propagating from bit 16 to bit 15, even if this generate and propagate signal is not used directly to calculate the carry signal (as is the case in this adder). The generate and propagate signals can also be forced to be true for instructions requiring halfword subtraction. This will force a carry into the more significant halfword of the adder by generating a carry from bit 16 into bit 15. This technique is used along with the ones complement of the operand to be subtracted to perform subtraction as twos complement addition.

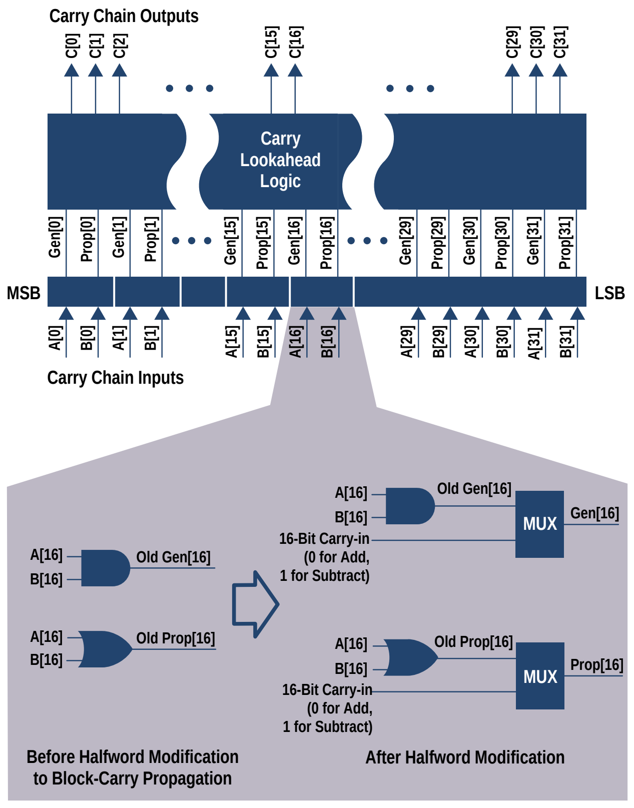

Fig. 1. Modifications to the carry lookahead adder to accommodate the halfword instructions.

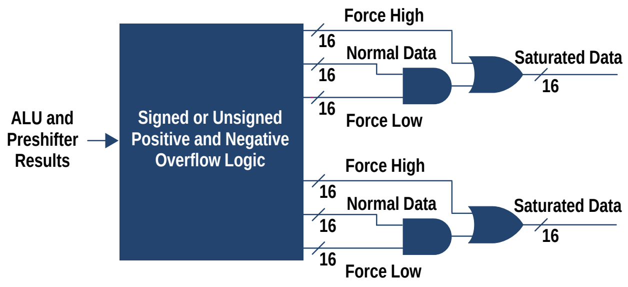
ALU and Preshifter Results
Signed or Unsigned Positive and Negative Overflow Logic
Force High
Normal Data
Force Low
16
Saturated Data

Fig. 2. Saturation logic. There is one of the these circuits for each halfword.

The original carry generate and propagate signals from bit 16 are still generated to calculate overflows from the less significant halfword addition. This overflow is used by the saturation logic, which can be invoked by some of these instructions.

Saturation requires groups of bits of the result to be forced to states of true or false, or passed unchanged. This is accomplished with an AND-OR gate (Fig. 2). The AND function can force the output of the gate to be false and the OR function can force the output of the gate to be true. Thus, the output is either forced high, forced low, or forced neither high nor low. It is never simultaneously forced high and low. The key is to determine when to force the result to a saturated value.

The saturation circuit is added at the end of the ALU's data path after the result selection multiplexer selects one of the results from the adder after it performs additions, subtractions, or logical operations such as bitwise AND, OR, or XOR (Fig. 3). The saturation circuit does not impact the critical speed paths of the ALU because it is downstream from the point where the cache data address is driven from the adder and where the test condition logic (i.e., logic for conditional branch instructions) obtains the results from which to calculate a test condition.

If signed saturation is selected, the ALU will force any 16-bit result that is larger than 0x7fff to 0x7fff (2^{15}–1) and any 16-bit result that is smaller than 0x8000 to 0x8000 (–2^{15}). These conditions represent positive and negative overflow of signed numbers. Positive and negative overflow can be detected by examining the sign bit (the MSB) of each operand and the result of the add. If both operands are positive and the result is negative then a positive overflow has occurred and the result in this case is saturated by forcing the most-significant bit to a logical 0 and the rest of the bits to a logical 1. If both operands are negative and the result is positive then a negative overflow has occurred and the result in this case is saturated by forcing the most significant bit to a logical 1 and the rest of the bits to a logical 0. Unsigned saturation is implemented in a similar way.

The average instruction, HAVE, requires manipulating the result after the addition is finished. Before the implementation of the halfword instructions the ALU selected between the results of a bitwise AND, a bitwise OR, a bitwise XOR, or the sum of the two input operands. The halfword average instruction adds an additional choice. The average result is the sum of the two input operands shifted right one bit position with a carry out of the most-significant bit (MSB) becoming the MSB of the result. To perform rounding of the result, the least-significant bit (LSB) of the result is replaced by an OR of the two least-significant bits before shifting right one bit.

The shift right and add and the shift left and add functions were added by modifying the x-bus preshifter in the operand selection logic of the ALU. The original ALU was capable of shifting 32-bit inputs left by zero, one, two, or three bits. To implement the 16-bit shift left and add instructions, the left-shift circuits had to be broken at

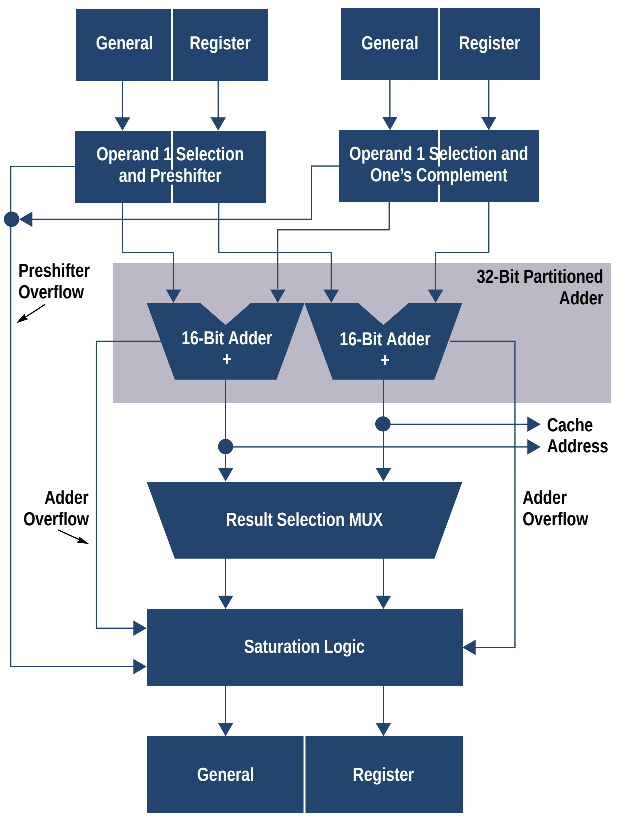

Fig. 3. Flow of halfword instructions showing the location of the saturation logic in relation to the ALU.

the halfword boundary. This was done by ANDing the bits shifted from the least-significant halfword to the most-significant halfword with a control signal that indicates when a 32-bit shift is being done. The 16-bit shift right and add instructions were implemented by adding the ability to shift one, two, or three bits right. This shift is always broken at the halfword boundary.

One challenging aspect of implementing the 16-bit shift left and add instructions was detecting when the results of shifting an operand left by one, two, or three bits causes a positive or negative overflow. A positive overflow occurs when the unshifted operand is positive and a logical one is shifted out of the left, or when the result of the shift is negative. A negative overflow occurs when the unshifted operand is negative and a logical zero bit is shifted out of the left, or when the result of the shift is positive. These overflow conditions are combined with the overflows calculated by the adder and used to saturate the final result. The final result is saturated if either the left shift or the adder causes an overflow.

The result of selecting instructions that can provide the most useful functionality while costing the least to implement was a relatively small increase in the area of the ALU. About 15% of the ALU's area is devoted to halfword instructions. Since the ALU's circuits were the only ones modified on the processor chip, only about 0.2% of the total processor's chip area is devoted to halfword instructions.

video clip that was compressed at 30 frames/s. The Model 715 and Model 735 are based on the PA 7100 processor. The Model 712 is based on the PA 7100LC processor, which is a derivative of the PA 7100. The PA 7100LC contains the multimedia enhancements and system integration features and is described in the article on page 12. The older, high-end Model 735 running at 99 MHz achieves 18.7 frames/s while the newer entry-level Model 712 achieves 26 frames/s at 60 MHz and 33.1 frames/s at 80 MHz. These frame decompression rates are quoted for MPEG video only (no audio) with

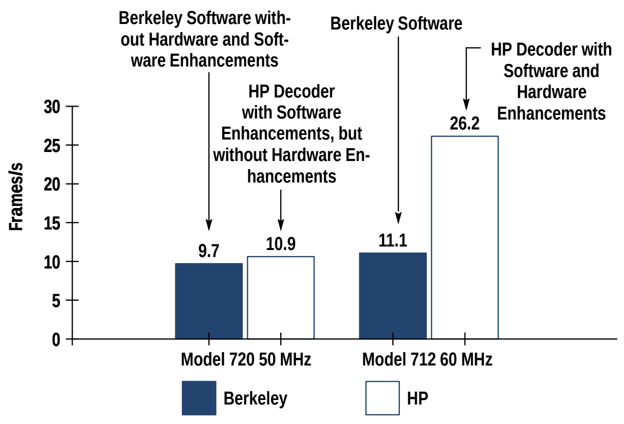

Fig. 6. Comparison between the performance of the enhanced Berkeley MPEG decoder and the HP MPEG decoder (without audio).

no constraints on how fast the decoding can proceed. In other words, the decoding rate is not constrained by the rate at which the MPEG stream has been compressed. Hence, although the video clip used was MPEG compressed at 30 frames/s, the 80-MHz Model 712 can decode it faster than 30 frames/s in unconstrained mode. This implies that there is some processor bandwidth left after achieving real-time software MPEG video decoding.

In the video player product in HP MPower 2.0, frames are skipped if the decoder cannot keep up with the desired real-time rate. This results in a lower effective frame rate, since skipped frames are not counted, even though execution time may have been used for partial decoding of a skipped frame.

Fig. 6 shows a comparison between the enhanced Berkeley software MPEG decoder and the HP software MPEG decoder running on the older HP 9000 Model 720 (with no hardware multimedia enhancements) and the newer Model 712 workstation (with hardware multimedia enhancements). The fourth column in Fig. 6 illustrates the performance obtainable with synergistic software and hardware enhancements.

In the Model 720, the Berkeley and HP software decoders have comparable performance. For the Model 712, the performance of the HP decoder was 2.4 times greater than the Berkeley decoder because of the synergistic coupling of the algorithms and software optimized with the PA-RISC multimedia instructions and the system-level enhancements in the Model 712.

Fig. 7 shows the performance when MPEG audio of various fidelity levels is also decompressed by software running on the general-purpose PA 7100LC processor. The highest-fidelity audio is stereo with no decimation. This means that every audio sample comes as a pair of left and right channel values, and every sample is used. Half decimation means that one out of every two audio samples is used. (3/4 decimation means that only one out of every four audio samples is used.) Mono means that every audio sample is a single value (channel) rather than a pair of values.

While software decompression of MPEG audio degrades the performance in terms of frames decoded per second, the PA 7100LC-based workstations achieved rates of 15.1 frames/s at 60 MHz, 24.2 frames/s at 80 MHz, and 27.4 frames/s at

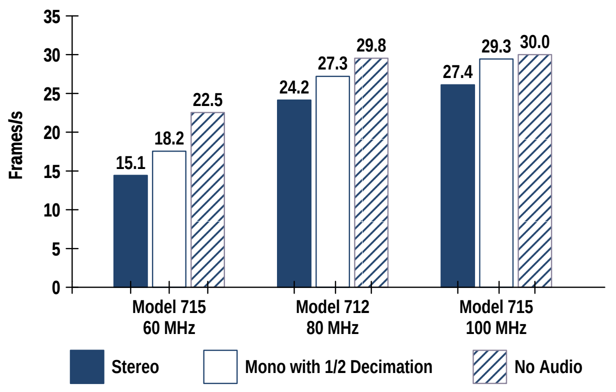

Fig. 7. Performance when MPEG video and MPEG audio are decoded in software.

100 MHz even with the highest-fidelity 44.1-kHz stereo 16-bit linear audio format with no decimation. With further enhancements of audio decoding and audio-video synchronization, we should be able to do even better.

Conclusion

We wanted a software approach to MPEG decoding because we felt that if video is to be useful it has to be pervasive, and to be pervasive, it should exist at the lowest incremental cost on all platforms. With a software video decoder, there is essentially no additional cost. In addition, the evolving standards and improving algorithms pointed to a flexible solution, like software running on a general-purpose processor. Using special-purpose chips designed for MPEG decoding, or even for JPEG, MPEG, and H.261 compression and decompression, would not allow one to take advantage of improved algorithms and adapt to evolving standards without buying and installing new hardware.

Furthermore, since the performance of general-purpose microprocessors continues to improve with each new generation, we wanted to be able to leverage these improvements for multimedia computations such as video decompression. This approach also allows us to focus hardware design efforts on improving the performance of the general-purpose processor and system without having to replicate performance efforts in each special-purpose subsystem, such as the graphics and video subsystems. The PA-RISC multimedia instructions are also useful for graphics, image, and audio computations, or any computations requiring arithmetic on a lot of numbers with precision less than 16 bits.

The net result is that we achieve real-time MPEG decoding of video streams at 30 frames/s with a software decoder. This was achieved by a synergistic combination of algorithm enhancements, software tuning, PA-RISC processor multimedia enhancements, combining video and graphics support for color conversions and color compression, and system tuning. The PA-RISC multimedia enhancements allow parallel processing of pixels in the standard integer data path at an insignificant addition to the silicon area. The total area used is less than 0.2% of the PA 7100LC processor chip with no impact on the cycle time or the control complexity.

The real-time software MPEG decoding rate of the final video player product exceeds our original goal of 10 to 15 frames/s for a software-based MPEG video decoder. It is also significant that MPEG video decoding at 30 frames/s is achieved by an entry-level rather than a high-end workstation. This is in the context of a full-function video player on the HP MPower 2.0 product. With MPEG audio decoding (also done by software), the frame rate is usually above 15 frames/s, even for the low-end Model 712/60 workstation, and around 24 frames/s for the Model 712/80 workstation.

We expect to see continuous improvement in the MPEG decoding rate as the performance of the general-purpose processors increases. With PA-RISC processors, there has been roughly a doubling of performance every 18 to 24 months. This would imply that larger frames sizes, multiple video streams, or MPEG–2 streams may be decoded in the future by such multimedia-enhanced general-purpose processors.

Acknowledgments

The success of the MPEG software decoder performance tuning depended on close, interdivisional teamwork distributed across four HP sites at Chelmsford, Cupertino, Palo Alto, and Fort Collins. We would like to thank all the other members of the multimedia architecture team, especially Vasudev Bhaskaran, Peter Kaczowka, Behzad Razban, Pat McElhatton, Larry Thayer, Konstantine Konstantinides, and Larry McMahan. We would also like to thank the HP MPower team, the PA-RISC extensions team, especially Michael Mahon, the PA 7100LC team, especially Mark Forsyth and Charlie Kohlhardt, the Model 712 team, the performance lab, especially the late Tian Wang, and the compiler lab, especially Pat Kwan, for their support.

References

1. *Hewlett-Packard Journal,* Vol. 45, no. 2, April 1994.
2. R. Lee, "Precision Architecture," *IEEE Computer,* Vol. 22 no. 1, January 1989, pp. 78-91.
3. R. Lee, M. Mahon, and D. Morris, "Pathlength Reduction Features in the PA-RISC Architecture," *Proceedings of IEEE Compcon,* February 1992, pp. 129-135.
4. L. McMahan and R. Lee, "Pathlengths of SPEC Benchmarks for PA-RISC, MIPS and SPARC," *Proceedings of IEEE Compcon,* February 1993, pp. 481-490.
5. *Digital Compression and Coding of Continuous-Tone Still Images,* CCIT REC. T.81 0918-1, July 1992.
6. P. Knebel, et al, "HP's PA7100LC: A Low-Cost Superscalar PA-RISC Processor," *Proceedings of IEEE Compcon,* February 1993, pp. 441-447.
7. S. Undy, et al, "A VLSI Chip-Set for Graphics and Multimedia Workstations," *IEEE Micro,* Vol. 14, no. 2, April 1994, pp. 10-22.
8. L. Gwennap, "New PA-RISC Processor Decodes MPEG Video," *Microprocessor Report,* Vol. 8, no. 1, January 1994, pp . 16-17.
9. *Coding of Moving Pictures and Associated Audio for Digital Storage Media up to 1.5 Mbit/s*, ISO/IEC JTCI CD 11172, 1991.
10. K. Patel, B. Smith, and L. Rowe, "Performance of a Software MPEG Video Decoder," *Proceedings of First ACM International Conference on Multimedia,* August 1993, pp. 75-82.
11. D. LeGall, "MPEG – A Video Compression Standard for Multimedia Applications," *Communications of the ACM,* April 1991, Vol. 34 no. 4, pp 46-58.
12. K. Konstantinides, "Fast Subband Filtering in MPEG Audio Coding," *IEEE Signal Processing Letters,* Vol. 1, no. 2, February 1994, pp. 26-28.