# Accelerating Multimedia with Enhanced Microprocessors

**A minimalistic set of multimedia instructions introduced into PA-RISC microprocessors implements SIMD-MIMD parallelism with insignificant changes to the underlying microprocessor. Thus, a software video decoder attains MPEG video and audio decompression and playback at real-time rates of 30 frames per second, on an entry-level workstation. Our general-purpose parallel subword instructions can accelerate a variety of multimedia programs.**

***Ruby B. Lee***

*Hewlett-Packard*

Multimedia is the integration of visual, audio, textual, and sensory information (see Figure 1). It is basically information represented in different ways by different media datatypes. Multimedia information can facilitate more natural human-to-computer interactions, enhance communication, shorten learning time, or lessen misinterpretation. While computers have successfully supported other media datatypes like text, graphics, images, and audio, high-fidelity video, as a datatype, remains challenging because it consumes enormous storage, bus, network, and computation resources.

For video to be a viable datatype in today's computer, it must be compressed. In turn, "reading" a video object requires decompressing and displaying it in real time. Until recently, the high computational complexity of high-fidelity video decompression (such as MPEG video, the informal name of the Moving Pictures Experts Group's video compression and decompression standard) required special-purpose chips or boards.

To significantly accelerate this step using software, we introduced a small set of multimedia instructions into a general-purpose PA-RISC microprocessor. These instructions enable for the first time an entry-level workstation to achieve MPEG video decompression and playback at real-time rates of 30 frames per second (fps), using a software video player. They do not require a DSP (digital signal processor), coprocessor, or special functional units. Rather, we enhanced the existing microprocessor data paths to enable parallel operations on subwords, that is, data narrower in width than the width of a word in the microprocessor.

Unlike special-purpose instructions designed specifically for MPEG video decompression, our parallel subword instructions perform general-purpose operations like add, subtract, average, and shift_and_add, in parallel. They accelerate many types of programs (multimedia and others) running in microprocessors, without the need for additional hardware.

Multimedia processing with additional circuitry on the microprocessor is sometimes referred to as native signal processing (NSP) in contrast to the use of special DSP chips, which are frequently used to process media. Proposed here is a type of NSP that is even more integrated into the design of the microprocessor. Since it uses the basic functional data paths of the microprocessor, we call it ISP (intrinsic signal processing).

The small set of PA-RISC multimedia instructions, together with synergistic software and hardware optimizations, allowed us to achieve real-time MPEG video decompression and playback (July 1993, lab; January 1994, product). No other product or research project has successfully accomplished real-time MPEG decoding by software.[1] (Recently, other software MPEG players



0272-1732/95/$04.00 © 1995 IEEE

| | | |
|---|---|---|
| Visual information: | Images | Video |
| | Graphics | Animation |
| Audio information: | Voice | Music |
| Textual information: | Keyboard | Handwriting |
| Sensory information: | Sensors | Controllers |

Figure 1. Spectrum of multimedia datatypes, from simpler forms to more complex forms on the right.

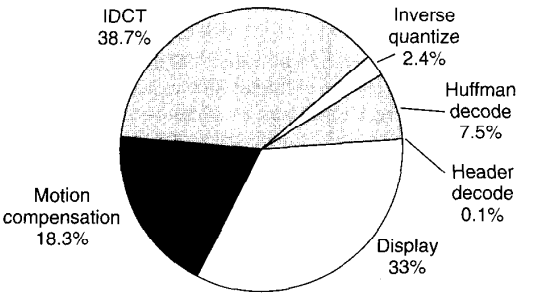


Figure 2. Execution time distribution of MPEG decoder steps for an nfl.mpg video clip.

have reportedly run close to real-time rates, but they either did not decode the full MPEG bitstream or required high-end workstations, for example, the 275-MHz Alpha.)

## The software approach

Studies indicated that users were accustomed to at least TV and VCR video fidelity, so the lower fidelity video decompression networks currently produced in software would not be acceptable for many users. Instead, we chose to implement MPEG-1, which achieves the desired video fidelity at the lowest bit rate among competing video compression algorithms. We also set a goal of 10- to 15-fps video playback, since this is the rate at which motion begins to appear smooth rather than jerky.

We felt that software, leveraging the basic hardware platform, would facilitate the pervasiveness, flexibility, and low cost of a new multimedia datatype like video. By software, we mean programming in high-level languages like C, using standard compiler and debugging support, and resorting to low-level assembly code only for a few critical code kernels, if necessary. First, we tried to improve the algorithms and tune the software, resorting to hardware support only if necessary.

Rather than add special-purpose MPEG circuitry to the microprocessor, we followed the same design principles used in selecting instructions for the original PA-RISC architecture.[2-4] This involved finding the most frequent operations, breaking them down into simple primitives, and accelerating their execution. The result is that we have multimedia instructions that are useful not only for accelerating MPEG video decompression but also for many other computations.

Our achievement of real-time MPEG decompression through software is a technology breakthrough in the support of high-fidelity video on desktops. That we could achieve this breakthrough with such a minimal set of generally useful instructions added to a microprocessor, the PA-7100LC,[5-7] intended for entry-level desktops, is also interesting.

## Multimedia benchmarks

In the past, designers optimized processor architecture based on either technical or commercial programs. Since multimedia information processing is likely to become increasingly important, we realized the necessity of including multimedia benchmarks in the design of processors. Audio and 3D graphics computations are well served by the single-precision floating-point instructions found in microprocessors. We discuss MPEG video decompression (see box, next page) as an example.

MPEG achieves roughly the video quality of today's analog TVs and VCRs, at around 1.5 Mbits per second. This standard uses Standard Interchange Format (SIF)-size frames of 352×240 pixels. MPEG compression takes an uncompressed video stream and audio stream, and compresses them into a stream of video and audio packets. MPEG playback reads them from a CD-ROM, disk, or network; decompresses the video (and audio) packets; displays the video on the computer's monitor; and plays back the audio on speakers or headphones.

Figure 2 shows the execution time spent in the six major steps of the optimized MPEG decompression software running on an older PA-RISC processor without any multimedia enhancements. (This was a 99-MHz PA-RISC workstation, the HP735, with 256 Kbytes each of instruction and data cache). The distribution of execution time varies for different video streams.

In most video clips we viewed, the IDCT step, even after considerable optimization, still consumed the largest chunk of execution time. For Figure 2's nfl.mpg clip, this amounted to 38.7 percent of the total execution time. This fraction was often larger in other video clips. The next two largest time consumers are the display step followed by motion compensation. Fortunately, the two inherently serial steps, decoding the MPEG headers and Huffman decoding, were relatively insignificant in execution time.

The IDCT was a prime candidate for speedup with processor multimedia enhancements. It has a great deal of parallelism that we could slice in many different ways, in its processing of 8×8 blocks of pixels.

At 30 fps, with YCbCr color components, we can decom-

## MPEG compression and decompression

The MPEG-1 standard (called MPEG in this article) describes two classes of frames: intracoded and non-intracoded.[1,2] Intracoded frames, or I-frames, are compressed by exploiting spatial redundancy within the frame itself. I-frames do not depend on comparisons with other reference frames. They use JPEG (the Joint Photographic Experts Group's standard) style compression for still images.[3]

MPEG divides non-intracoded frames into P- and B-frames. P-frames are predicted frames, based on comparisons with an earlier reference frame. By considering temporal redundancy in addition to spatial redundancy, MPEG allows P-frame encoding with fewer bits. B-frames are bidirectionally predicted frames, using one backward and one forward reference frame. A reference frame can be an I- or a P-frame, but not a B-frame. By detecting motion of blocks from both a frame that occurred earlier and a frame that will be played back later in the video sequence, we can encode B-frames in even fewer bits.

MPEG further divides each frame into macroblocks of 16×16 pixels for 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- or I-blocks, and a B-frame has B-, P-, or I-blocks. For each P-block in the current frame, a motion vector identifies the block in the reference frame that best matches it. Then a discrete cosine transform (DCT) encodes the differences between the pixel values in the matching block in the reference frame and the current block in the current frame.

MPEG uses the YCbCr, rather than the RGB (red, green, blue), color representation. Here, 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, we can subsample the Cb and Cr components in both the $X$ and $Y$ dimensions. This means that we have one Cb value and one Cr value for every four $Y$ values. Hence, a 16×16 macroblock contains four 8×8 blocks of $Y$ and only one 8×8 block each of Cb and Cr. This is a reduction from the twelve 8×8 blocks (four for each of the three color components), if the Cb and Cr were not subsampled. The six 8×8 blocks in each 16×16 macroblock then undergo transform coding.

Transform coding concentrates energy in the lower frequencies. By dividing by the corresponding quantization coefficient, the transformed data values are quantized. This results in the discarding of some 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, we can encode the nonzero coefficients of an 8×8 block used in the DCT via variable-length, entropy encoding, as in Huffman coding. Entropy encoding removes coding redundancy by assigning the code words with the fewest number of bits to those coefficients that occur most frequently.

MPEG decompression reverses the functional steps taken for compression. Decompression (Figure A) involves six basic steps.

pose each 8×8 IDCT into eight independent, one-dimensional IDCTs on the rows, followed by eight independent, 1D IDCTs on the columns. This implies sixteen 1D IDCTs per 8×8 block, or almost a million (30×1.5×1,320×16 = 950,400) 1D IDCTs per second.

Each 1D, 8-point IDCT itself has room for parallel operations. Hence, we can perform parallel operations within a 1D IDCT, across 1D IDCTs, across 2D 8×8 IDCTs of either one-color component or multiple-color component blocks, or even across frames. This indicated that parallel processing of some sort could be extremely beneficial in improving MPEG and other multimedia applications.

Other popular video and image compression and decompression standards also use the IDCT, and clearly benefit from speeding up this step in the MPEG decoder. (The other standards are the CCITT H.261 for video teleconferencing; MPEG-2 for higher fidelity, higher bandwidth compressed video; and JPEG for image compression.)

## PA-RISC enhancements

We chose to adhere to RISC design principles, rather than add complex, special-purpose instructions to the processor's instruction repertoire. This involved understanding the basic operations in performance-critical portions of the code and finding ways to speed them. For example, Figure 3 shows a breakdown of the instructions executed in the IDCT step in MPEG decompression.

Key observations about the multimedia benchmarks we used in designing the PA-RISC are that

- a great deal of parallelism exists in MPEG and other pixel-oriented algorithms;
- the data being operated on were small integers, narrower than the existing integer data paths of microprocessors; and
- the most common operations were add, subtract, and simple forms of multiplication and division.

## MPEG compression and decompression (continued)

- *Header decode.* Provides video sequence parameters such as picture rate, bit rate, and image size.
- *Huffman decode.* Decodes variable-length codes into fixed-length numbers, which represent quantized inverse DCT (IDCT) coefficients, scaling factors, and motion vectors. This step includes run-length decoding of zeros.
- *Inverse quantization.* Multiplies coefficients by quantizer coefficients to restore them to their original range.
- *IDCT.* Changes each 8×8 block of IDCT coefficients to convert the data 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- and B-blocks.
- *Motion compensation.* Adds the differences in the IDCT step 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.
- *Display.* Converts color from YCbCr coordinates to RGB color coordinates, including upsampling Cb and Cr values, and writing to the frame buffer for displaying the decoded video.

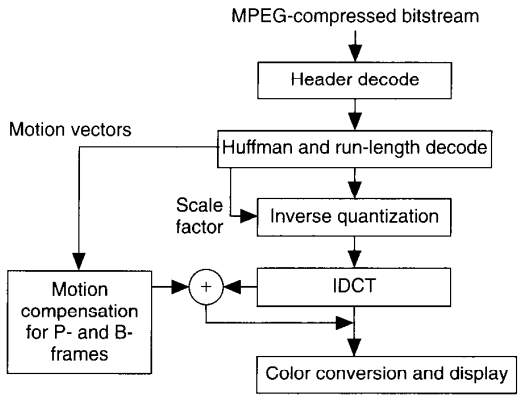


Figure A. Steps in MPEG decoding.

### References


1. *ISO/IEC JTC1 CD 11172, Information Technology—Coding of Moving Pictures and Associated Audio for Digital Storage Media up to 1.5 Mbits/s; Part 2: Coding of Moving Picture Information*, International Standards Organization/International Electrotechnical Commission, Geneva, 1991.
2. D. LeGall, "MPEG—A Video Compression Standard for Multimedia Applications," *Commun. ACM*, Vol. 34, No. 4, Apr. 1991, pp. 46-58.
3. *CCITT Rec. T.81 10918-1, Information Technology: Digital Compression and Coding of Continuous-Tone Still Images*, Comité Consultatif International de Téléphonique et Télégraphique, Geneva, July 2, 1992.


This suggested the possibility of partitioning existing functional units, to execute parallel operations on multiple pairs of subword data. By subword data, we mean data sizes less than 32 bits in a 32-bit microprocessor and less than 64 bits in a 64-bit microprocessor.

**SIMD parallelism**. We decided to introduce SIMD (single-instruction, multiple-data) parallelism[8] into the microprocessor, without violating RISC design principles. SIMD represents a type of parallel computer in which a control processor dispatches a common instruction to multiple data processors, each of which performs the instruction on its own pair of data items.

Previous SIMD machines were very large, complex, and expensive parallel supercomputers such as the Illiac IV, which provided each processor with its own memory subsystem. We have now brought these same SIMD concepts into a single-chip, general-purpose microprocessor, where the parallel data items are subwords packed into standard-size words. The data processors are just partitions of existing or new functional units, and the control processor is the normal instruction fetch and dispatch unit. The parallel memory subsystems feeding the data processors are the usual word fetch mechanism in a standard microprocessor: from the single memory into gen-

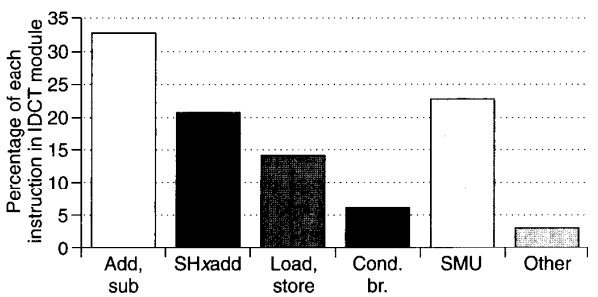


Figure 3. Instruction class frequencies in IDCT step for input file nfl.mpg. There are 206 million instructions in the IDCT module.

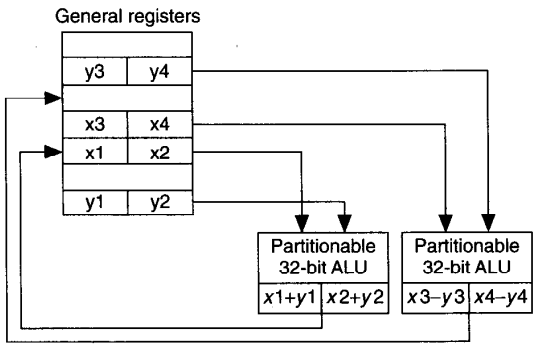


Figure 4. SIMD-MIMD subword parallelism in the superscalar PA-7100LC.

eral registers and functional units. The "SIMD instruction" here is a standard, 32-bit RISC processor instruction. The microprocessor required no other pipeline, register, or memory changes.

Useful subword sizes in a 64-bit-word computer are four 16-bit halfwords, eight 8-bit bytes, or two 32-bit words. The processor carries out parallel operations on these subwords with a single SIMD-style, parallel subword instruction, using a standard 64-bit functional unit like an ALU. Current 32-bit integer functional units are also amenable to parallel 16-bit or 8-bit operations.

While many pixel-oriented data start out and end up as 8-bit color components, their intermediate processing requires precision greater than 8 bits. IDCT values are 12 bits, and since applications such as medical imaging require 12-bit pixels, the most useful subword size in many multimedia applications appears to be 16 bits. While 8-bit parallel operations are useful for low-end video and graphics algorithms, higher fidelity video algorithms like MPEG need 16-bit subword precision.

**SIMD-MIMD parallelism.** In fact, superscalar PA-RISC processors implement an even more sophisticated type of parallelism, SIMD-MIMD. MIMD (multiple-instruction, multiple-data) means that different operations may be performed in parallel (by different data processors) on many pairs of data. Many microprocessors today are superscalar, in that they can execute more than one instruction per pipeline cycle. Superscalar execution, coupled with parallel subword instructions, is a RISC-like embodiment of SIMD-MIMD parallelism.

Figure 4 shows this type of SIMD-MIMD parallelism implemented in the superscalar PA-7100LC processor, which has two 32-bit integer ALUs. For example, two different multimedia instructions—or one multimedia instruction and a non-multimedia instruction (a load instruction, for instance)—can execute in parallel. A SIMD-MIMD processor offers more opportunities for parallel execution than if all the parallel operations had to be identical, as in a SIMD-only machine.

We chose the following actual operations for these parallel subword, SIMD-style instructions to be implemented in the PA-7100LC. They are based on the frequency with which they occur in easily parallelizable computations, such as the IDCT (see Figure 3). Table 1 summarizes these Multimedia Extensions. For the 32-bit PA-7100LC processor, each multimedia instruction involves two parallel subword operations (n = 2). For 64-bit PA-RISC 2.0 processors, each instruction involves four parallel, 16-bit operations (n = 4). PA-RISC 2.0 processors also have a larger set of multimedia instructions.

**Parallel addition and subtraction.** In many multimedia programs, the most frequent operations are simple additions and subtractions. This is true, for example, in the IDCT, as illustrated in Figure 3, which shows the breakdown of the IDCT program step, in terms of basic operation types. Since the data being decompressed was 12-bit signed integers, the program did not use the entire 32-bit integer data path efficiently.

We decided to allow the ALU to be partitionable into 16-bit chunks, so that two parallel 16-bit operations could execute in parallel, using the existing ALU with a minor modification. This consisted of blocking the carry from the low halfword to the high halfword, when parallel adds or subtracts executed. In the 32-bit PA-7100LC, each ALU can now perform two parallel halfword adds (or subtracts) in the same time it takes to perform a single 32-bit add, that is, a single cycle. In 64-bit PA-RISC 2.0 processors,[9] each ALU can perform four 16-bit adds (or subtracts) in a single cycle. Since most PA-RISC processors today have at least two ALUs, this means that four 16-bit ALU operations can execute in a 32-bit architecture like the PA-7100LC (see Figure 4 again), and eight in a 64-bit architecture like the PA8000, in a single cycle. This essentially increases by either four or eight times the peak execution bandwidth for multimedia applications that can use parallel 16-bit arithmetic, at negligible incremental hardware cost.

**Parallel multiplication primitives.** We considered including a 16-bit×16-bit multiplier, a sizable piece of circuitry, to complement the parallel 16-bit add and subtract instructions. While other multimedia applications, such as audio and modem code, use multiplication as frequently as additions, this is not the case in many video algorithms.

In particular, we can multiply by constants effectively without requiring a full multiplier circuit. PA-RISC compilers successfully do so as a series of shift and add instructions. The PA-RISC architecture has always provided a shift_left_and_add instruction that shifts one operand left by 1, 2, or 3 bits, before adding the other operand.[2] This was needed for load and store instructions with indexed addressing by unit sizes greater than bytes (that is, 16-bit halfwords, 32-bit words, and 64-bit doublewords). The preshifter balanced the

complementer used for subtraction on the other input port to the ALU and hence did not add to the cycle time of the processor.

For multimedia acceleration, we added parallel 16-bit shift_left_and_add instructions, as well as parallel 16-bit shift_right_and_add instructions with the same shift amounts of 1, 2, or 3 bits. The former provided useful multiplication by integer constants, while the latter was more useful for multiplication by fractional constants.

Multiplication by fractional constants, either with no integer component, or only a very small integer component, is common. Shifting a value $x$ right by 1 bit is equivalent to dividing it by 2, shifting it right by 2 bits is equivalent to $x/4$, and shifting it right by 3 bits is equivalent to $x/8$. So, by a small sequence of such shift_right_and_add sequences, we could synthesize multiplication of $x$ by any fractional constant, with or without a small integer part. Since the adder in the ALU is already partitioned to allow parallel adds and subtracts, the only incremental change required for implementing the parallel shift_and_add instructions was that of partitioning the preshifter input to the adder. This again did not cause cycle time impact in the PA-7100LC.

**Parallel averages.** In several imaging and video algorithms, including the motion compensation step of MPEG decompression, we often need to find the arithmetic average of two pixels. We observed that this is merely an add followed by a right shift of one bit. Since we already have parallel 16-bit add operations, we quite easily performed parallel 16-bit averages as well.

The beauty of an average instruction is that no overflow is possible. The carry generated from the add operation shifts in on the left, as the most significant bit of the result, at the same time that the least significant bit of the sum shifts out on the right. However, rather than just truncate the result by shifting out the least significant bit on the right, we round the result, to preserve accuracy in a sequence of cascaded averages. This rounding function is a simple OR of the two least significant bits of the sum before the right shift of one bit. It performs a round-to-odd function applied to integers. (In the past, rounding functions were discussed only for floating-point mantissas.)

The advantage of this simple rounding function is that the net difference between the true averages and the rounded averages is zero, if the results are uniformly distributed over the result range. From an implementation perspective, this rounding causes no additional delay, since there is at least one gate's delay between the time the two least significant bits of the sum are generated and the time the carry bit out of the sum is generated.

**Table 1. PA-RISC Multimedia Extensions 1.0.**

| Instruction | Parallel operation |
|---|---|
| HADD ra,rb,rt | t1=(a1+b1)mod$2^{16}$; ... tn=(a2+b2)mod$2^{16}$ |
| HADD,ss ra,rb,rt<br>(with signed saturation option) | t1=IF(a1+b1)>($2^{15}$–1)THEN($2^{15}$–1)<br>ELSEIF(a1+b1) ≤ $2^{15}$THEN(–$2^{15}$)<br>ELSE(a1+b1); ...<br>tn=IF(a2+b2)>($2^{15}$–1)THEN($2^{15}$–1)<br>ELSEIF(a2+b2) ≤ $2^{15}$THEN(–$2^{15}$)<br>ELSE(a2+b2) |
| HADD,us ra,rb,rt<br>(with unsigned saturation option) | t1=IF(a1+b1)>($2^{16}$–1)THEN($2^{16}$–1)<br>ELSEIF(a1+b1)<0 THEN 0<br>ELSE(a1+b1); ...<br>tn=IF(a2+b2)>($2^{16}$–1)THEN($2^{16}$–1)<br>ELSEIF(a2+b2)<0 THEN 0<br>ELSE(a2+b2) |
| HSUB ra,rb,rt | t1=(a1–b1)mod$2^{16}$; ... tn=(a2–b2)mod$2^{16}$ |
| HSUB,ss ra,rb,rt<br>(with signed saturation option) | t1=IF(a1–b1)>($2^{15}$–1)THEN($2^{15}$–1)<br>ELSEIF(a1–b1) ≤ $2^{15}$THEN(–$2^{15}$)<br>ELSE(a1–b1);...<br>tn=IF(a2–b2)>($2^{15}$–1)THEN($2^{15}$–1)<br>ELSEIF(a2–b2) ≤ $2^{15}$THEN(–$2^{15}$)<br>ELSE(a2–b2) |
| HSUB,us ra,rb,rt<br>(with unsigned saturation option) | t1=IF(a1–b1)>($2^{16}$–1)THEN($2^{16}$–1)<br>ELSEIF(a–b1)<0 THEN 0<br>ELSE(a1–b1); ...<br>tn=IF(a2–b2)>($2^{16}$–1)THEN($2^{16}$–1)<br>ELSEIF(a2–b2)<0 THEN 0<br>ELSE(a2–b2) |
| HAVE ra,rb,rt | t1=(a1+b1)/2; ... tn=(a2+b2)/2<br>with round_to_odd |
| HSHLADD ra,k,rb,rt | t1=(a1<<k)+b1; ... tn=(a2<<k)+b2<br>(for k=1, 2, or 3) , with signed saturation |
| HSHRADD ra,k,rb,rt | t1=(a1>>k)+b1; ... tn=(a2>>k)+b2<br>(for k=1, 2, or 3), with signed saturation |

a1 and a2 are 16-bit data packed into the 32-bit register ra; register rb contains b1 and b2; rt contains t1 and tn. n=2 for 32-bit data paths and n=4 for 64-bit data paths.

**Average parallel overflows.** Unlike the parallel average instructions, the parallel add, subtract, and shift_and_add

instructions can all generate overflows. A 64-bit architecture has four 16-bit results with four possible overflows. We have at least five options for handling these overflows:

- ignore them,
- trap on any overflow,
- save an overflow flag if at least one overflow occurred,
- save all four overflow flags, and
- clip the result to the desired range.

Ignoring the trap is known as performing "modular arithmetic," or "wrap-around arithmetic." We implement this option as the default parallel subword add or subtract option. However, in certain situations we need to know that an overflow has occurred or incorrect data would propagate without any indication of an error.

Trapping on any overflow is fine, except that it requires two versions of each instruction: one that traps on overflow and one that does not. Many times modular arithmetic is desired, and trapping is the incorrect action.

Saving one or multiple overflow flags is an extra state that adds to the cost of the implementation. The program then checks that the overflow flag or flags are set and handles the overflows, if necessary.

The last alternative, clipping to the desired result range, is saturation arithmetic. In most video and graphics algorithms, this clipping occurs in implementations that use mechanisms for trapping on overflow or checking that an overflow flag is set.

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 has a *negative overflow* if it is smaller than the smallest value in the defined range of the result. If a saturation option accompanies the parallel add or subtract instruction, 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.

We have two saturation options: signed and unsigned. With the signed saturation option, both operands and the result have signed, 16-bit numbers in the range $(-2^{15}, 2^{15}-1)$. The result of an add operation with signed saturation could have a positive overflow only if both operands are positive, in which case the result would be clipped to a maximum value of $2^{15}-1$. It could have a negative overflow only if both operands are negative, in which case the result will be clipped to a minimum value of $-2^{15}$.

The second type of saturation supported is more unusual. Unsigned saturation corresponds directly to that of signed saturation when both operands and the result are unsigned, 16-bit numbers from 0 to $2^{16}-1$. While such a definition of unsigned saturation may be useful for parallel, 8-bit arithmetic, we could not find many uses for it in parallel, 16-bit arithmetic, where most often at least one of the operands is signed. Moreover, such a definition of unsigned saturation does not allow the clipping of negative numbers to zero, which is highly desirable in many applications, as described later.

Our definition of unsigned saturation allows operations with mixed signed and unsigned numbers, and saturation of a signed number to an unsigned number: One operand is an unsigned 16-bit number from 0 to $2^{16}-1$. The other operand is a signed 16-bit number from $-2^{15}$ to $2^{15}-1$. The result is an unsigned 16-bit number from 0 to $2^{16}-1$.

Saturation arithmetic further speeds up the processing of multimedia data. About five instructions must be used to check for positive and negative overflows and perform the desired clipping of one result. When saturation arithmetic is used, each multimedia instruction in the PA-7100LC generates two 16-bit clipped results and replaces 10 ordinary PA-RISC instructions. Each multimedia instruction in 64-bit PA-RISC 2.0 processors generates four parallel, 16-bit results, replacing around 20 instructions when saturation arithmetic is used.

**Uses of saturation arithmetic.** Saturation arithmetic is useful when dealing with pixel values, which often represent hues or color intensities. It is undesirable to perform the normal modular 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 modular arithmetic. In saturation arithmetic, a result of 256 would be clipped to 255. In 16-bit arithmetic, an unsigned 8.8 fixed-point number, with 8 bits of pixel value and 8 fractional bits may represent a pixel. We can then add some signed number to it and still clip the result to an unsigned 8.8 number, using our definition of unsigned saturation.

The two saturation options also allow efficient clipping to arbitrary maximum and minimum values in the defined range of the result. We first figure the difference between the new maximum (or minimum) value and the maximum (or minimum) defined values to which the result saturates. To clip to a new maximum value, xmax, we need two instructions: First, add this difference to the result, then subtract this same difference. If the result is less than xmax, these two operations cancel each other out. However, if the result is greater than xmax, the first instruction causes saturation to occur, and the second instruction brings this saturated maximum value down to the new maximum value, xmax (Figure 5a).

We use a similar pair of instructions to clip to a new minimum value (Figure 5b). In fact, to clip to both new maximum and new minimum values, we need only three instructions, rather than four (Figure 5c), since we can combine two subtract operations into one. Furthermore, if the new minimum value is zero (that is, clipping to an unsigned number), we need only two instructions (Figure 5d), since we can use the unsigned saturation option in the second instruction to clip to zero.

If xmax is greater than or equal to $2^{15}-1$ and xmin is zero

(that is, clipping a signed 16-bit number to an unsigned 16-bit number), we need just one instruction, with unsigned saturation (Figure 5e). This adds zero to the value in Rx with unsigned saturation, causing negative results to be clipped to zero.

## Performance

Table 2 summarizes the key implementation parameters (processor used, frequency, and cache sizes), and performance metrics (integer and floating-point SPECmarks based on the SPEC92 benchmark suites) of several of the workstations mentioned here. The rows are sorted according to the PA-RISC processor used and its frequency (second column). The PA-7100LC processor is leveraged from the PA-7100 design, for lower cost products. In addition to the multimedia instructions described earlier, the PA-7100LC processor also integrates controllers for memory, cache, and I/O.[6,7] Referring to the workstation rather than just the processor is important since the entire workstation system design contributes to the overall performance, not just the processor design.

**a) Clip to a maximum result value, xmax, less than ($2^{15}$–1):**

```
HADD,ss  Rx, Ry, Rx;   Ry contains [(2^15–1) – xmax], clip Rx results to (2^15–1) at
                         high end
HSUB,ss  Rx, Ry, Rx;   Rx results are at most xmax at high end, unchanged at
                         low end (clipped to –2^15)
```

**b) Clip to a minimum result value, xmin, greater than $-2^{15}$:**

```
HSUB,ss  Rx, Ry, Rx;   Ry contains (2^15–xmin), clip Rx results to –2^15 at low end
HADD,ss  Rx, Ry, Rx;   Rx results are at least xmin at low end, unchanged at
                         high end [clipped to (2^15–1)]
```

**c) Clip to a new maximum and minimum value, so that the result is in the range (xmin, xmax):**

```
HADD,ss  Rx, Ry, Rx;   Ry contains [(2^15–1) – xmax], clip Rx results to (2^15–1) at
                         high end
HSUB,ss  Rx, Rz, Rx;   Rz contains [(2^15–1) – xmax + (2^15–xmin)], results are
                         clipped to –2^15 at low end
HADD,ss  Rx, Rw, Rx;   Rw contains (2^15–xmin), results are at most xmax at high
                         end and at least xmin at low end.
```

**d) Clip a signed 16-bit integer to an unsigned integer in the range (0, xmax), where xmax less than ($2^{15}$–1):**

```
HADD,ss  Rx, Ry, Rx;   Ry contains the value [(2^15–1) – xmax], clip Rx results to
                         (2^15–1)
HSUB,us  Rx, Ry, Rx;   Rx results are at most xmax at high end, and clipped to 0
                         at low end
```

**e) Clip a signed 16-bit integer to an unsigned integer in the range (0, xmax), where xmax is greater than or equal to ($2^{15}$–1):**

```
HADD,us  Rx, R0, Rx;
```

Figure 5. Using saturation for clipping to arbitrary maximum and minimum values.

**Path length reduction.** The performance of an application is inversely proportional to its execution time, which is a product of path length, average cycles executed per instruction (CPI), and the cycle time of the processor:

$$\text{Performance} = \frac{1}{\text{path length} \times \text{CPI} \times \text{cycle time}}$$

**Table 2. Implementation and performance characteristics of the referenced workstations, using abbreviated names; for example, HP 9000 Model 720 appears as HP720.***

| Workstation | Processor (MHz) | Multimedia-enhanced | External I/D cache sizes (Kbytes) | Integer SPECmark | Floating-point SPECmark | Mo./yr. introduced |
|---|---|---|---|---|---|---|
| HP720 | PA-7000 (50) | No | 128 I, 256 D | 38.5 | 66.1 | 3/91 |
| HP715/50, HP725/50 | PA-7100 (50) | No | 64 I, 64 D | 35.3 | 66.8 | 3/93 |
| HP735 | PA-7100 (99) | No | 256 I, 256 D | 109.1 | 167.9 | 11/92 |
| HP712/60 | PA-7100LC (60) | Yes | 32 I, 32 D | 58.1 | 84.9 | 1/94 |
| HP712/80 | PA-7100LC (80) | Yes | 128 I, 128 D | 84.3 | 122.3 | 1/94 |
| HP715/100, HP725/100 | PA-7100LC (100) | Yes | 128 I, 128 D | 99.6 | 137.0 | 5/94 |

*Some HP715 models use the nonmultimedia-enhanced PA-7100 and others use the multimedia-enhanced PA-7100LC. The HP725 models have the same performance as the equivalent HP715 models, except that they have more I/O slots.

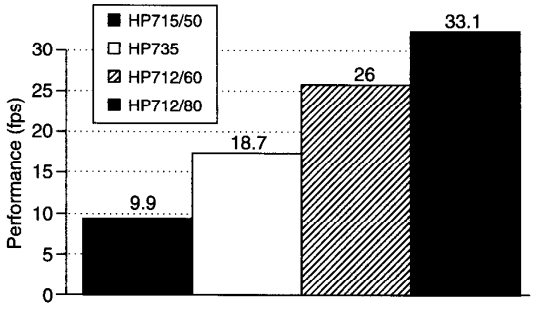


**Figure 6. Maximum MPEG video decode performance for input file cats.mpg (352×240 pixels, encoded at 30 fps) on different workstations.**

It is well recognized that reducing the cycle time, or equivalently, increasing the MHz rate of the processor, increases its performance. RISC processors have also emphasized the importance of reducing the CPI for improved performance. It is not as well recognized that we can get significance performance boost by merely reducing the number of instructions that need to execute, that is, reducing the path length of the program.

From the very beginning of the PA-RISC architectural design, we strove to incorporate path length reduction features into the architecture,[2-4] without violating the principles of RISC design. We now add parallel subword arithmetic instructions, SIMD-style, for the acceleration of multimedia programs, again following the same design principles. By performing more operations in a single instruction, without adding undue complexity to the pipeline, we can significantly reduce the path length of the program (for example, MPEG decompression) without increasing the CPI or cycle time. This in turn, improves the performance of the software program.

For example, in the IDCT routine, the multimedia-enhanced HP712 has a much shorter path length than the HP735. We reduced the number of instructions required for adds, subtracts, shift_and_adds, loads, and branches (the first four columns in the earlier Figure 3). We also reduced the add, subtract, and shift_and_add instructions by parallel subword arithmetic operations. We reduced the load and conditional branch operations, since a word load instruction now brings in multiple parallel pieces of subword data. Reduced conditional branches result from the reduced number of loop iterations. Figure 3's fifth column (SMU) refers to shift, extract, and deposit instructions which another functional unit, the shift-merge unit, executes. Multimedia instructions accelerating SMU instructions allow parallel data alignment and rearrangement to be performed on the data packed into registers. All PA-RISC 2.0 processors, including the PA8000, implement these instructions.[9]

These reductions result in a PA-7100LC-based workstation (with multimedia instructions) requiring only 138.5 million instructions for all the IDCT executions. A PA-7100-based workstation (without multimedia instructions) requires 206.5 million instructions.[10] Overall, the MPEG decoder required only 340.8 million instructions running on the PA-7100LC processor with multimedia instructions, whereas it required 539.7 million instructions on a PA-7100 processor. This resulted in the 60-MHz HP712 being faster in the MPEG decoding of the same video clip than the 99-MHz HP735—even though the latter has eight times the cache size—and a 65 percent faster processor clock rate. Using SPEC92 performance ratings, we would have expected the HP712/60 to display only about half the performance of the HP735 (Table 2).

**Relative MPEG video decompression performance.** Figure 6 shows the performance for the MPEG-1 software decompression of the video clip, cats.mpg. The multimedia-enhanced PA-7100LC-based workstations outperform the older PA-7100-based workstations. For example, the high-end HP735 running at 99 MHz, achieves 18.7 fps, while the multimedia-enhanced, entry-level HP712 workstations produce 26 fps at 60 MHz, and 33.1 fps at 80 MHz.

These are frame decompression rates for MPEG video only (no audio) with no constraints on how fast the decoding may proceed. In other words, the rate at which the MPEG stream had been compressed does not constrain the decoding rate. Although the video clip used was MPEG compressed at 30 fps, the 80-MHz HP712 can decode it faster than 30 fps, in unconstrained mode.

In the actual video player in our MPower 2.0 multimedia user interface, the encoded rate synchronizes (or constrains) the playback rate. It skips frames if the decoder cannot keep up with the real-time (encoded) rate, resulting in a lower effective frame rate, since skipped frames are not counted. On the other hand, task switching or waiting occurs if the decoder is too fast, so that the playback rate is at most the encoded rate, as can be seen in Figure 7.

**MPEG video with MPEG audio software performance.** Figure 7 shows the performance of MPEG video with MPEG audio at different audio fidelity levels when we also used software running on the general-purpose PA-RISC processor to decompress MPEG audio. In this figure, video decompression, audio decompression, system layers, the video player itself, and MPower 2.0 are all running in software.

The leftmost column in each set represents the highest fidelity audio: stereo with no decimation, that is, every audio sample (44.1-kHz, linear, 16-bit stereo audio, for CD-quality music) comes as a pair of left and right channel values, and every sample is used.

The middle column in each set represents average-grade audio. Mono means that every audio sample is a single value (channel) rather than a pair of values. Half decimation means

that one out of every two audio samples is used. The rightmost column in each set occurs when audio decoding is bypassed: The MPEG system layer still decodes the headers of audio packets but discards them.

The reason the rightmost column is slightly lower than the unconstrained video-only numbers in Figure 6 is that the playback is synchronized to the encoded rate. The player synchronizes each frame to play back at 33-millisecond intervals, if the video stream was encoded at 30 fps—as is usual in US television programs. For movies, the MPEG video stream is encoded at 24 fps, which then defines the real-time rate for MPEG playback.

MPEG audio is relatively complex compared to the computational needs of simpler audio standards. Software decompression of MPEG audio (and all the other software product overhead) degrades performance in terms of decoded frames per second. Yet, even when the highest fidelity stereo audio is used, a low-end, 60-MHz PA-RISC HP712 achieves a rate above 15 fps, an 80-MHz HP712 achieves 24.2 fps, and a 100-MHz HP715 achieves 27.4 fps. With further enhancements of audio decoding and audio-video synchronization, we should do even better.

**Performance improvements in other applications.** Figure 8 shows four sample real-time applications coded with and without multimedia instructions. The speedup in frame rates using multimedia instructions is in the range of 1.9 to 2.7. This illustrates that the general-purpose multimedia enhancements introduced into the PA-RISC processors and systems extend beyond enhancing MPEG-1 video decompression to other multimedia applications.

THE PERFORMANCE OF OUR small set of multimedia instructions is remarkable, since it enabled a low-cost workstation, the HP712, to achieve real-time MPEG video playback with software on a microprocessor. In fact, more multimedia instructions and complex structures on the processor chip would not necessarily improve software MPEG decompression. For example, a complex multimedia instruction that executes a subtract, absolute value, and accumulate of parallel subwords would not accelerate MPEG decompression, since this is useful only for motion estimation in some video-*compression* algorithms and not useful in video decompression.

We can view the set of multimedia enhancements present in the PA-7100LC, as described here, as the basic core of subword SIMD instructions making up a minimal set useful for accelerating multimedia programs. All these instructions are used in performing parallel operations in a basic adder circuit. The 64-bit PA-RISC 2.0 architecture implemented by the PA8000 incorporates a superset of multimedia instructions, which would support all the instructions described here and more.[9]

Since the performance of general-purpose microprocessors continues to improve with each new generation, we can leverage these improvements for multimedia computations. PA-RISC processors have roughly doubled performance every 18 to 24 months. This approach also allows us to focus hardware design efforts on improving the performance of the general-purpose processor and system without having

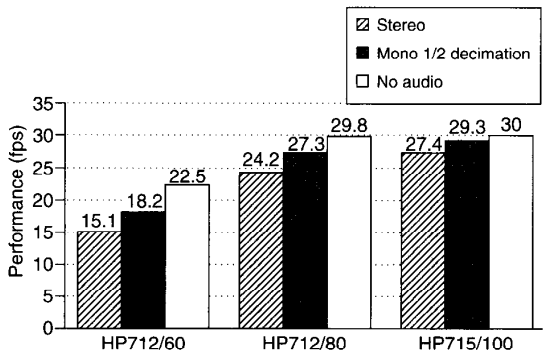


Figure 7. MPEG video with MPEG audio software performance. All three workstations use the multimedia-enhanced PA-7100LC processor.

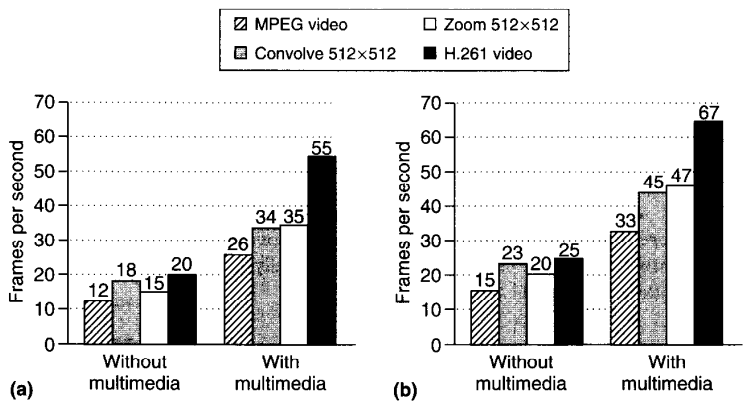


Figure 8. Performance of PA-RISC multimedia instructions for different applications: using the HP712 at 60 Hz (a) and at 80 Hz (b).

to replicate performance efforts in each special-purpose subsystem, such as the graphics or video subsystem.

These multimedia instructions are generally useful for graphics, image, audio, and modem computations, or any computations that can benefit from parallel operations on a large number of integers with precision less than 16 bits.

The PA-RISC multimedia enhancements are yet another evolutionary step in introducing sophisticated, high-performance computer design techniques into the ordinary microprocessor. The SIMD-MIMD parallelism embedded into a superscalar microprocessor does not violate RISC design principles. As mentioned earlier, we used the same RISC principles as in the original PA-RISC design:[2] 1) speeding up the most frequent operations (this time, in a multimedia work load), and 2) streamlining their implementation in a simple, pipelined processor.

We continue to believe in the concept that it is better to implement simple operations in a single cycle, using these to build many different complex functions, than to implement complex instructions that have limited usefulness. Specifically, we resisted putting in complex multicycle instructions, that were MPEG specific. Rather, we added simple, generic operations, like add, subtract, average, and shift_and_add, which have essentially unlimited usefulness in many multimedia and other applications.

Since we made only minor modifications to the two existing integer ALUs, the cost of these instructions is insignificant, amounting to less than 0.2 percent of the silicon area in the PA-7100LC processor chip.[5-7] These multimedia instructions, together with other software and system optimizations, enabled our entry-level workstations to support the lowest cost MPEG video player, since no special-purpose chip or board was required.

We plan to expand multimedia work loads to include higher level multimedia programs such as conferencing, editing, encryption, and recognition. Second-generation multimedia instructions will be part of the PA-RISC 2.0 architecture. These future developments will continue to use synergistic software and hardware optimizations to deliver cost-effective multimedia acceleration.

## Acknowledgments

I thank all members of my interdivisional Multimedia Architecture team; the PA-RISC Extensions team, especially Michael Mahon; and the PA-7100LC team, especially Joel Lamb, Mark Forsyth, and Charlie Kohlhardt.

## References

1. K. Patel, B. Smith, and L. Rowe, "Performance of a Software MPEG Video Decoder," *Proc. First ACM Int'l Conf. Multimedia,* Assoc. Computing Machinery, N.Y., 1993, pp. 75-82.
2. R. Lee, "Precision Architecture," *Computer,* Vol. 22, No. 1, Jan. 1989, pp. 78-91.
3. R. Lee, M. Mahon, and D. Morris, "Pathlength Reduction Features in the PA-RISC Architecture," *Proc. Compcon,* IEEE Computer Society Press, Los Alamitos, Calif., 1992, pp. 129-135.
4. L. McMahan and R. Lee, "Pathlengths of SPEC Benchmarks for PA-RISC, MIPS, and SPARC," *Proc. Compcon,* CS Press, 1993, pp. 481-490.
5. P. Knebel et al., "HP's PA-7100LC: A Low-Cost Superscalar PA-RISC Processor," *Proc. Compcon,* CS Press, 1993, pp. 441-447.
6. S. Undy et al., "A VLSI Chip Set for Graphics and Multimedia Workstations," *IEEE Micro,* Vol. 14, No. 2, Apr. 1994, pp. 10-22.
7. L. Gwennap, "New PA-RISC Processor Decodes MPEG Video," *Microprocessor Report,* Vol. 8, No. 1, Jan. 24, 1994, pp. 16-17.
8. M. Flynn, "Very High-Speed Computing Systems," *Proc. IEEE,* Vol. 54, No. 12, Dec. 1966.
9. D. Hunt, "Advanced Performance Features of the 64-Bit PA8000," *Proc. Compcon,* CS Press, 1995.
10. R. Lee, "Real-Time MPEG Video via Software Decompression on a PA-RISC Processor," *Proc. Compcon, CS Press,* 1995.

**Ruby B. Lee** serves as chief architect of the cross-functional multimedia architecture team and processor architect of the PA-RISC Extensions team at Hewlett-Packard. She is also a consulting associate professor of electrical engineering at Stanford University and co-leader of the joint HP-Intel architecture team.

Lee holds a BA from Cornell University, and an MS in computer science and a PhD in electrical engineering from Stanford University. She holds eight patents in processor architecture, pipeline design, cache hints, branch optimizations, and multimedia. She is a member of the IEEE, ACM, Phi Beta Kappa, and Alpha Lambda Delta.

Address questions concerning this special issue to the author at Computer Systems Architecture, Hewlett-Packard, 19410 Homestead Road, MS43UG, Cupertino, CA 95014; rblee@nsa.hp.com.

## Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156 Medium 157 High 158